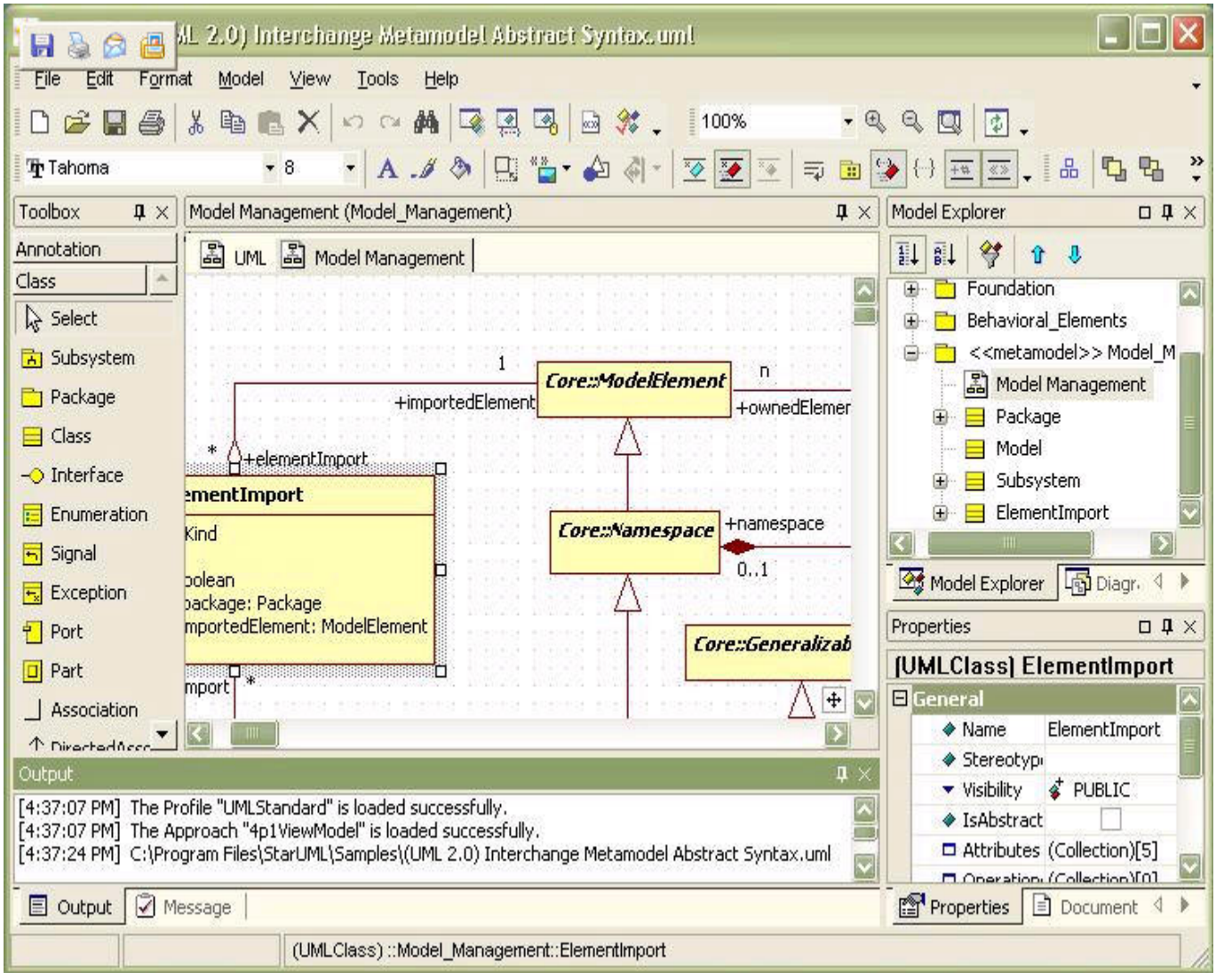
Руководство разработчика

Глава 1. Введение

Руководство разработчика StarUML™ (программной платформы моделирования на основе UML) предоставляет существенную информацию для программистов, позволяющую использовать механизм расширения StarUML™ и разрабатывать аддины (дополнительные модули) для StarUML™.

Краткий обзор StarUML

StarUML™ - программная платформа моделирования, которая поддерживает UML (Унифицированный Язык Моделирования). Она основана на версии UML 1.4 и поддерживает нотацию UML версии 2.0 и одиннадцать различных типов диаграмм. Она активно поддерживает подход MDA (Архитектура Управляемая Моделью) и концепцию профилей UML. StarUML™ превосходен в отношении настройки окружения пользователя и имеет высокую степень расширяемости в том, что касается его функциональных возможностей.



Инструмент UML, который настраивается пользователем

StarUML™ предоставляет максимальные возможности адаптации к требованиям пользователя, предлагая настройку параметров, которые определяют методологию создания программного обеспечения, проектную платформу и язык разработки.

Настоящая поддержка MDA

Разработка методологии описания архитектуры программ - важный процесс, который будет развиваться ещё 10 или более лет. Намерение OMG (Ассоциация объектно-ориентированного управления) состоит в том, чтобы использовать технологию MDA (Модельно-управляемая архитектура) для создания платформенно-независимых моделей, на базе которых можно автоматически формировать платформенно-зависимые модели или программные коды. StarUML™ полностью поддерживает стандарт UML 1.4 и нотацию UML 2.0., а также реализует концепцию профилей UML. Пользователи могут легко получить свои конечные продукты, выполняя через внешние COM интерфейсы готовые сценарии или создавая шаблоны документов.

Превосходная расширяемость и гибкость

StarUML™ обладает превосходной расширяемостью и гибкостью. Он поддерживает использование аддинов[1], позволяющих расширять его функциональные возможности. При их разработке предоставляется доступ ко всем функциям модели/метамодели и другому инструментарию через COM, включая расширение меню и набора опций конфигурации. Также, пользователи могут создавать свои собственные подходы[2] и фреймворки[3], соответствующие специальным методологиям. StarUML™ может также быть интегрирован с любыми внешними инструментальными средствами.

Почему выбрана платформа UML/MDA

StarUML™ - платформа моделирования программ. Почему нужна именно расширяемая платформа моделирования, а не просто инструмент, реализующий UML?

- Конечные пользователи хотят иметь настраиваемые инструментальные средства. Обеспечение возможности настройки параметров, учитывающее все требования к операционной среде, может гарантировать высокую производительность и качество разработок.
- Никакой инструмент моделирования не предоставляет абсолютно исчерпывающий набор всех возможных функций. Поэтому, хороший инструмент должен предусматривать необходимость расширения своих функций в будущем, оправдывая тем самым инвестиционные затраты пользователя, сделанные при его покупке.

1 Термин «аддин» (add-in) означает набор спецификаций расширения StarUML. Аддин может включать автономный программный объект (COM или Windows scripting host), используемый в качестве сервера обработки событий, а также различные спецификации в т.ч. структуры проекта, дополнительных пунктов меню, опций настройки, дополнительных модельных элементов и т.п. Далее в качестве синонима термина «аддин» часто применяется термин «модуль» (module). (прим. пер.)

2 Термин «подход» (approach) здесь означает спецификацию базовой структуры проекта. Элементы подхода (как правило, это пустые пакеты и диаграммы) автоматически генерируются при создании нового проекта. Синонимами понятия «подход» являются термины «шаблон» или «макет», применительно к проекту вцелом. (прим. пер.)

3 Термин «фреймворк» (framework) здесь означает определённым образом оформленный набор готовых UML-спецификаций, которые могут использоваться в проекте. Проект только ссылается на фреймворк, а не интегрирует его в себя, поэтому спецификации фреймворка не могут быть изменены в проекте. (прим. пер.)

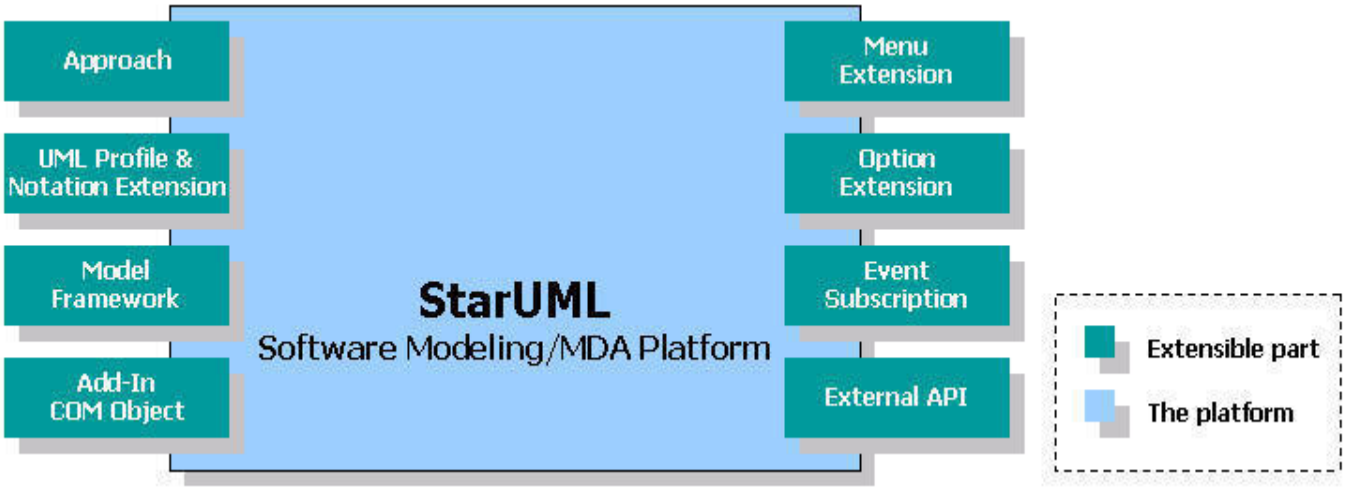
- Технология MDA (Управляемая моделью архитектура) требует не только платформенной независимости, но и мультиплатформенных функциональных возможностей. Инструментальные средства моделирования, ограниченные специфическими средами разработки, не подходят для технологии MDA. Инструмент моделирования сам должен являться платформой моделирования, предоставляющей функциональные возможности, настраиваемые на различные технологии, среды разработки и инструментальные средства.
- Интеграция с другими инструментальными средствами жизненно важна для максимизации эффективности использования инструмента. Инструмент должен предоставлять высокий уровень расширяемости и позволять интегрироваться с существующими инструментальными средствами, которые находятся в распоряжении пользователя.

Глава 2. Архитектура StarUML

Эта глава раскрывает основную архитектуру StarUML™. Она описывает, главным образом, архитектуру самой платформы, аддинов и внешнего программного интерфейса приложения.

Архитектура платформы

StarUML™ - расширяемая платформа моделирования программного обеспечения; она не только предоставляет готовые функции, но позволяет добавлять новые. Диаграмма, представленная ниже, иллюстрирует архитектуру StarUML™. Светлым фоном показана собственно платформа, а тёмным - элементы расширения. Элементы расширения могут быть разработаны пользователем или третьими лицами, а затем интегрированы в платформу.



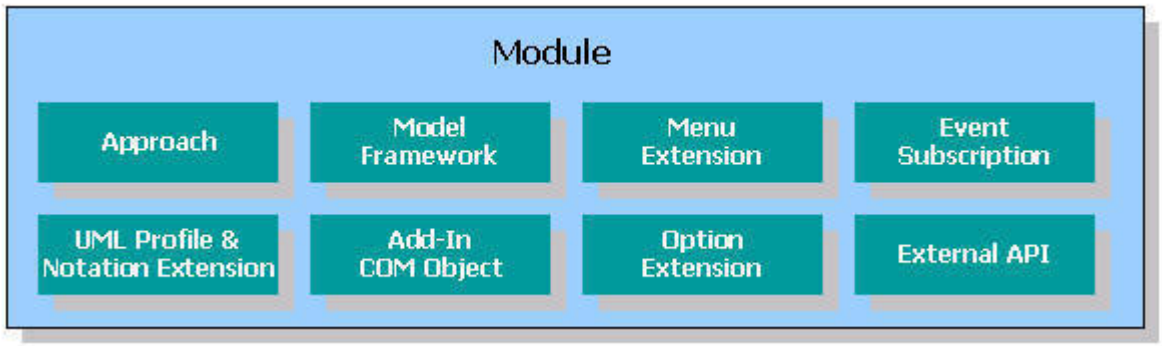
- **Подход**: Подход определяет структуру проекта и основные параметры организации диаграмм. Для детального ознакомления с концепцией подходов, см. "Глава 5. Написание подходов".
- **Профиль UML** и **Расширение нотации**: Профиль UML обеспечивает расширение набора спецификаций модели программного обеспечения через механизм расширения UML. Для детального ознакомления с профилями UML, см. "Глава7. Написание профилей UML" и "Глава 10. Расширение нотации"
- **Модельный фреймворк**: Фреймворк делает часть модели программного обеспечения многократно используемой и позволяет применять её при разработке других моделей программного обеспечения. Для детального ознакомления с концепцией фреймворков, см. "Глава 6. Написание фреймворков".
- **COM-объект расширения**: COM-аддин позволяет добавлять новые функциональные возможности к StarUML™. Для детального ознакомления с концепцией дополнительных COM-объектов см. "Глава 9. Написание дополнительных COM-объектов".
- **Расширение меню**: Меню приложения StarUML™ (главное меню и всплывающие меню) можно расширять. Для детального ознакомления с концепцией расширения меню, см. "Глава 8. Расширен меню".
- **Расширение опций**: опции настройки StarUML™ могут добавляться пользователем. Для детального ознакомления с концепцией расширения опций настройки, см. "Главу 9.

Написание дополнительных COM-объектов".

- **Обработка событий**: Различные события, возникающие в StarUML™, могут быть перехвачены и обработаны. Для детального ознакомления с концепцией обработки событий, см. "Главу 9. Написание дополнительных COM-объектов".
- **Внешнее API**: внешнее API StarUML™ позволяет получать извне доступ к различным функциональным возможностям и информации программы. Детали, касающиеся API, обсуждаются всюду в данном руководстве, и, например, включены в документационный проект, поставляющийся со StarUML™ «StarUML Application Model.uml».[4]

Организация модуля

Модуль - пакет программ, который осуществляет добавление новых функциональных возможностей к StarUML™. Модуль включает различные механизмы расширения StarUML™. Как показано на диаграмме ниже, дополнительный пакет может использовать подходы, фреймворки модели, профили UML, скрипты, расширения меню, расширения опций настройки, систему помощи и COM-объекты.



Применение модулей

Модули могут содержать различные элементы, поскольку они разрабатываются для различных целей. Модули могут использоваться для того, чтобы поддерживать специфические процессы, языки или платформы, объединяя их с другими инструментами, или расширяя существующие функции.

- Поддержка специфических процессов: компонентов UML, RUP, Catalysis, XP...
- Поддержка специфических языков программирования: C/C ++, Python, C#, Visual Basic, Java, Perl, Object Pascal...
- Интеграция со специфическими инструментами: Visual SourceSafe, CVS, MS Word, Eclipse, Visual Studio .NET...
- Расширение других функциональных возможностей: менеджер трассировки, поддержка шаблонов проектирования, проверка правил...
- Построение специфической рабочей среды (личной или стандартной для предприятия).

Элементы модуля

- **Подход**: подход применяется, чтобы определить в начале проекта его исходную структуру. Например, при создании аддина для специфического процесса, подход может использоваться, чтобы предопределить структуру моделей, создаваемых на каждой стадии

4 Данный файл находится в подкаталоге «Samples» каталога StarUML. (прим. пер.)

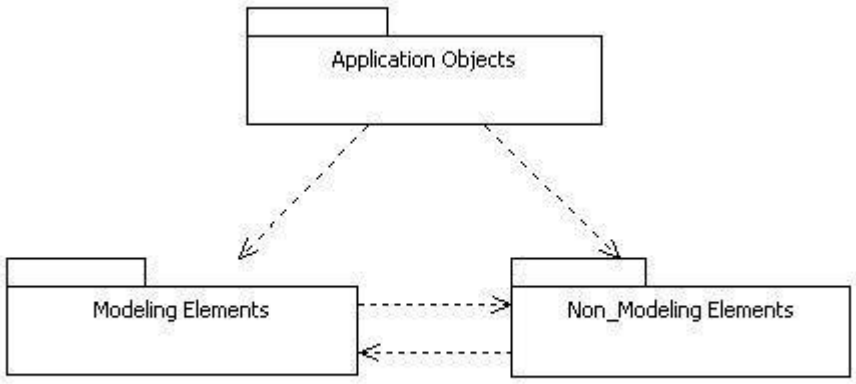
разработки проекта.

- **Фреймворк модели**: При разработке модуля, связанного со специфическими языками или платформами, фреймворк модели может предоставить готовую библиотеку классов или иной прикладной инструментарий. Любые типовые элементы проекта (например, «Событиея», «Трансакции», «Безопасность», «Каталоги»...), уже разработанные ранее, также могут быть добавлены в качестве фрагментов модели.
- **Профиль UML**: Профиль UML определяется для расширения нотации UML под специфические процессы, языки или фреймворки, включая введение дополнительных свойств элементов. Он имеет глобальный эффект в модуле.
- **Расширение меню**: Расширение меню используется, чтобы добавлять новые функциональные возможности и расширять главное меню или всплывающие меню, и тем самым позволить пользователю выбирать и выполнять новые функции. Это - критический элемент в разработке аддинов.
- **Расширение опций настройки**: Аддин может иметь собственные параметры настройки. Для их редактирования StarUML™ допускает использовать свой стандартный диалог управления опциями.
- **Дополнительный COM-объект**: Дополнительные функции могут быть реализованы с помощью языковых инструментов, подобных Visual Basic, Delphi, Visual C++, и C#. Вообще, объекты COM используются для разработки дополнительного GUI или реализации сложных функциональных возможностей. Скрипты используются для воплощения более простых функциональных возможностей. При программировании COM-объектов, обычно, используется API StarUML™.
- **Скрипт**: Простое расширение функциональных возможностей может быть выполнено, с помощью скриптовых средств (JScript, VBScript, Python...). Они обычно взаимодействуют с приложением StarUML™ через его API.
- **Помощь**: Помощь для аддина может быть создана на HTML и зарегистрирована локально или с указанием удалённого пути.

Краткий обзор API

StarUML™ предоставляет широкий набор средств API (Application Programming Interface). Внешний API StarUML™ - стандартизированный интерфейс программирования, который позволяет использовать внутренние функциональные возможности программы извне.

Как показано на диаграмме ниже, внешний API StarUML™ может быть разделен на три главных части: «элементы моделирования», «немодельные элементы» и «объекты приложения». Группа элементов моделирования предоставляет интерфейс для доступа к элементам модели, а немодельная часть предоставляет интерфейс к MOF (Meta-Object Facility) и другим объектам. Объекты приложения - это та часть интерфейса, которая позволяет управлять приложением непосредственно.

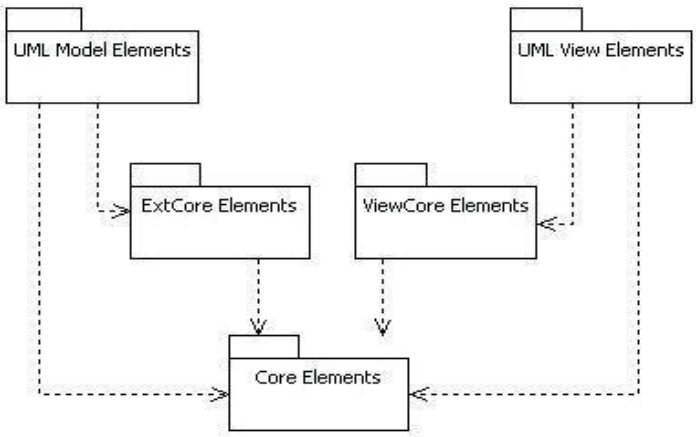


Группа объектов приложения (Application Objects)

Группа объектов приложения включает интерфейсы, которые непосредственно управляют приложением. Интерфейсы, включенные в эту часть, - это **`IStarUMLApplication`**, как основной интерфейс, **`ISelectionManager`** для управления выбором текущего элемента, **`IUMLFactory`** для того, чтобы создавать элементы, **`IProjectManager`** для того, чтобы управлять проектами, и интерфейсами, связанными с событиями и GUI.

Группа элементов моделирования (Modeling Elements)

Группа элементов моделирования включает интерфейсы для управления модельными элементами. Эта группа далее разделена на несколько подгрупп. Подгруппа «Core Elements» определяет общие корневые интерфейсы модельных элементов[5], визуальных представлений[6] и диаграмм. Подгруппа ExtCore Elements включает интерфейсы для расширяемых модельных элементов, подгруппа UML Model Elements определяет элементы моделирования UML, основанные на элементах ExtCore. Подгруппа ViewCore Elements включает интерфейсы для основных компонентов визуальных представлений, а подгруппа UML View Elements определяет визуальные представления элементов UML, также основанные на элементах ViewCore.

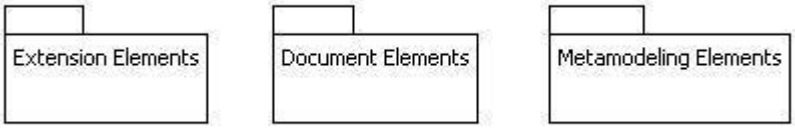


5 Термин «модельный элемент» (model element) далее означает спецификацию элемента UML.(прим. пер.)

6 Термин «визуальное представление» (view) далее, обычно, обозначает визуальный образ модельного элемента на диаграмме. Но в общем случае, диаграммы могут содержать утилитарные визуальные представления (комментарии, обводные фигуры и т.п.), которым не соответствуют никакие модельные элементы.(прим. пер.)

Подгруппа немодельных элементов (Non Modeling Elements)

Подгруппа немодельных элементов включает интерфейсы других элементов, не являющихся элементами моделирования. Она может быть далее разделена на несколько суб-частей: часть «Extension Elements», которая включает интерфейсы для элементов, связанных с механизмом расширения UML, часть «Document Elements», которая управляет сохранением файлов StarUML™, и часть «Metamodeling Elements» которая, управляет элементами мета-уровня.



Глава 3. Пример Hello World

Эта глава кратко описывает методы и процедуры разработки аддина, используя стандартный пример "Здравствуй, мир".

Пример "Hello World"

"Hello World" - первый и самый простой пример разработанной программы. В этой главе мы будем использовать этот пример, чтобы побольше узнать об аддинах. Данный пример использует не все элементы расширения, а только основные. Он включает:

- одно расширение меню,
- один скрипт .

Этот пример добавляет пункт [Hello world!] в меню, и добавляет функцию, изменяющую название проекта на "Helloworld", когда пользователь выбирает этот элемент меню.

Создание скрипта

Сначала, используйте Jscript, чтобы создать скрипт, который изменяет заголовок проекта на "Helloworld". Используйте редактор текста, чтобы ввести исходный текст скрипта как показано ниже и сохранить его как «helloworld.js».

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var prj = app.GetProject();
prj.Title = "Helloworld";
```

Первая строка скрипта создает объект с именем **StarUMLApplication**. Этот объект обязательно должен быть создан, поскольку он предоставляет исходную точку для связи со StarUML™.

Вторая строка получает доступ к объекту проекта, а третья строка присваивает названию проекта строку "Helloworld".

Создание файла расширения меню

Файл расширения меню (.mnu) должен быть создан, чтобы расширить меню StarUML™. В этом примере, мы добавим пункт [Hello, world!] в меню [Tools].

```
<?xml version="1.0"?>
<ADDINMENU addInID="StarUML.HelloworldAddIn">
  <BODY>
    <MAINMENU>
      <MAINITEM base="TOOLS" caption="Hello, world!" availableWhen="PROJECT_OPENED"
script="helloworld.js"/>
    </MAINMENU>
  </BODY>
</ADDINMENU>
```

Файл расширения меню начинается с тега **<ADDINMENU>**, включающего теги **<HEADER>** и <**BODY**>. Секция <**HEADER**> может быть опущена, а секция <**BODY**> содержит информацию для расширения

меню. В этом примере элемент <**MAINITEM**> добавляется под элементом <**MAINMENU**>, чтобы расширить главное меню. Для элемента <**MAINITEM**> (это "основной" атрибут объявления элемента меню) параметр "**caption**" - означает название пункта меню, "**availableWhen**" - условие активизации пункта меню, а "**script**" - скрипт, который выполнится, когда элемент меню будет выбран.

Обратите внимание: для детального изучения технологии расширения меню, см. "Главу 8. Расширение меню".

Разработка аддина

Файл скрипта (helloworld.js) и файл расширения меню (helloworld.mnu) должны быть размещены в одном каталоге. В корневом каталоге StarUML™ есть каталог "modules". Создайте в нём подкаталог с имением "HelloworldAddIn" и поместите в него эти два файла.

Регистрация аддина

Если Вы создали собственный аддин, Вы должны создать для него файл описания, чтобы зарегистрировать аддин в StarUML. Файл описания аддина - XML-документ, имеющий то же имя, что и аддин, и расширение «.aid». Он содержит полную информацию об аддине, включая имя аддина, имя COM объекта, имя файла выполнимого модуля, имя файла расширения меню, URL системы помощи, и так далее. Для детального ознакомления со структурой файла описания аддина, см. "Главу 9. Написание дополнительного COM-объекта".

Ниже показан файл описания аддина «HelloWord».

```
<?xml version="1.0" encoding="UTF-8"?>
<ADDIN>
    <NAME>Helloworld AddIn</NAME>
    <DISPLAYNAME>Helloworld Sample</DISPLAYNAME>
    <COMPANY>Plastic Software, Inc.</COMPANY>
    <COPYRIGHT>Copyright 2005 Plastic Software, Inc. All rights reserved.</COPYRIGHT>
    <HELPFILE>http://www.staruml.com</HELPFILE>
    <ICONFILE>Helloworld.ico</ICONFILE>
    <ISACTIVE>True</ISACTIVE>
    <MENUFILE>helloworld.mnu</MENUFILE>
    <VERSION>1.0.1.35</VERSION>
</ADDIN>
```

Сохраните файл описания в каталоге, где развернут аддин.

Проверка и выполнение аддина

Если регистрация аддина завершилась успешно, можно удостовериться, что пункт [Hello World!] добавлен в меню [Tools]. Когда этот пункт меню будет выбран, файл helloworld.js будет выполнен, и заголовок проекта изменится на "Helloworld".

Глава 4. Использование API

StarUML™ поддерживает технологию COM и предоставляет API, позволяющее обращаться к нему внешним программам.

В этой главе обсуждается использование API StarUML™.

Использование API для управления проектом

Этот раздел описывает способы управления проектами, секциями и фрагментами модели в StarUML™.

Основные концепции управления проектами

Чтобы управлять проектами, важно понять концепции, связанные с проектами (проекты, секции, и фрагменты модели).

Проект

Проект - основная структурная единица в StarUML™. Проект содержит одну или более программных моделей, он может интерпретироваться как пакет верхнего уровня, который не изменяется. Один проект обычно сохраняется как один файл. Проект содержит следующие элементы моделирования.

- **Model**
  Элемент-контейнер, содержащий одну программную модель.
- **Subsystem**
  Элемент, содержащий другие элементы, образующие подсистему.
- **Package**
  Основной элемент для объединения других элементов.

Проектные файлы сохраняются в формате XML, и имеют расширение - ".UML".

Хотя все модели, представления, и диаграммы, созданные в StarUML™, обычно сохраняются в одном проектном файле, тем не менее проект может быть разделен и сохранен в нескольких файлах, путём использования механизма секций, который описан в следующем разделе. Следующая информация сохраняется в проектных файлах.

- Профиль UML, используемый проектом
- Файлы секций, на которые ссылается проект
- Все модельные элементы, содержащиеся в проекте
- Все диаграммы и визуальные представления, содержащиеся в проекте

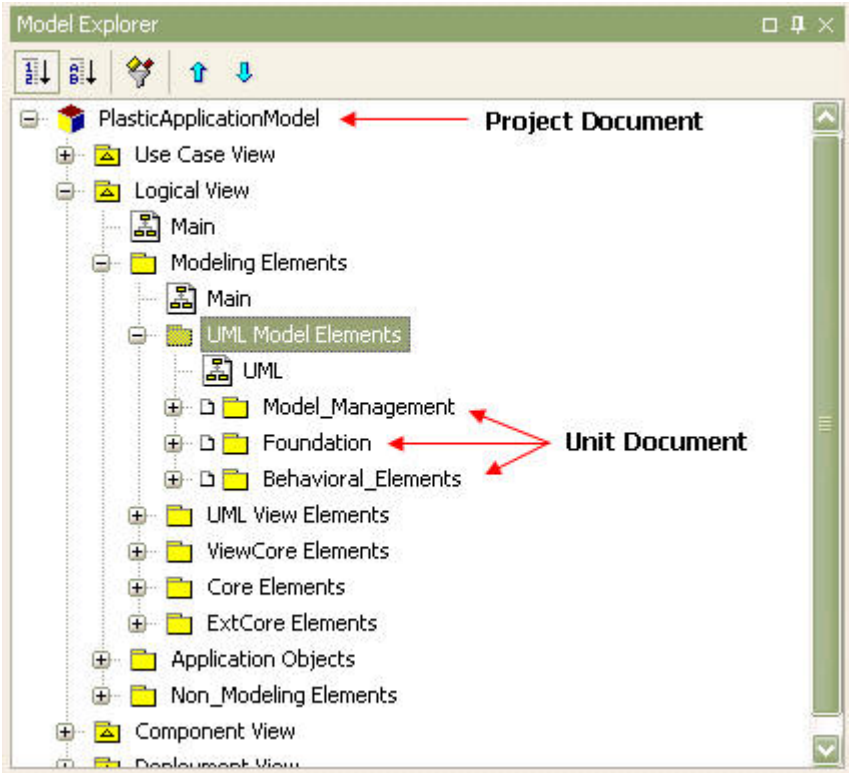
Секция

Хотя проект обычно сохраняется в одном файле, возможны случаи, когда проект целесообразно разделить и сохранять в нескольких файлах, например, потому что с ним одновременно должны

работать много людей и т.п. В подобной ситуации, проект может быть разделён на несколько секций[7]. Секции могут быть организованы иерархически, и одна секция может иметь много подсекций. Секция сохраняется в файле с расширением ".UNT", на который ссылаются проектные файлы (.UML) и другие файлы секции (.UNT).

Только пакет, подсистема, или модель могут быть оформлены в виде секции. Любой из этих элементов может быть сохранен в файле секции (.UNT).

Так же, как проект может включать множество секций, секция может включать много подсекций. Корневая секция содержит ссылки на подсекции, так что секция имеет иерархическую структуру.



Фрагмент модели

Фрагмент модели – часть данных проекта, скопированная в отдельный файл. Только модель, подсистема, или пакет могут быть выведены в качестве фрагмента в файл с расширением ".MFG". Фрагмент модели можно только добавить к любому другому проекту. Фрагменты модели существенно отличаются от секций, поскольку могут быть только подгружены в проект[8].

Работа с объектом документа

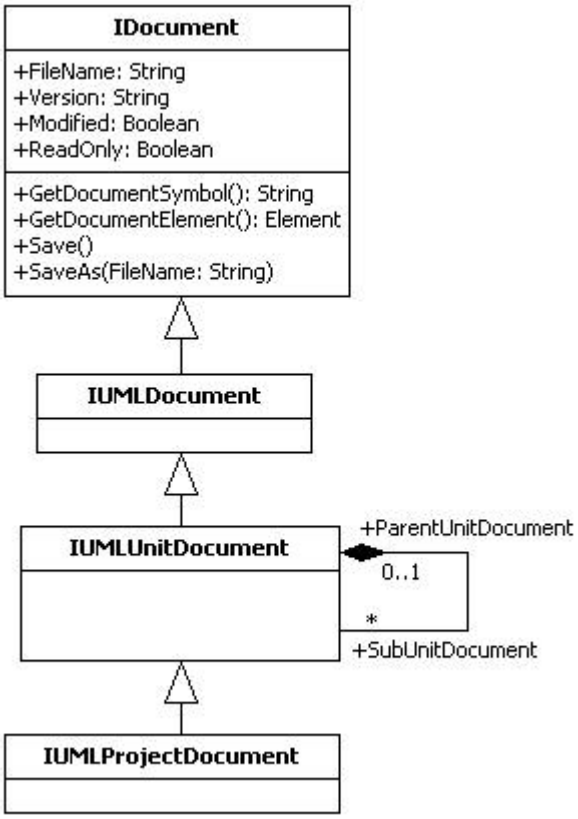
Концепция документа

Документ - абстрактный объект, соответствующий части проекта, сохраняемой как файл в StarUML™. Другими словами, документ предоставляет различные свойства и методы для работы с

7 В оригинале используется термин «unit», который традиционно переводится как «модуль». Однако термин «модуль» уже используется в данном тексте, как перевод термина «module», являющегося синонимом термина «аддин».

8 Здесь имеется в виду, что фрагмент в отличие от секции не является частью проекта. После загрузки в проект данные фрагмента интегрируются с остальным содержимым проекта и при сохранении проекта сохраняются в файле «UML», так что файл «MFG» становится ненужным и может быть удалён.

частями проекта типа «.UML» или «.UNT», как с единым объектом. Хотя фрагмент модели (.MFG) также представляет собой отдельный файл, он не может интерпретироваться как документ, поскольку используется исключительно для импорта/экспорта и отдельно не обрабатывается приложением StarUML™. Следующая диаграмма иллюстрирует иерархическую структуру интерфейсов документа.



- **`IDocument`**: Общий родительский интерфейс для документов.
- **`IUMLDocument`**: интерфейс верхнего уровня для документов, связанных с UML-моделями.
- **`IUMLUnitDocument`**: интерфейс для документов, обрабатываемых как секции (.UNT) в StarUML™.
- **`IUMLProjectDocument`**: интерфейс для документов, обрабатываемых как проекты (.UML) в StarUML™. Так как проектный документ рассматривается как частный случай документа секции, он наследует свойства интерфейса документа секции.

Доступ к документарным объектам

Чтобы обратиться к проекту или секции, необходимо иметь ссылку на объект **`IProjectManager`**. Он позволяет осуществлять прямой доступ к документам проекта и секций.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var prjmgr = app.ProjectManager;
// Get project document object.
var prj_doc = prjmgr.ProjectDocument;
// Get unit document objects.
for (var i = 0; i < prjmgr.GetUnitDocumentCount(); i++)
{ var unit_doc = prjmgr.GetUnitDocumentAt(i);}
```

Ссылка на документ может быть получена не только через **`IProjectManager`**, но и через модельные элементы, которые в нём содержатся. Следующий пример иллюстрирует получение ссылки на документ из модельного элемента и последующее сохранение документа.

```
var elem = ... // Присваивается конкретный элемент (класс, пакет, и т.п.)
var elem_doc = elem.GetContainingDocument();
elem_doc.Save();
```

Свойства и методы документа

Интерфейс **`IDocument`** предоставляет следующие свойства и методы.

Свойство	Описание
FileName: String	Хранит имя файла документа. Имя файла включает полный путь и расширение.
Version: String	Хранит версию документа.
Modified: Boolean	Определяет, изменялся ли документ пользователем.
ReadOnly: Boolean	Определяет, может ли изменяться файл документа.

Метод	Описание
GetDocumentSymbol(): String	Возвращает тип документа. Строка 'PROJECT' возвращается для проектных документов, а строка 'UNIT' для документов секций.
GetDocumentElement(): IElement	Возвращает главный элемент документа.
Save()	Сохраняет документ в файле, из которого он был прочитан.
SaveAs(FileName: String)	Сохраняет документ в другом файле и изменяет свойство FileName.

Управление объектом проекта

Доступ к объекту проекта

Чтобы непосредственно работать с проектом, необходимо наличие ссылки на проектный объект. Ниже приведён код на Jscript, иллюстрирующий получение такой ссылки.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var prj = app.GetProject();
...
```

Хотя ссылка на проект может быть получена непосредственно от объекта приложения, к проекту можно также обратиться, используя следующий способ

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var prjmgr = app.ProjectManager;
var prj = prjmgr.Project;
...
```

Изменение заголовка и свойств проекта

Как только ссылка на проект будет получена, становятся доступными его заголовок, свойства и различные методы. Чтобы изменить заголовок проекта, нужно изменить его свойство "Title". Другие свойства типа "Copyright", "Author" и "Company" могут изменяться тем же самым способом.

```
...
prj.Title = "MyProject";
...
```

1. **Предупреждение**: Хотя формально проект, как и все модельные элементы, имеет свойство "Name", объект проекта, не должен его использовать. Проект - корневой пакет, он не может иметь имя. Это ограничение наложено из-за того, что для идентификации модельных элементов обычно используются их полные имена (пути), которые станут некорректными, если корневому пакету будет присвоено имя.[9]

Добавление пакетов в проект

Только модель, подсистему или пакет можно добавить непосредственно в проект (как в пакет верхнего уровня).

Чтобы создавать и добавлять новые элементы, должен использоваться объект IUMLFactory. См. следующий пример.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var factory = app.UMLFactory;
var prj = app.GetProject();
var newPackage = factory.CreatePackage(prj);
newPackage.Name = "NewPackage";
```

Создание нового проекта

Чтобы создать новый проект, получите ссылку на объект IProjectManager, и вызовите метод NewProject.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var prjmgr = app.ProjectManager;
prjmgr.NewProject();
```

Чтобы создать новый проект с определенным подходом (вместо пустого проекта), используйте метод NewProjectByApproach. Следующий пример иллюстрирует создание нового проекта, используя подход "UMLComponents".

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var prjmgr = app.ProjectManager;
prjmgr.NewProjectByApproach("UMLComponents");
```

Открытие проекта

Чтобы открыть проектный файл (.UML), получите ссылку на объект IProjectManager, и затем используйте метод OpenProject.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var prjmgr = app.ProjectManager;
prjmgr.OpenProject("C:\\MyProject.uml");
```

9 Полное имя модельного элемента (path) представляет собой последовательность имён всех его контейнеров, в порядке вложенности, разделённых символами «::», за которыми следует собственно имя элемента. Здесь имеет место абсолютная аналогия с именами файлов в операционной системе: пакет проекта подобен корневому каталогу диска, также не имеющему собственного имени.

Сохранение проекта

Чтобы сохранить текущий проект, открытый в StarUML™, нужно получить ссылку на объект **`IProjectManager`** и затем использовать метод **`SaveProject`**. Используйте метод **`SaveProjectAs`**, чтобы сохранить проект под другим именем. Используйте метод **`SaveAllUnits`**, чтобы сохранить все секции проекта.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var prjmgr = app.ProjectManager;
prjmgr.SaveProject();
prjmgr.SaveProjectAs("MyProject2.uml");
prjmgr.SaveAllUnits();
```

Закрытие проекта

Чтобы закрыть проект, получите ссылку на объект IProjectManager, и затем используйте его метод CloseProject

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var prjmgr = app.ProjectManager;
prjmgr.CloseProject();
```

Работа с секциями

Выделение новой секции

Чтобы выделить новую секцию и затем работать с пакетом, моделью или подсистемой как отдельным файлом, нужно получить ссылку на объект IProjectManager и затем использовать метод SeparateUnit.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var prjmgr = app.ProjectManager;
var pkg = ... // Присвоить ссылку на пакет, превращаемый в секцию
var new_unit = prjmgr.SeparateUnit(pkg, "NewUnit.unt");
```

Присоединение секции

Если отделенный пакет, модель или секция подсистемы больше не должны обрабатываться как отдельный файл и должны быть объединены с проектом, получите ссылку на объект IProjectManager и затем используйте метод MergeUnit.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var prjmgr = app.ProjectManager;
var pkg = ... // Assigns reference for the package that will no longer be managed as a unit.
prjmgr.MergeUnit(pkg);
```

Доступ к подсекциям

Секция может быть организована иерархически. Проект может иметь много секций, а каждая секция может иметь много подсекции. Следующий пример иллюстрирует доступ к подсекции в пределах секции.

```
var unit = ... // Присваиваем ссылку на секцию, которая имеет подсекции
for (var i = 0; i < unit.GetSubUnitDocumentCount(); i++) {
```

```
var sub_unit = unit.GetSubUnitDocumentAt(i);
...
}
```

Работа с фрагментами модели

Создание фрагмента модели на основе пакета

Пакет, модель или подсистема могут быть сохранены как отдельный файл фрагмента модели. Получите ссылку на объект IProjectManager, и затем используйте метод ExportModelFragment.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var prjmgr = app.ProjectManager;
var pkg = ... // Получаем пакет, который экспортируем как фрагмент.
prjmgr.ExportModelFragment(pkg, "MyFragment.mfg");
```

Импорт фрагмента модели

Файл фрагмента модели можно добавить к пакету, модели, или подсистеме. Получите ссылку на объект IProjectManager, и затем используйте метод ImportModelFragment.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var prjmgr = app.ProjectManager;
var pkg = ... // Ссылка на элемент , в который будет добавлен фрагмент
prjmgr.ImportModelFragment(pkg, "MyFragment.mfg");
```

Использование API для модельных элементов

Этот раздел вводит типы интерфейсов, которые соответствуют модельным элементам StarUML™, и описывает их использование. Элементы моделирования ссылаются на UML-модель, представления и диаграммы, которые используются при моделировании программного обеспечения. Элементы моделирования типа пакета, класса, и актора, визуальные представления, которые соответствуют каждому модельному элементу, а также диаграммы типа диаграммы классов и прецедентов - примеры элементов моделирования. Модельный элемент, представление и диаграмма могут быть созданы, удалены или изменены посредством использования API.

Обратите внимание: Пожалуйста, обратитесь к "Приложению B. Список модельных элементов UML" для получения полного перечня модельных элементов.

Структура элемента моделирования

Элементы моделирования организованы в следующие логические группы.

- **Core Elements**: Группа Core Elements определяет интерфейс верхнего уровня для модельных элементов, представлений и диаграмм.
- **ExtCore Elements**: Группа ExtCore Elements определяет общий интерфейс для расширяемых модельных элементов.
- **ViewCore Elements**: Группа ViewCore Elements определяет основные типы для представлений.
- **UML Model Elements**: Определяет элементы UML-модели. Стандартные модельные элементы UML относятся к этой категории.
- **UML View Elements**: Группа UML View Elements определяет визуальные представления

элементов UML.

Типы элементов моделирования разделены на модельные элементы, представления и диаграммы. Однако, диаграммы фактически являются подмножеством модельных элементов и представлений, и, таким образом, более точным будет разделение на модельные элементы и представления. Модельный элемент - элемент, который содержит актуальную информацию о модели программной системы, а представление - визуальное отображение информации, содержавшейся в определенном модельном элементе. Один модельный элемент может иметь множество визуальных представлений, в то время как представление, в общем случае, имеет ссылку только на один модельный элемент.

Простой пример использования элементов моделирования

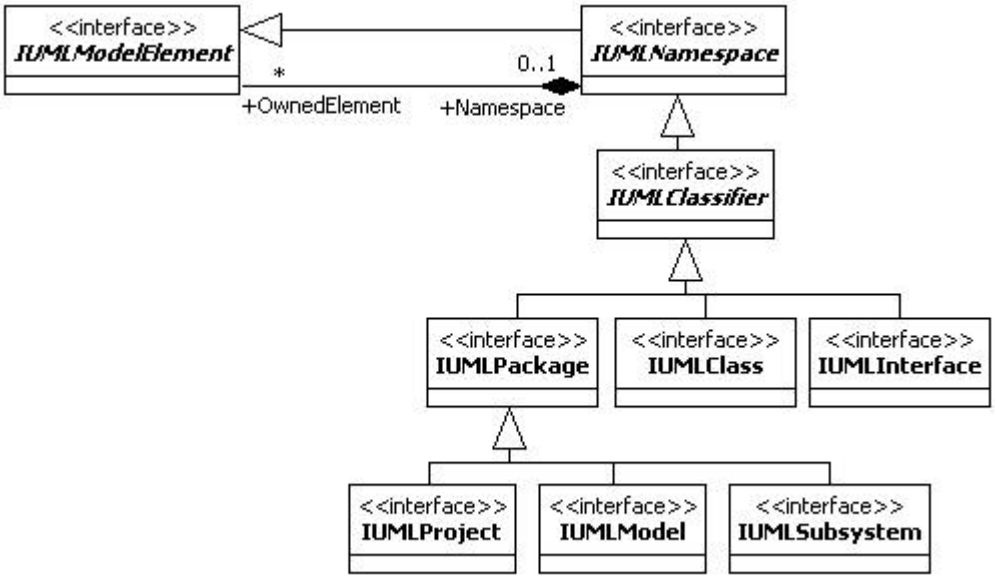
Перед тем как представить внешние интерфейсы API для элементов моделирования, позвольте рассмотреть простой пример их использования. Предположим, что мы хотим перебрать все элементы приложения StarUML™ сверху до низу через пространства имён пакетов, классов, интерфейсов и т.д. В этом случае должна анализироваться структура элементов моделирования. Следующий код на Jscript иллюстрирует эту структуру.

```
var app, prj;
app = new ActiveXObject("StarUML.StarUMLApplication");
prj = app.GetProject();
VisitOwnedElement(prj);

function VisitOwnedElement(owner){
var elem;
for (var i = 0; i < owner.GetOwnedElementCount(); i++){
elem = owner.GetOwnedElementAt(i);
...
if (elem.IsKindOf("UMLNamespace")) VisitOwnedElement(elem);
}
}
```

В этом примере, все суб-элементы, которые находятся в отношении "OwnedElement" с некоторым главным элементом, извлекаются рекурсивно. Самая важная часть этого кода - определяемая пользователем функция с именем VisitOwnedElement. Эта функция берет некоторый элемент абстрактного общего типа IUMLNamespace (который является модельным элементом) как входной параметр и использует функции GetOwnedElementCount и GetOwnedElementAt, которые являются методами интерфейса IUMLNamespace.

Информация, требуемая чтобы структурировать функцию VisitOwnedElement, может быть получена из отношений элементов моделирования. Следующая диаграмма иллюстрирует отношения между типами интерфейсов API StarUML™, которые связаны с примером использования интерфейса IMULNamespace (см. выше).



Интерфейс IUMLNamespace унаследован от IUMLModelElement, который является родительским типом для типов IUMLPackage, IUMLClass и IUMLInterface. IUMLNamespce также имеет ассоциацию Namespace-OwnedElement. Диаграмма показывает, что элементы типа IUMLNamespace, подобные IUMLPackage, IUMLClass, и т.д. обладают (как агрегаты) элементами типа IUMLModelElement в качестве OwnedElements. Таким образом, интерфейсы API определены в соответствии с реальными отношениями между модельными элементами.

Обратите внимание: имена модельных элементов, которые относятся к категории стандартных элементов UML, начинаются с префикса "UML" перед стандартным названием элемента UML. Например, имя элемента Актор - UMLActor. Кроме того используется префикс "I", в соответствии с соглашением о кодировании имён интерфейсов, в итоге - IUMLActor. Пожалуйста обратитесь к "Приложению B. Список модельных элементов UML" для получения полного перечня элементов UML и их названий.

Соглашение для выражения ассоциации в API

Диаграмма, представленная выше показывает что интерфейсы IUMLModelElement и IUMLNamespace имеют ассоциацию OwnedElement-Namespace. Такие ассоциации выражают ссылки между интерфейсами API StarUML™. Например, ассоциация с Namespace в интерфейсе IUMLModelElement выражается в виде свойства, как показано ниже.

IUMLModelElement
Namespace: IUMLNamespace

Эта же ассоциация OwnedElement, но в интерфейсе IUMLNamespace выражена по другому (см. ниже). Это происходит, потому что атрибут Multiplicity метамодели имеет значение *, и для его реализации во внутреннем представлении программы будет использована структура группы или списка. Поскольку все ассоциации в интерфейсах API выражены, используя то же самое соглашение, то данный пример применим и ко всем другие интерфейсам, так же как к IUMLModelElement-IUMLNamespace.

IUMLNamespace
function GetOwnedElementCount(): Integer;
function GetOwnedElementAt(Index: Integer): IUMLModelElement;

Элементы Core

Core Elements - главные родительские интерфейсы для модельных элементов. К этой категории относятся IElement, IModel, IView, IDiagram и интерфейс IDiagramView, которые организованы как показано на диаграмме ниже. Этой организации необходимо уделить специальное внимание, поскольку основные типы интерфейсов этой группы весьма часто используются и играют критические роли. На ассоциациях между интерфейсами нужно сделать особый акцент.

Имя интерфейса	Описание
IElement	Тип интерфейса, который определяет верхний уровень, общий родительский элемент для всех элементов моделирования.
IModel	Тип интерфейса, который определяет родительский элемент для модельных элементов.
IView	Тип интерфейса, который определяет родительский элемент для представлений элементов.
IDiagram	Тип интерфейса, который определяет родительский элемент для диаграмм.
IDiagramView	Тип интерфейса, который определяет родительский элемент для представлений диаграмм.

IElement

Интерфейс IElement определяет тип общего предка всех элементов моделирования, и предоставляет следующие главные методы.

Метод	Описание
GetGUID(): String	Функция, которая возвращает GUID (Глобальный Уникальный Идентификатор) элементов моделирования. GUID закодирован как Base64.
GetClassName(): String	Функция, которая возвращает имя класса модельного элемента. Пример возвращаемого значения: "UMLClass"
IsKindOf(ClassName: String): Boolean	Функция, которая проверяет, принадлежит ли элемент моделирования к типу, указанному в качестве параметра. Пример значения параметра: "UMLClass"
IsReadOnly(): Boolean	Функция, которая проверяет, можно ли изменять элемент моделирования. Свойства элементов моделирования, имеющие признак "только для чтения" не могут изменяться.
MOF_GetAttribute(Name: String): String	Возвращает в виде строки значение заданного свойства модельного элемента.
MOF_GetReference(Name: String): IElement	Возвращает значение заданного свойства модельного элемента в виде ссылки (объектной ссылки) .
MOF_GetCollectionCount(Name: String): Integer	Возвращает количество элементов в коллекции ссылок, заданной аргументом.
MOF_GetCollectionItem(Name: String; Index: Integer): IElement	Возвращает в виде ссылки (объектная ссылка) заданный по индексу элемент указанной коллекции модельного элемента.

Среди методов интерфейса IElement, методы с именами MOF_XXX предоставляют стандартный способ обращаться к свойствам каждого элемента моделирования по их именам. Например, тип IUMLModelElement, подтип типа IElement, имеет свойство с именем "Visibility". В общем случае,

чтобы получить значение этого свойства используется выражение типа IUMLModelElement.Visibilty. Но метод IElement.MOF_GetAttribute может использоваться как показано ниже, чтобы получить значение свойства в виде строки по его имени "Visibilty". Таким же образом, другие методы MOF_XXX предоставляют доступ к свойствам простого типа/типа ссылки/ типа коллекции ссылок каждого элемента моделирования по их именам, и это очень полезно во многих случаях.

Обратите внимание: Строковые названия свойств, которые используются как параметры в методах MOF_XXX, должны совпадать с именами соответствующих свойств.

Следующий пример читает значение свойства "Visibilty" типа IUMLModelElement, используя метод IElement.MOF_GetAttribute. Должно быть отмечено, что метод MOF_GetAttribute использует строки в качестве возвращаемого значения. В этом примере, возвращаемые значения могут быть "vkPrivate", "vkPublic", и т.д.

```
...
var elem = ... // присвойте ссылку на объект типа IUMLModelElement.
var val = elem.MOF_GetAttribute("Visibility");
...
```

Метод IElement.MOF_GetReference используется при чтении значений свойств типа ссылки на элемент моделирования. Метод MOF_GetReference возвращает ссылку на объект типа IElement. Следующий пример читает свойство типа ссылки на IUMLModelElement.

```
...
var elem = ... // получает ссылку на объект типа IUMLModelElement.
var refElem = elem.MOF_GetReference("Namespace");
...
```

Метод IElement.MOF_GetCollectionItem используется при чтении свойств со значением типа коллекции ссылок на элементы моделирования. Метод MOF_GetCollectionItem получает имя коллекции и индекса элемента в качестве параметров. Количество элементов коллекции может быть получено, используя метод MOF_GetCollectionCount. Аналогично, метод MOF_GetCollectionItem, подобный методу MOF_GetReference возвращает ссылку на объект типа IElement. Следующий пример читает коллекцию "Attributes" элемента типа IUMLClassifier.

```
...
var elem = ... // Get reference to IUMLClassifier type element object.
var colCount = elem.MOF_GetCollectionCount("Attributes");
for (var i = 0; i < colCount; i++){
var colItem = elem.MOF_GetCollectionItem("Attributes", i);
...
}
```

Обратите внимание: Произойдёт ошибка, если значение параметра для MOF_XXX метода не соответствует имени существующего свойства.

IModel

Интерфейс IModel определяет родительский тип интерфейса для модельных элементов, и предоставляет следующие главные свойства и методы.

Свойство	Описание
Name: String	Имя
Documentation: String	Документация

Свойство	Описание
Pathname: String	Полное имя модельного элемента (путь). Формат пути включает разделитель "::" для всех элементов верхнего уровня кроме высшего (проектного элемента). Пример имени пути: "::Application Model::Modeling Elements::UML Model Elements". * Только для чтения.

Метод	Описание
AddAttachment(Attach: String);	Добавляет приложение к элементу (файл, URL).
FindByName(AName: String): IModel	Возвращает модельный элемент более низкого уровня с указанным именем.
FindByRelativePathname(RelPath: String): IModel	Возвращает модельный элемент более низкого уровня по указанному относительному пути. Имя самого текущего элемента не должно присутствовать в параметре. Пример значения параметра: "Model_Management:: UMLPackage"
ContainsName(AName: String): Boolean	Проверяет, существует ли модельный элемент более низкого уровня с указанным именем.
CanDelete(): Boolean	Проверяет, является ли текущий элемент "только для чтения ".
GetViewCount: Integer	Возвращает количество представлений (представлений) элемента.
GetViewAt(Index: Integer): IView	Возвращает визуальный образ по индексу.
GetOwnedDiagramCount: Integer	Возвращает количество диаграмм, которыми владеет элемент.
GetOwnedDiagramAt(Index: Integer): IDiagram	Возвращает диаграмму элемента по индексу.

Следующий пример вызывает основные свойства модельного элемента и устанавливает их значения.

```
function DoingSomething(elem){
if (elem.GetClassName() == "UMLClass"){
if (elem.IsReadOnly() != true){
elem.Name = "class_" + elem.Name;
elem.Documentation = "I am a class";
elem.AddAttachment("http://www.staruml.com");
}}}
```

Методы FindByName и FindByRelativePathname могут использоваться, чтобы найти элементы нижнего уровня по отношению к текущему элементу. Метод FindByName возвращает, имя первого попавшегося элемента, который имеет имя идентичное полученному как параметр. Метод FindByName выполняет поиск только на более низком уровне от вызывающего модельного элемента.

Чтобы искать среди всех элементов более низкого уровня, когда элементы более низкого уровня имеют перекрывающуюся структуру, может использоваться метод FindByRelativePathname. Следующий пример показывает, как использовать методы FindByName и FindByRelativePathname.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var rootElem = app.FindByPathname("::Application Model::Modeling Elements::UML Model
Elements");
var elem = rootElem.FindByName("Model_Management");
var elem2 = rootElem.FindByRelativePathname("Model_Management::UMLPackage");
```

Как показано на диаграмме выше, интерфейс IModel и интерфейс IView имеют ассоциацию Model-View. Элементу типа IModel может соответствовать много элементов типа IView, а каждому

элементу типа IView может соответствовать только один элемент типа IModel. Следующий пример показывает, как получить ссылку на все элементы типа IView для каждого элемента типа IUMLClass.

```
var elem = ... // получает ссылку на элемент типа IModel
if (elem.GetClassName() == "UMLClass"){
for (var i = 0; i < elem.GetViewCount(); i++){
var view = elem.GetViewAt(i);
...
}}
```

Как иллюстрировано на диаграмме выше, интерфейс IModel и интерфейс IDiagram имеют ассоциацию DiagramOwner-OwnedDiagram. Так как интерфейс IDiagram - родительский тип для всех типов диаграмм, ссылка на любую диаграмму, содержащуюся в модельном элементе, может быть получена, используя способ, показанный в следующем примере.

```
var elem = ... // IModel type element
for (int i = 0; i < elem.GetOwnedDiagramCount(); i++){
var dgm = elem.GetOwnedDiagramAt(i);
...
}
```

IView

Интерфейс IView определяет общий родительский тип представлений (представлений) и предоставляет следующие главные свойства.

Свойство	Описание
LineColor: String	Определяет цвет строки. Использует формат RGB. Примеры: "0xff0000" (синий); "0x00ff00" (зеленый); "0x0000ff" (красный); "0x000000" (черный); "0xffffff" (белый)
FillColor: String	Определяет цвет заливки.
FontFace: String	Определяет шрифт. Пример: "Times New Roman"
FontColor: String	Цвет шрифта.
FontSize: String	Размер шрифта.
FontStyle: Integer	Определяет начертание шрифта. Целые числа 1 (полужирный), 2 (курсив), 3 (подчеркивание), и 4 (зачеркнутый) могут использоваться отдельно или в комбинации. Пример: 1 + 2 (полужирный и курсивный) * Не применяется к элементам, с предопределенными заданными по умолчанию стилями.
Selected: Boolean	Определяет, выбрано ли визуальное представление. * Только для чтения.
Model: IModel	Определяет ссылку на модельный элемент, соответствующий представлению * Только для чтения.
OwnerDiagramView: IDiagramView	Определяет образ диаграммы, содержащий даный образ. * Только для чтения.

Следующий пример показывает установку основных атрибутов элемента типа IVew.

```
var view = ... // IView type element
view.LineColor = "0x0000ff";
view.FillColor = "0x00ffff";
view.FontFace = "Times New Roman";
view.FontColor = "0x0000ff";
view.FontSize = "12";
```

```
view.FontStyle = 1;
```

Все визуальные элементы, кроме IUMLNoteView, IUMLNoteLinkView и IUMLTextView имеют ссылки на модельный элемент. Следующий код может использоваться, чтобы получить ссылку на владеющий элемент типа IModel.

```
var view = ... // IView type element
var mdl = view.Model;
...
```

Следующий код может использоваться, чтобы получить ссылку на диаграмму, которая содержит элемент типа IView.

```
var view = ... // IView type element
var dgmView = view.OwnerDiagramView;
...
```

IDiagram

Интерфейс IDiagram унаследован от интерфейса IModel, он является общим родительским типом всех модельных элементов, соответствующих диаграммам. Интерфейс IDiagram имеет следующие основные свойства.

Свойство	Описание
DefaultDiagram: Boolean	Определяет, является ли текущая диаграмма заданной по умолчанию диаграммой. Заданная по умолчанию диаграмма - это диаграмма, которая автоматически открывается, когда открывается проект. Только диаграмма классов/прецедентов/компонентов/развертывания может быть установлена как заданная по умолчанию диаграмма.
DiagramOwner: IModel	Определяет элемент верхнего уровня, который содержит текущую диаграмму. * Только для чтения.
DiagramView: IDiagramView	Определяет представление диаграммы, который соответствует диаграмме. * Только для чтения.

IDiagramView

Интерфейс IDiagramView унаследован от интерфейса IView, и является общим родительским типом для типов визуальных представлений всех диаграмм.

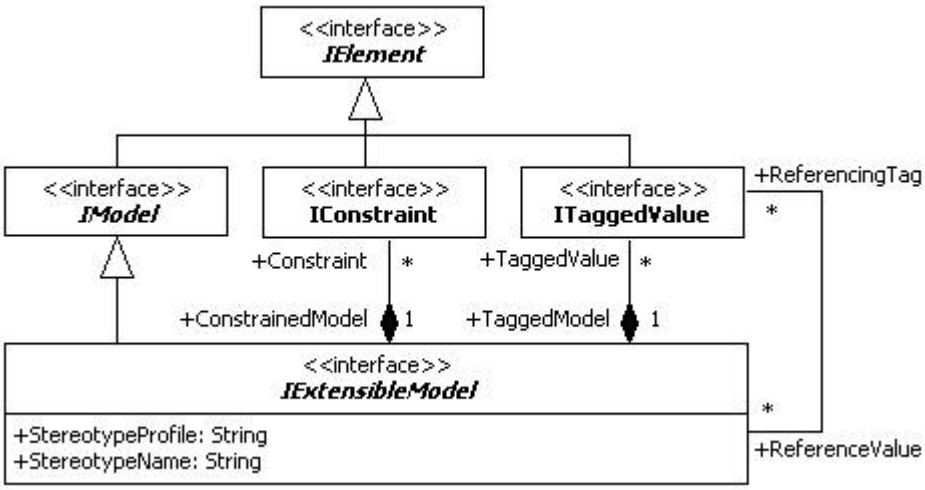
Свойство	Описание
Diagram: IDiagram	Определяет элемент диаграммы, который соответствует текущему представлению диаграммы *Только для чтения.

Метод	Описание
GetSelectedViewCount: Integer	Возвращает количество визуальных представлений, в настоящее время выбранных на диаграмме.
GetSelectedViewAt(Index: Integer): IView	Возвращает по индексу представление, которое в настоящее время выбрано на диаграмме.
GetOwnedViewCount: Integer	Возвращает количество представлений, содержащихся на диаграмме.
GetOwnedViewAt(Index: Integer): IView	Возвращает по индексу представление, содержащееся на диаграмме.

Метод	Описание
LayoutDiagram()	Автоматически реорганизует формат диаграммы.
ExportDiagramAsBitmap(FileName: String)	Конвертирует диаграмму в растровое изображение и сохраняет её как файл, используя указанные путь и имя файла.
ExportDiagramAsMetafile(FileName: String)	Конвертирует диаграмму в метафайл Windows и сохраняет её как файл, используя указанные путь и имя файла.
ExportDiagramAsJPEG(File Name: String)	Конвертирует диаграмму в JPEG-изображение и сохраняет её как файл, используя указанные путь и имя файла.

Элементы ExtCore

Элементы ExtCore предоставляют структурную основу для элементов моделирования, к которым могут быть применены функции расширения спецификаций UML. Все подобные элементы унаследованы от интерфейса IExtensibleModel. Интерфейс IExtensibleModel может иметь много ограничений и тегов, как показано на диаграмме ниже.



Interface name	Описание
IExtensibleModel	Общий тип верхнего уровня для элементов, к которым могут быть применены функции расширения спецификаций UML.
IConstraint	Элемент ограничения.
ITaggedValue	Элемент дополнительного свойства.

Интерфейс IExtensibleModel предоставляет следующие основные свойства и методы.

Свойство	Описание
StereotypeProfile: String	Определяет имя профиля UML, который определяет стереотип, присвоенный текущему модельному элементу * Только для чтения.
StereotypeName: String	Имя стереотипа модельного элемента. * Только для чтения.

Метод	Описание
GetConstraintCount: Integer	Возвращает количество элементов ограничений, содержащихся в текущем модельном элементе.

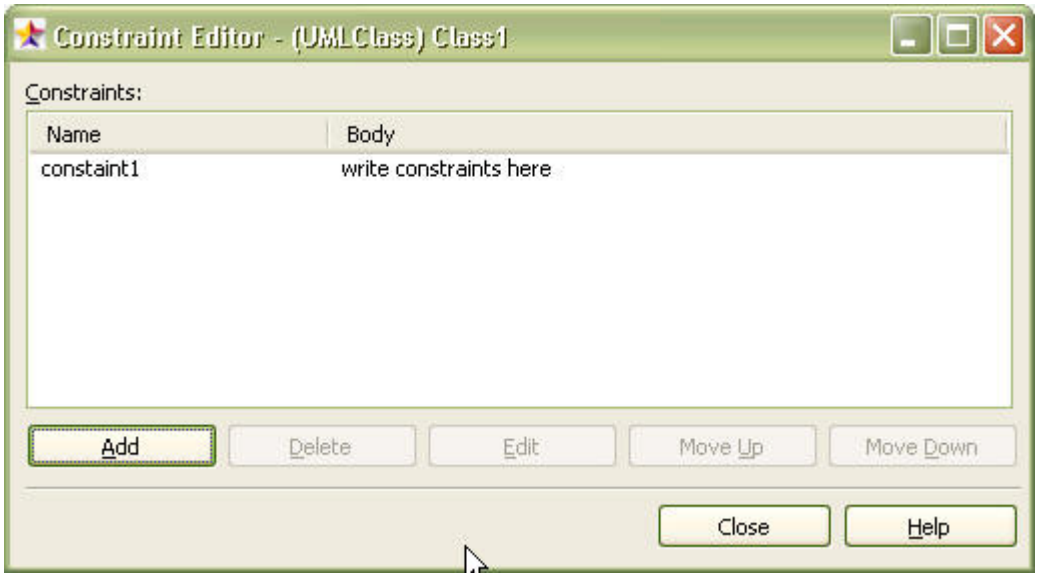
Метод	Описание
GetConstraintAt(Index: Integer): IConstraint	Возвращает ограничение, содержащееся в текущем модельном элементе, по его индексу.
AddConstraint(Name: String; Body: String): IConstraint	Создает элемент ограничения с указанным именем и значением.
IndexOfConstraint(AConstraint: IConstraint): Integer	Возвращает индекс элемента ограничения, заданного параметром.
DeleteConstraint(Index: Integer)	Удаляет элемент ограничения, содержавщийся в текущем модельном элементе, по его индексу.
GetTaggedValueCount: Integer	Возвращает количество тегов, содержащихся в текущем модельном элементе.
GetTaggedValueAt(Index: Integer): ITaggedValue	Возвращает значение тега, содержащегося в текущем модельном элементе, по его индексу.
GetStereotype: IStereotype	Возвращает элемент стереотипа текущего модельного элемента.
SetStereotype(const Name: WideString)	Определяет значение стереотипа как строку вместо того, чтобы использовать элемент IStereotype.
SetStereotype2(Profile: String; Name: String)	Определяет профиль UML в котором определяется стереотип и стереотипное значение.

В соответствии с соглашением, стереотип и теги должны быть определены в профиле UML. Однако, StarUML™ позволяет определять стереотипы строковыми значениями, отсутствующими в профилях UML. Следующий пример показывает, как прочесть значение стереотипа некоторого элемента типа IExtensibleModel и переопределить его.

```
var elem = ... // Get reference to model element.
if (elem.IsKindOf("ExtensibleModel")){
var stereotypeStr = elem.StereotypeName;
if (stereotypeStr == ""){
elem.SetStereotype("Stereotype1");
} }
```

В отличие от стереотипа, значения тегов должны быть определены только через профиль UML. Пожалуйста обращайтесь к "Главе 7. Написание профилей UML" для детального ознакомления с профилями UML, стереотипами и тегами.

IConstraint



Как добавлять и редактировать ограничения в редакторе ограничений StarUML™ было уже иллюстрировано выше. Посредством API ограничения можно добавлять или редактировать с помощью интерфейса IConstraint. Интерфейс IConstraint предоставляет следующие свойства.

Свойство	Описание
Name: String	Имя ограничения.
Body: String	Содержание ограничения.
ConstrainedModel: IExtensibleModel	Элемент типа IExtensibleModel, содержащий ограничение.

Элементы ограничений могут быть созданы с помощью метода, предоставляемого элементом типа IExtensibleModel. Следующий пример иллюстрирует добавление, редактирование, и удаление ограничения для некоторого элемента типа IExtensibleModel.

```
var elem = ... // Get reference to IExtensibleModel type element.
var AConstraint = elem.AddConstraint("Constraint1", "Constraint Value1");
var constrName = AConstraint.Name;
var constrValue = AConstraint.Body;
var idx = elem.IndexOfConstraint(AConstraint);
elem.DeleteConstraint(idx);
```

ITaggedValue

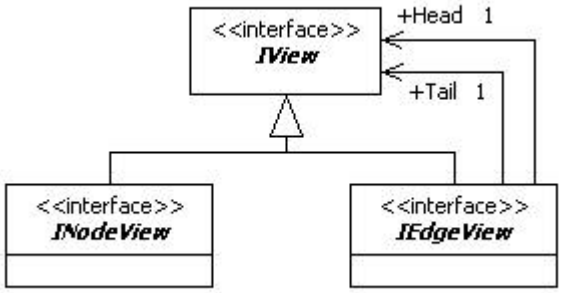
Интерфейс ITaggedValue определяет элементы тегов и предоставляет следующие свойства и методы. Пожалуйста обращайтесь к "Главе 7. Написание профиля UML" для детального ознакомления с тегами.

Свойство	Описание
ProfileName: String	Содержит имя профиля UML, который определяет текущее значение тега. * Только для чтения.
TagDefinitionSetName: String	Определяет имя набор определения тега, который содержит текущее значение тега. * Только для чтения.
Name: String	Определяет имя тега, определенного в профили UML. * Только для чтения.
DataValue: String	Определяет значение тега. * Только для чтения.
TaggedModel: IExtensibleModel	Определяет ссылку на элемент типа IExtensibleModel, содержащий тег. * Только для чтения.

Метод	Описание
GetTagDefinition: ITagDefinition	Возвращает элемент определения для тега.
GetTagDefinitionSet: ITagDefinitionSet	Возвращает элемент определения набора для текущего значения тега.
GetProfile: IProfile	Возвращает элемент профиля UML, который определяет текущий тег.

Элементы ViewCore

Интерфейсы группы ViewCore, унаследованные от IView , обеспечивают структурную платформу для всех представлений. Группа ViewCore содержит много интерфейсных типов. Этот раздел описывает интерфейсы INodeView и IEdgeView, которые являются самыми важными интерфейсами.



Имя интерфейса	Описание
INodeView	Интерфейсный тип верхнего уровня для представлений узлов.
IEdgeView	Интерфейсный тип верхнего уровня для представлений граней.

INodeView

Интерфейс INodeView - базовый тип интерфейсов представлений узлов. Представление узла - визуальный образ, который имеет видимую область, подобную образу класса. Интерфейс INodeView предоставляет следующие основные свойства.

Свойство	Описание
Left: Integer	Местоположение представления (Левая координата).
Top: Integer	Местоположение представления (Верхняя координата).
Width: Integer	Размер представления (Ширина).
Height: Integer	Размер представления (Высота).
MinWidth: Integer	Определяет минимальный размер представления (Ширина). * Только для чтения.
MinHeight: Integer	Определяет минимальный размер представления (Высота). * Только для чтения.
AutoResize: Boolean	Определяет наличие режима автоизменяющихся размеров представления.

Следующий пример показывает, как изменить местоположение и размер элемента INodeView.

```
var nodeView = ... // Получение ссылки на элемент INodeView
var l = nodeView.Left;
var t = nodeView.Top;
var w = nodeView.Width;
var h = nodeView.Height;
nodeView.Left = l * 2;
nodeView.Top = t * 2;
nodeView.Width = w * 2;
nodeView.Height = h * 2;
```

IEdgeView

Интерфейс IEdgeView - базовый тип представлений граней. Представление грани - визуальный образ на основе линии, подобный образу зависимости. Интерфейс IEdgeView предоставляет следующие основные свойства.

Свойство	Описание
LineStyle: LineStyleKind	Определяет тип линии.
Points: IPoints	Определяет координаты линии.
Tail: IView	Определяет представление в начальной точке линии.

Свойство	Описание
Head: IView	Определяет представление в конечной точке линии.

Следующие значения, определенные в нумераторе LineStyleKind, могут использоваться для обозначения типа линии представления грани.

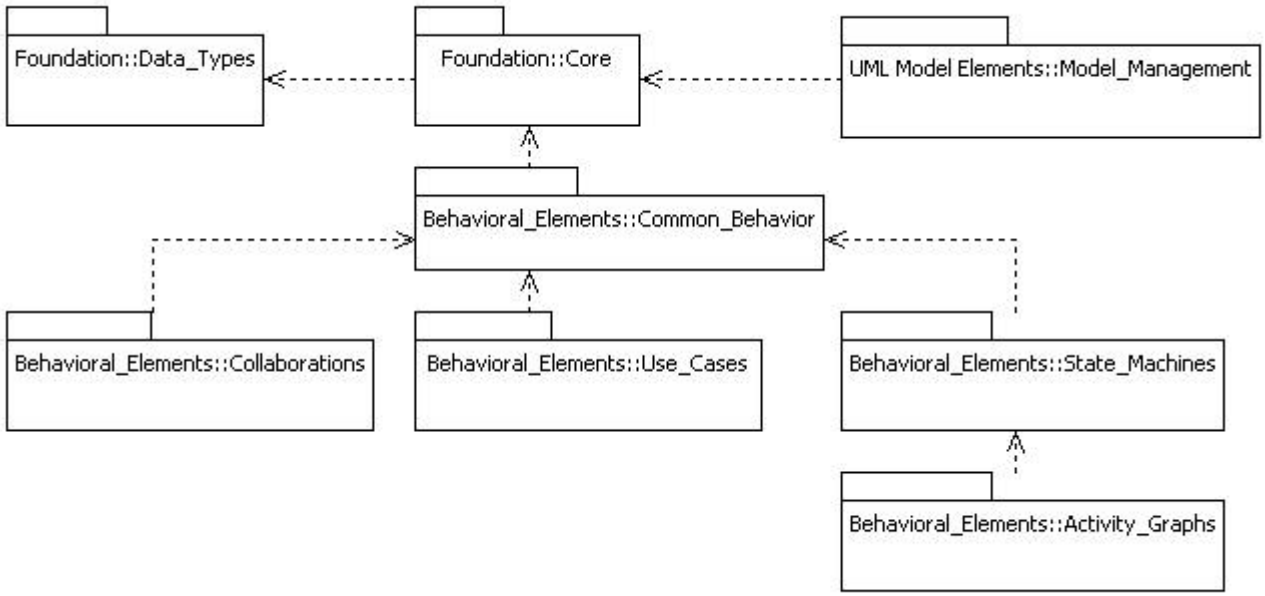
Значение	Описание
lsRectilinear	Прямолинейный тип линии.
lsOblique	Наклонный тип линии.

Следующий пример показывает, как изменить тип линии образа.

```
lsRectilinear = 0;
lsOblique = 1;
var view = ... // получение ссылки на элемент образа.
if (view.IsKindOf("EdgeView")){
view.LineStyle = lsRectilinear;
}
```

Доступ к модельным элементам UML

Группа Model Elements UML разделена на различные пакеты, как показано ниже. Отметим, что Model Elements UML - это реализация в StarUML™ элементов, определенных в спецификациях стандарта UML; они почти идентичны стандартным элементам UML. Здесь мы опустим детальное описание модельных элементов группы Model Elements UML.



Создание модельных элементов UML

Для создания модельного элемента UML, должен использоваться интерфейс IUMLFactory. Интерфейс IUMLFactory предоставляет методы создания не только модельных элементов UML но также и элементов диаграмм UML, представлений UML и всех остальных элементов UML. Объект типа IUMLFactory может быть получен через объект типа IStarUMLApplication, как показано ниже.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var facto = app.UMLFactory;
```

. . .

IUMLFactory обеспечивает следующие методы создания элементов модели UML.

Тип элемента	Метод создания
UMLModel	CreateModel(AOwner: UMLNamespace): IUMLModel
UMLSubsystem	CreateSubsystem(AOwner: UMLNamespace): IUMLSubsystem
UMLPackage	CreatePackage(AOwner: UMLNamespace): IUMLPackage
UMLClass	CreateClass(AOwner: UMLNamespace): IUMLClass
UMLInterface	CreateInterface(AOwner: UMLNamespace): IUMLInterface
UMLEnumeration	CreateEnumeration(AOwner: UMLNamespace): IUMLEnumeration
UMLSignal	CreateSignal(AOwner: UMLNamespace): IUMLSignal
UMLException	CreateException(AOwner: UMLNamespace): IUMLException
UMLComponent	CreateComponent(AOwner: UMLNamespace): IUMLComponent
UMLComponentInstance	CreateComponentInstance(AOwner: UMLNamespace): IUMLComponentInstance
UMLNode	CreateNode(AOwner: UMLNamespace): IUMLNode
UMLNodeInstance	CreateNodeInstance(AOwner: UMLNamespace): IUMLNodeInstance
UMLUseCase	CreateUseCase(AOwner: UMLNamespace): IUMLUseCase
UMLActor	CreateActor(AOwner: UMLNamespace): IUMLActor
UMLActivityGraph	CreateActivityGraph(AContext: UMLModelElement): IUMLActivityGraph
UMLStateMachine	CreateStateMachine(AContext: UMLModelElement): IUMLStateMachine
UMLCompositeState	CreateCompositeState(AOwnerState: UMLCompositeState): IUMLCompositeState
UMLCollaboration	CreateCollaboration(AOwner: UMLClassifier): IUMLCollaboration
UMLCollaboration	CreateCollaboration2(AOwner: UMLOperation): IUMLCollaboration
UMLCollaborationInstanceSet	CreateCollaborationInstanceSet(AOwner: UMLClassifier): IUMLCollaborationInstanceSet
UMLCollaborationInstanceSet	CreateCollaborationInstanceSet2(AOwner: UMLOperation): IUMLCollaborationInstanceSet
UMLInteraction	CreateInteraction(ACollaboration: UMLCollaboration): IUMLInteraction
UMLInteractionInstanceSet	CreateInteractionInstanceSet(ACollaborationInstanceSet: UMLCollaborationInstanceSet): IUMLInteractionInstanceSet
UMLActionState	CreateActionState(AOwnerState: UMLCompositeState): IUMLActionState
UMLSubactivityState	CreateSubactivityState(AOwnerState: UMLCompositeState): IUMLSubactivityState
UMLPseudostate	CreatePseudostate(AOwnerState: UMLCompositeState): IUMLPseudostate
UMLFinalState	CreateFinalState(AOwnerState: UMLCompositeState): IUMLFinalState
UMLPartition	CreatePartition(AActivityGraph: UMLActivityGraph): IUMLPartition
UMLSubmachineState	CreateSubmachineState(AOwnerState: UMLCompositeState): IUMLSubmachineState
UMLAttribute	CreateAttribute(AClassifier: UMLClassifier): IUMLAttribute
UMLAttribute	CreateQualifier(AAssociationEnd: UMLAssociationEnd): IUMLAttribute

Тип элемента	Метод создания
UMLOperation	CreateOperation(AClassifier: UMLClassifier): IUMLOperation
UMLParameter	CreateParameter(ABehavioralFeature: UMLBehavioralFeature): IUMLParameter
UMLTemplateParameter	CreateTemplateParameter(AClass: UMLClass): IUMLTemplateParameter
UMLTemplateParameter	CreateTemplateParameter2(ACollaboration: UMLCollaboration): IUMLTemplateParameter
UMLEnumerationLiteral	CreateEnumerationLiteral(AEnumeration: UMLEnumeration): IUMLEnumerationLiteral
UMLUninterpretedAction	CreateEntryAction(AState: UMLState): IUMLUninterpretedAction
UMLUninterpretedAction	CreateDoAction(AState: UMLState): IUMLUninterpretedAction
UMLUninterpretedAction	CreateExitAction(AState: UMLState): IUMLUninterpretedAction
UMLUninterpretedAction	CreateEffect(ATransition: UMLTransition): IUMLUninterpretedAction
UMLSignalEvent	CreateSignalEvent(ATransition: UMLTransition): IUMLSignalEvent
UMLCallEvent	CreateCallEvent(ATransition: UMLTransition): IUMLCallEvent
UMLTimeEvent	CreateTimeEvent(ATransition: UMLTransition): IUMLTimeEvent
UMLChangeEvent	CreateChangeEvent(ATransition: UMLTransition): IUMLChangeEvent
UMLClassifierRole	CreateClassifierRole(ACollaboration: UMLCollaboration): IUMLClassifierRole
UMLObject	CreateObject(ACollaborationInstanceSet: UMLCollaborationInstanceSet): IUMLObject
UMLObject	CreateObject2(AOwner: UMLNamespace): IUMLObject
UMLTransition	CreateTransition(AStateMachine: UMLStateMachine; Source: UMLStateVertex; Target: UMLStateVertex): IUMLTransition
UMLDependency	CreateDependency(AOwner: UMLNamespace; Client: UMLModelElement; Supplier: UMLModelElement): IUMLDependency
UMLAssociation	CreateAssociation(AOwner: UMLNamespace; End1: UMLClassifier; End2: UMLClassifier): IUMLAssociation
UMLAssociationClass	CreateAssociationClass(AOwner: UMLNamespace; AAssociation: UMLAssociation; AClass: UMLClass): IUMLAssociationClass
UMLGeneralization	CreateGeneralization(AOwner: UMLNamespace; Parent: UMLGeneralizableElement; Child: UMLGeneralizableElement): IUMLGeneralization
UMLLink	CreateLink(ACollaborationInstanceSet: UMLCollaborationInstanceSet; End1: UMLInstance; End2: UMLInstance): IUMLLink
UMLAssociationRole	CreateAssociationRole(ACollaboration: UMLCollaboration; End1: UMLClassifierRole; End2: UMLClassifierRole): IUMLAssociationRole
UMLStimulus	CreateStimulus(AInteractionInstanceSet: UMLInteractionInstanceSet; Sender: UMLInstance; Receiver: UMLInstance; Kind: UMLFactoryMessageKind): IUMLStimulus
UMLStimulus	CreateStimulus2(AInteractionInstanceSet: UMLInteractionInstanceSet; Sender: UMLInstance; Receiver: UMLInstance; CommunicationLink: UMLLink; Kind: UMLFactoryMessageKind): IUMLStimulus
UMLMessage	CreateMessage(AInteraction: UMLInteraction; Sender: UMLClassifierRole; Receiver: UMLClassifierRole; Kind: UMLFactoryMessageKind): IUMLMessage
UMLMessage	CreateMessage2(AInteraction: UMLInteraction; Sender:

Тип элемента	Метод создания
	UMLClassifierRole; Receiver: UMLClassifierRole; CommunicationConnection: UMLAssociationRole; Kind: UMLFactoryMessageKind): IUMLMessage
UMLInclude	CreateInclude(AOwner: UMLNamespace; Includer: UMLUseCase; Includee: UMLUseCase): IUMLInclude
UMLExtend	CreateExtend(AOwner: UMLNamespace; Extender: UMLUseCase; Extendee: UMLUseCase): IUMLExtend
UMLRealization	CreateRealization(AOwner: UMLNamespace; Client: UMLModelElement; Supplier: UMLModelElement): IUMLRealization

Следующий пример иллюстрирует создание модельных элементов UML с использованием IUMLFactory.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var facto = app.UMLFactory;
var pjt = app.GetProject();
var mdlElem = facto.CreateModel(pjt); // Create UMLModel element.
var pkgElem = facto.CreatePackage(mdlElem); // Create UMLPackage element.
var clsElem1 = facto.CreateClass(pkgElem); // Create UMLClass element.
var clsElem2 = facto.CreateClass(pkgElem); // Create UMLClass element.
var attrElem = facto.CreateAttribute(clsElem1); // Create UMLAttribute element.
var opElem = facto.CreateOperation(clsElem1); // Create UMLOperation element.
var paramElem1 = facto.CreateParameter(opElem); // Create UMLParameter element.
var paramElem2 = facto.CreateParameter(opElem); // Create UMLParameter element.
paramElem1.TypeExpression = "String";
paramElem2.Type_ = clsElem2;
...
```

Удаление модельных элементов UML

Метод DeleteModel интерфейса IStarUMLApplication может использоваться, чтобы удалять элементы модели UML. Метод CanDelete интерфейса IModel может использоваться, чтобы проверять, может ли текущий модельный элемент быть удален. Если текущий модельный элемент имеет статус "только для чтения", метод CanDelete возвращает "false". Помните, когда модельный элемент удаляется, все его элементы более низкого уровня, и все его визуальные представления автоматически удаляются вместе с ним. Следующий пример, продолжение примера, приведённого выше, показывает удаление класса.

```
...
if (clsElem1.CanDelete() == true){
app.DeleteModel(clsElem1);
}
...
```

Работа с диаграммами UML

Создание элемента диаграммы UML

IUMLFactory может использоваться, чтобы создавать диаграммы UML аналогично созданию элементов модели UML. IUMLFactory обеспечивает следующие связанные с диаграммами методы создания.

Тип диаграммы	Метод создания
UMLClassDiagram	CreateClassDiagram(AOwner: Model): IUMLClassDiagram

Тип диаграммы	Метод создания
UMLUseCaseDiagram	CreateUseCaseDiagram(AOwner: Model): IUMLUseCaseDiagram
UMLSequenceDiagram	CreateSequenceDiagram(AOwner: UMLInteractionInstanceSet): IUMLSequenceDiagram
UMLSequenceRoleDiagram	CreateSequenceRoleDiagram(AOwner: UMLInteraction): IUMLSequenceRoleDiagram
UMLCollaborationDiagram	CreateCollaborationDiagram(AOwner: UMLInteractionInstanceSet): IUMLCollaborationDiagram
UMLCollaborationRoleDiagram	CreateCollaborationRoleDiagram(AOwner: UMLInteraction): IUMLCollaborationRoleDiagram
UMLStatechartDiagram	CreateStatechartDiagram(AOwner: UMLStateMachine): IUMLStatechartDiagram
UMLActivityDiagram	CreateActivityDiagram(AOwner: UMLActivityGraph): IUMLActivityDiagram
UMLComponentDiagram	CreateComponentDiagram(AOwner: Model): IUMLComponentDiagram
UMLDeploymentDiagram	CreateDeploymentDiagram(AOwner: Model): IUMLDeploymentDiagram

Методы создания диаграмм UML почти идентичны методам создания модельных элементов UML. Но у диаграммы UML есть одно отличие - представления диаграмм, автоматически создаются при создании самих диаграмм.

Следующий пример создаёт диаграмму и обращается к автоматически созданному представлению диаграммы UML.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var pkgElem = ... // модельный элемент верхнего уровня, содержащий диаграмму UML.
var dgmElem = facto.CreateClassDiagram(pkgElem); // создаёт UMLClassDiagram.
var dgmViewElem = dgmElem.DiagramView; // автоматически созданное представление диаграммы.
app.OpenDiagram(dgmElem);
...
```

Удаление диаграммы UML

Так как диаграммы UML являются элементами моделирования, они могут быть удалены, используя метод DeleteModel интерфейса IStarUMLApplication, подобно удалению других элементов. Метод CanDelete интерфейса IModel может использоваться, чтобы проверять, может ли диаграмма быть удалена.

Обработка представлений

Создание представления

Объект IUMLFactory может использоваться также для создания представлений модельных элементов. Предусмотрены следующие методы IUMLFactory, связанные с созданием представлений.

Тип представления	Метод создания
UMLNoteView	CreateNoteView(ADiagramView: DiagramView): IUMLNoteView
UMLNoteLinkView	CreateNoteLinkView(ADiagramView: DiagramView; ANote: UMLNoteView; LinkTo: View): IUMLNoteLinkView

Тип представления	Метод создания
UMLTextView	CreateTextView(ADiagramView: DiagramView): IUMLTextView
UMLModelView	CreateModelView(ADiagramView: DiagramView; AModel: UMLModel): IUMLModelView
UMLSubsystemView	CreateSubsystemView(ADiagramView: DiagramView; AModel: UMLSubsystem): IUMLSubsystemView
UMLPackageView	CreatePackageView(ADiagramView: DiagramView; AModel: UMLPackage): IUMLPackageView
UMLClassView	CreateClassView(ADiagramView: DiagramView; AModel: UMLClass): IUMLClassView
UMLInterfaceView	CreateInterfaceView(ADiagramView: DiagramView; AModel: UMLInterface): IUMLInterfaceView
UMLEnumerationView	CreateEnumerationView(ADiagramView: DiagramView; AModel: UMLEnumeration): IUMLEnumerationView
UMLSignalView	CreateSignalView(ADiagramView: DiagramView; AModel: UMLSignal): IUMLSignalView
UMLExceptionView	CreateExceptionView(ADiagramView: DiagramView; AModel: UMLException): IUMLExceptionView
UMLComponentView	CreateComponentView(ADiagramView: DiagramView; AModel: UMLComponent): IUMLComponentView
UMLComponentInstanceView	CreateComponentInstanceView(ADiagramView: DiagramView; AModel: UMLComponentInstance): IUMLComponentInstanceView
UMLNodeView	CreateNodeView(ADiagramView: DiagramView; AModel: UMLNode): IUMLNodeView
UMLNodeInstanceView	CreateNodeInstanceView(ADiagramView: DiagramView; AModel: UMLNodeInstance): IUMLNodeInstanceView
UMLActorView	CreateActorView(ADiagramView: DiagramView; AModel: UMLActor): IUMLActorView
UMLUseCaseView	CreateUseCaseView(ADiagramView: DiagramView; AModel: UMLUseCase): IUMLUseCaseView
UMLCollaborationView	CreateCollaborationView(ADiagramView: DiagramView; AModel: UMLCollaboration): IUMLCollaborationView
UMLCollaborationInstanceSetView	CreateCollaborationInstanceSetView(ADiagramView: DiagramView; AModel: UMLCollaborationInstanceSet): IUMLCollaborationInstanceSetView
UMLGeneralizationView	CreateGeneralizationView(ADiagramView: DiagramView; AModel: UMLGeneralization; Parent: View; Child: View): IUMLGeneralizationView
UMLAssociationView	CreateAssociationView(ADiagramView: DiagramView; AModel: UMLAssociation; End1: View; End2: View): IUMLAssociationView
UMLAssociationClassView	CreateAssociationClassView(ADiagramView: DiagramView; AModel: UMLAssociationClass; AssociationView: View; ClassView: View): IUMLAssociationClassView
UMLDependencyView	CreateDependencyView(ADiagramView: DiagramView; AModel: UMLDependency; Client: View; Supplier: View): IUMLDependencyView
UMLRealizationView	CreateRealizationView(ADiagramView: DiagramView; AModel: UMLRealization; Client: View; Supplier: View): IUMLRealizationView
UMLIncludeView	CreateIncludeView(ADiagramView: DiagramView; AModel: UMLInclude; Base: View; Addition: View): IUMLIncludeView
UMLExtendView	CreateExtendView(ADiagramView: DiagramView; AModel: UMLExtend; Base:

Тип представления	Метод создания
	View; Extension: View): IUMLExtendView
UMLColObjectView	CreateObjectView(ADiagramView: DiagramView; AModel: UMLObject): IUMLColObjectView
UMLSeqObjectView	CreateSeqObjectView(ADiagramView: UMLSequenceDiagramView; AModel: UMLObject): IUMLSeqObjectView
UMLColClassifierRoleView	CreateClassifierRoleView(ADiagramView: DiagramView; AModel: UMLClassifierRole): IUMLColClassifierRoleView
UMLSeqClassifierRoleView	CreateSeqClassifierRoleView(ADiagramView: UMLSequenceRoleDiagramView; AModel: UMLClassifierRole): IUMLSeqClassifierRoleView
UMLLinkView	CreateLinkView(ADiagramView: DiagramView; AModel: UMLLink; End1: View; End2: View): IUMLLinkView
UMLAssociationRoleView	CreateAssociationRoleView(ADiagramView: DiagramView; AModel: UMLAssociationRole; End1: View; End2: View): IUMLAssociationRoleView
UMLColStimulusView	CreateStimulusView(ADiagramView: UMLCollaborationDiagramView; AModel: UMLStimulus; LinkView: View): IUMLColStimulusView
UMLSeqStimulusView	CreateSeqStimulusView(ADiagramView: UMLSequenceDiagramView; AModel: UMLStimulus; Sender: View; Receiver: View): IUMLSeqStimulusView
UMLColMessageView	CreateMessageView(ADiagramView: UMLCollaborationRoleDiagramView; AModel: UMLMessage; AssociationRoleView: View): IUMLColMessageView
UMLSeqMessageView	CreateSeqMessageView(ADiagramView: UMLSequenceRoleDiagramView; AModel: UMLMessage; Sender: View; Receiver: View): IUMLSeqMessageView
UMLStateView	CreateStateView(ADiagramView: UMLStatechartDiagramView; AModel: UMLCompositeState): IUMLStateView
UMLSubmachineStateView	CreateSubmachineStateView(ADiagramView: UMLStatechartDiagramView; AModel: UMLSubmachineState): IUMLSubmachineStateView
UMLPseudostateView	CreatePseudostateView(ADiagramView: DiagramView; AModel: UMLPseudostate): IUMLPseudostateView
UMLFinalStateView	CreateFinalStateView(ADiagramView: DiagramView; AModel: UMLFinalState): IUMLFinalStateView
UMLActionStateView	CreateActionStateView(ADiagramView: UMLActivityDiagramView; AModel: UMLActionState): IUMLActionStateView
UMLSubactivityStateView	CreateSubactivityStateView(ADiagramView: UMLActivityDiagramView; AModel: UMLSubactivityState): IUMLSubactivityStateView
UMLSwimlaneView	CreateSwimlaneView(ADiagramView: UMLActivityDiagramView; AModel: UMLPartition): IUMLSwimlaneView
UMLTransitionView	CreateTransitionView(ADiagramView: DiagramView; AModel: UMLTransition; Source: View; Target: View): IUMLTransitionView

Следующий пример создает элементы типа IUMLClassView в представлении диаграммы классов, а также создает элементы IUMLDependencyView и IUMLAssociationView, которые связывают эти два элемента. Поскольку для того, чтобы создать представления, требуются соответствующие модельные элементы, эти модельные элементы также предварительно создаются .

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var factory = app.UMLFactory;
// Получение ссылок на существующие модельные элементы.
var rootElem = app.FindByPathname("::Logical View");
if (rootElem != null){

app.BeginUpdate();
try{
// Создание модельных элементов.
```

```
var class1 = factory.CreateClass(rootElem);
var class2 = factory.CreateClass(rootElem);
var dependency = factory.CreateDependency(rootElem, class1, class2);
var association = factory.CreateAssociation(rootElem, class1, class2);
var diagram = factory.CreateClassDiagram(rootElem);
var diagramView = diagram.DiagramView;
// Создание представлений.
var classView1 = factory.CreateClassView(diagramView, class1);
var classView2 = factory.CreateClassView(diagramView, class2);
var dependencyView = factory.CreateDependencyView(diagramView, dependency,
classView1, classView2);
var associationView = factory.CreateAssociationView(diagramView, association,
classView1, classView2);
// Выравнивание образов.
classView1.Left = 100;
classView1.Top = 100;
classView2.Left = 300;
classView2.Top = 100;
app.OpenDiagram(diagram);
}
finally{
app.EndUpdate();
}
}
```

Удаление представлений

Метод DeleteView интерфейса IStarUMLApplication может использоваться, чтобы удалять представления. Помните, что когда модельный элемент удаляется, его представления автоматически удаляются вместе с ним, но когда удаляется представление, соответствующий ему модельный элемент не удаляется.

Следующие пример показывает, как удалить представления, которые были созданы в предыдущем примере.

```
...
app.DeleteView(dependencyView);
app.DeleteView(associationView);
...
```

Использование API для объекта приложения

Управление объектом приложения

Объект StarUMLApplication

Первое, что нужно получить, чтобы использовать API StarUML™ - ссылку на объект StarUMLApplication. Все другие объекты могут быть получены через него.

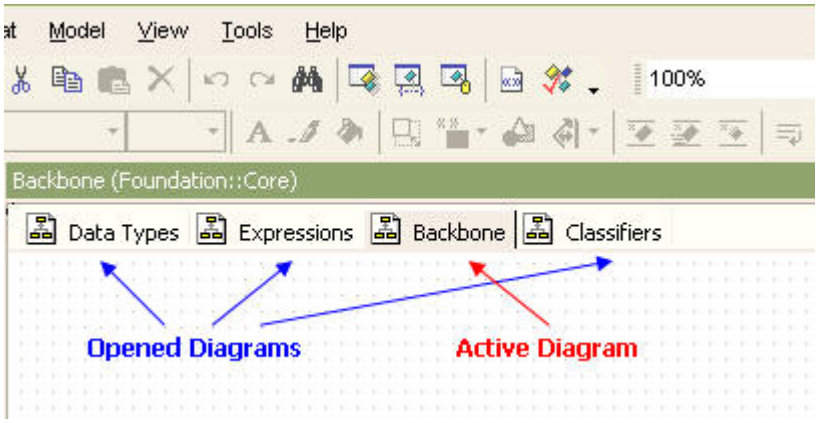
Интерфейс IStarUMLApplication - абстракция приложения StarUML™, содержит следующие методы.

- Методы акций пользователя (Undo, Redo, ClearHistory, BeginUpdate, EndUpdate, BeginGroupAction, EndGroupAction, ...)
- Методы редактирования элементов(Copy, Cut, Paste, ...)
- Методы удаления модельных элементов, представлений, диаграмм (DeleteModel, DeleteView, ...)

- Чтение значений опций (GetOptionValue)
- Методы просмотра журнала, сообщений и web-страниц (Log, AddMessageItem, NavigateWeb, ...)
- Управление диаграммами (OpenDiagram, CloseDiagram, ...)
- Другие методы (FindByPathname, SelectInModelExplorer, ...)

Управление открытыми диаграммами

В области диаграмм StarUML™ , открытые диаграммы располагаются на вкладках, как иллюстрировано ниже. Активизированную на конкретный момент времени диаграмму называют активной диаграммой.



Чтобы открыть диаграмму, используйте следующий код. Если диаграмма не является уже открытой, она откроется и автоматически станет активной. Если диаграмма уже открыта, она будет установлена как активная диаграмма.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var dgm = ... // присваивание открываемой диаграммы.
app.OpenDiagram(dgm);
Чтобы получить ссылку на открытые диаграммы, используйте методы GetOpenedDiagramCount и
GetOpenedDiagramAt.
var app = new ActiveXObject("StarUML.StarUMLApplication");
...
for (i=0; i<app.GetOpenedDiagramCount(); i++) {
var dgm = app.GetOpenedDiagramAt(i);
...
}
```

Открытые диаграммы могут быть закрыты. В этом случае может использоваться метод CloseDiagram. Используйте метод CloseAllDiagram, чтобы закрыть все диаграммы, или метод CloseActiveDiagram, чтобы закрыть только активную диаграмму.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var dgm = ... // Assign a diagram to close.
app.CloseDiagram(dgm);
```

Запись в журнал

Закладка [Output] в информационной области StarUML™ обеспечивает интерфейс для того, чтобы записывать и показывать журнал выполнения приложения пользователю.

Чтобы делать запись в секцию [Output] , используйте метод Log как показано в следующем примере.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
app.Log("This is a test log...");
```

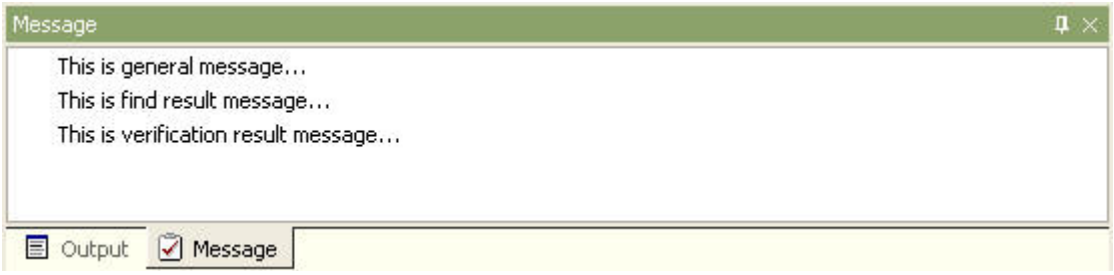
Работа с сообщениями

StarUML™ использует элементы сообщений, чтобы отобразить определенные сообщения пользователю. Элементы сообщений могут использоваться, например, чтобы уведомить о том, что искомые элементы не были найдены, или не прошли проверку на корректность. Есть три типа элементов сообщения: общие элементы, элементы результатов поиска и элементы результатов проверки на корректность.

Значение	Литерал	Описание
0	mkGeneral	Общие элементы сообщения.
1	mkFindResult	Элементы результатов поиска.
2	mkVerificationResult	Элементы результатов проверки на корректность.

При добавлении элемента сообщения, нужно указать тип сообщения и связанный с ним элемент проекта. Следующий пример показывает создание сообщений трех типов с различным содержанием, ссылающихся на проект. Результат показан на следующей иллюстрации.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
app.AddMessageItem(0, "This is general message...", app.GetProject());
app.AddMessageItem(1, "This is find result message...", app.GetProject());
app.AddMessageItem(2, "This is verification result message...", app.GetProject());
```



Двойное нажатие на сообщении автоматически выбирает связанный элемент в навигаторе модели, и если элемент отображается на диаграмме, эта диаграмма становится активной.

Поиск элемента по пути

Элемент может быть найден по своему полному имени (пути). Например, путь для элемента Class1, расположенного в Package2, который находится в Package1 - ":: Package1:: Package2:: Class1". Путь (полное имя) - цепочка имен элементов, связанных разделителем "::". Поиск всегда

начинается от проекта верхнего уровня.

Так как имя проекта верхнего уровня - всегда пустая строка, любое имя пути начинается с "::". Однако, можно опустить начальный разделитель "::". Другими словами, выражение типа "Package1:: Package2:: Class2" интерпретируется также как то, что указано выше. Следующий примера показывает получение ссылки на модельный элемент по его пути.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var elem = app.FindByPathname("::Logical View::Class1");
...
```

Управление изменениями в приложении

Когда пользователь вносит некоторое изменение или исполняет некоторую команду через API, соответствующая модификация модели отображается немедленно.

Однако, при выполнении сложных задач посредством API, когда должно быть выполнено много команд, синхронное отображение может уменьшить скорость обработки. В этом случае, лучше остановить отображение, выполнить различные сложные задачи, и только затем отобразить все изменения сразу. Объект StarUMLApplication обеспечивает такие функции через методы BeginUpdate и EndUpdate.

Пользователь может вызвать метод BeginUpdate перед выполнением сложной и длинной задачи, а метод EndUpdate - немедленно после её выполнения, чтобы отобразить изменения. Нужно учитывать, что никакие изменения не будут отображены вообще, если после вызова BeginUpdate метод EndUpdate не будет вызван из-за ошибок или других проблем при выполнении задачи. Для предотвращения таких проблем должен использоваться блок обработки исключений, как показано в следующем примере.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
...
app.BeginUpdate();
try {
... // Place tasks to process here.
}
finally {
app.EndUpdate(); // The finally block will be executed even if an exception occurs in the try
block.
}
...
```

Для того, чтобы указывать конец модификаций и вызывать отображение изменений, может также использоваться метод EndUpdate2 вместо метода EndUpdate. Оба метода имеют одинаковый эффект, но EndUpdate2 обеспечивает более детальный контроль. Этот метод имеет следующие два аргумента.

Аргумент	Тип	Описание
Completely Rebuild	Boolean	Восстанавливает все древовидные структуры, отображаемые в навигаторе модели. Установка значения этого аргумента в 'True' позволяет более быстро отобразить изменения, если они включают создание или модификацию большого количества модельных элементов. Метод EndUpdate () аналогичен установке этого параметра в 'Ложь'.
UseUpdate Lock	Boolean	Отображение операций вставки/удаления/изменения в дереве элементов модельного навигатора выполняется каждый раз. Другими словами, изменения в дереве, визуально не отображаемые в графическом интерфейсе пользователя, тем не менее синхронно выполняются. Установка значения этого аргумента в 'True', когда модель очень большая, может привести к относительно более долгому выполнению

Аргумент	Тип	Описание
		изменений, даже если число изменяемых модельных элементов небольшое. Когда это значение 'True', время выполнения процесса пропорционально общему количеству модельных элементов, а не количеству изменяемых модельных элементов. Метод EndUpdate () аналогичен установке этого значения в 'True'.

Использование групп акций

Возможно отменить или восстановить любые выполненные пользователем действия. То же самое касается любых команд, выполненных через API. Если команда выполнена дважды, и пользователь желает отменить результат, отмена также должна быть выполнена дважды. Однако, бывают случаи, когда пользователь хочет комбинировать различные команды и обрабатывать их как единое действие. Например, для отмены изменения значения определенного свойства, нужно вернуть свойству то значение, которое у него было до вызова функций Get и Set. Однако, в комбинации с функциями Get/Set могло выполняться много команд. В таком случае, несколько команд могут быть объединены в группу и обработаны как одно действие.

Объект StarUMLApplication позволяет выполнять группы команд как единое действие, используя методы BeginGroupAction и EndGroupAction. При вызове метода BeginGroupAction, будет создана новая виртуальная группа акций.

Все действия, выполненные после этого вызова, будут добавлены в группу, а когда будет вызван метод EndGroupAction, группировка действий, будет закончена. После выполнения BeginGroupAction нужно вызвать EndGroupAction, даже если произошла ошибка в действиях, включенных в группу. Поэтому должен использоваться блок обработки прерываний. Действие группы интерпретируется как единое действие при выполнении отмен или восстановлений.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
...
app.BeginGroupAction();
try {
...
}
finally {
app.EndGroupAction();
}
...
```

При вызове метода BeginGroupAction, результат будет тот же самый, как при вызове BeginUpdate. Аналогично, вызов метода EndGroupAction приводит к тому же результату, что и вызов EndUpdate. Другими словами, изменения не будут приняты, пока группа должным образом не будет закончена. Поэтому методы BeginUpdate или EndUpdate не должны использоваться между BeginGroupAction и EndGroupAction.

Обработка выбранных элементов

StarUML™ предоставляет способ получения информации относительно модельных элементов или представлений, выбранных пользователем, а также позволяет выбирать некоторые элементы принудительно. Все эти функции определены в интерфейсе ISelectionManager.

Получение выбранных элементов

Чтобы получить список модельных элементов или представлений, выбранных в данный момент, сначала должна быть получена ссылка на SelectionManager. Затем нужно написать код подобный следующему.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var selmgr = app.SelectionManager;
// Список выбранных элементов
for (i=0; i<selmgr.GetSelectedModelCount(); i++) {
var m = selmgr.GetSelectedModelAt(i);
...
}
// Список выбранных представлений.
for (i=0; i<selmgr.GetSelectedViewCount(); i++) {
var v = selmgr.GetSelectedViewAt(i);
...
}
```

Получение активной диаграммы

Можно получить ссылку на активную диаграмму (диаграмму, которая в данный момент отображена на экране). Диаграмма всегда управляется двумя отдельными объектами: Diagram и DiagramView. Ссылки и на объект Diagram и на объект DiagramView могут быть получены отдельно.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var selmgr = app.SelectionManager;
var dgm = selmgr.ActiveDiagram // Диаграмма, которая становится активной
var dgmview = selmgr.ActiveDiagram // объект DiagramView для активной диаграммы
```

Выбор модельных элементов

Чтобы выбрать определенные элементы (например, класс, интерфейс, компонент...), используется метод SelectModel. Вызов этого метода сбрасывает все выбранные до этого элементы и устанавливает выбор только одного указанного элемента. Чтобы сохранить текущий выбор и добавить к выбору другие модельные элементы, должен использоваться метод SelectAdditionalModel.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var selmgr = app.SelectionManager;
var m = ... // Присваиваем ссылку на выбираемый модельный элемент.
...
selmgr.SelectModel(m); // Выбираем только элемент 'm'.
...
selmgr.SelectAdditionalModel(m); // Добавляем элемент 'm' к выбранным.
...
```

Чтобы отменить выбор модельных элементов, используйте метод DeselectModel как показано в примере ниже.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var selmgr = app.SelectionManager;
var m = ... // Присваиваем ссылку на элемент, который будет деактивирован.
...
selmgr.DeselectModel(m); // Деактивируем элемент 'm'.
...
selmgr.DelselectAllModels(); // Деактивируем все элементы.
...
```

Выбор представлений

Чтобы выбирать визуальные представления, отображаемые на диаграмме, используйте метод SelectView. Вызов этого метода деактивирует все выбранные на данный момент визуальные образы и выбирает только один. Чтобы, не снимая текущего выбора, добавлять визуальные образы к уже выбранным, должен использоваться метод SelectAdditionalView.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var selmgr = app.SelectionManager;
var v = ...
...
selmgr.SelectView(v);
...
selmgr.SelectAdditionalView(v);
...
```

Чтобы отменить выбор представлений, используйте метод DeselectView как показано в примере ниже.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var selmgr = app.SelectionManager;
var v = ... // Присваиваем ссылку на элемент.
...
selmgr.DeselectView(v); // Снимаем выбор образа 'v'.
...
selmgr.DelselectAllViews(); // Снимаем выбор всех элементов.
...
```

Выбор области диаграммы

Визуальные образы могут быть выбраны, посредством ввода координат области выбора в активной диаграмме. Используйте метод SelectArea, чтобы сделать это, а также метод SelectAdditionalArea, чтобы добавить элементы к выбранным через дополнительную область. Следующий пример выбирает все визуальные образы, расположенные в пределах области координат (100, 100, 500, 300) на активной диаграмме.

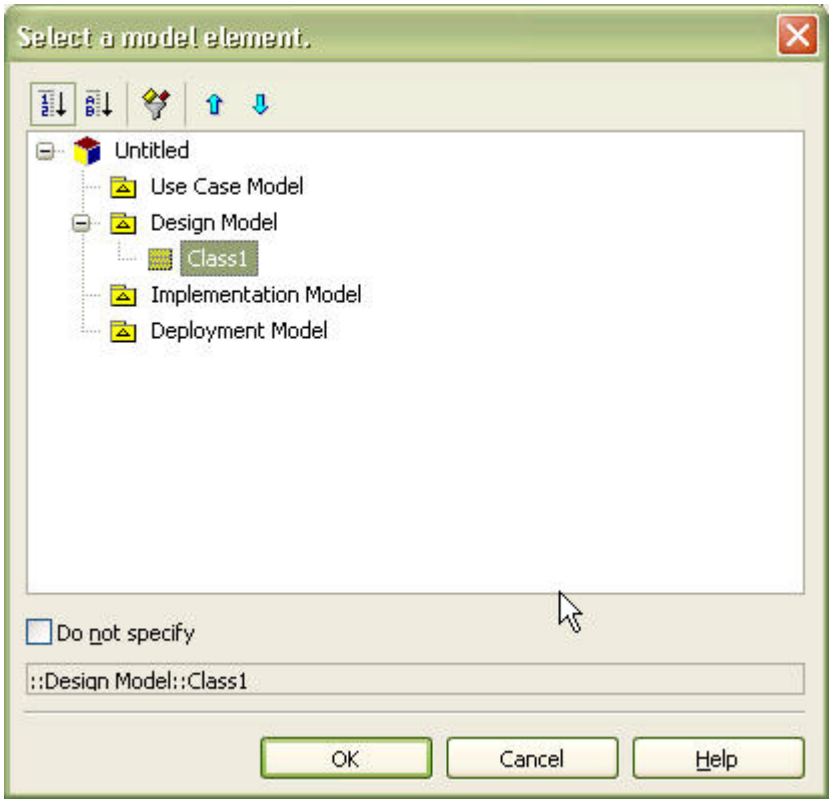
```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var selmgr = app.SelectionManager;
selmgr.SelectArea(100, 100, 500, 300);
```

Использование диалога выбора элементов

StarUML™ предоставляет два диалога для того, чтобы выбирать определенные элементы: ElementSelector, с древовидным изображением иерархии элементов, и ElementListSelector с перечнем элементов в виде списка. ElementSelector - используется чаще, поскольку он позволяет выбирать элементы в иерархической структуре, аналогичной навигатору модели. ElementListSelector обычно используется, если нужно выбирать элементы одного типа.

Работа с объектом ElementSelector

ElementSelector - диалог, который отображает иерархию элементов и позволяет выбрать элемент, как показано на иллюстрации ниже. Пользователь может выбрать элемент или не выбрать ничего (установив пустое возвращаемое значение).



Ссылка на объект ElementSelector может быть получена через объект StarUMLApplication как показано в примере ниже.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var sel_dlg = app.ElementSelector;
```

Диалог ElementSelector предоставляет следующие свойства и методы.

Свойство	Описание
AllowNull: Boolean	Определяет, позволено ли возвращать пустое значение.

Метод	Описание
Filter(Filtering: ElementFilteringKind)	Определяет какие элементы моделирования будут отображены. Значение может быть одним из следующих. • fkAll (0): Показывать все элементы. • fkPackages (1): Показывать только элементы типа UMLPackage (UMLPackage, UMLModel, UMLSubsystem). • fkClassifiers (2): Показывать только элементы типа UMLClassifier .
ClearSelectableModels	Очищает список выбранных типов элементов.
AddSelectableModel(ClassName: String)	Добавляет указанный тип к списку выбираемых типов элементов. Пример значения параметра: "UMLClass"
RemoveSelectableModel(ClassName: String)	Удаляет выбранный тип из списка выбираемых типов элементов. Пример значения параметра: "UMLClass"
Execute(Title: String): Boolean	Выполняет диалог. Устанавливает указанный заголовок диалога.
GetSelectedModel: IModel	Возвращает ссылку на выбранный пользователем элемент.

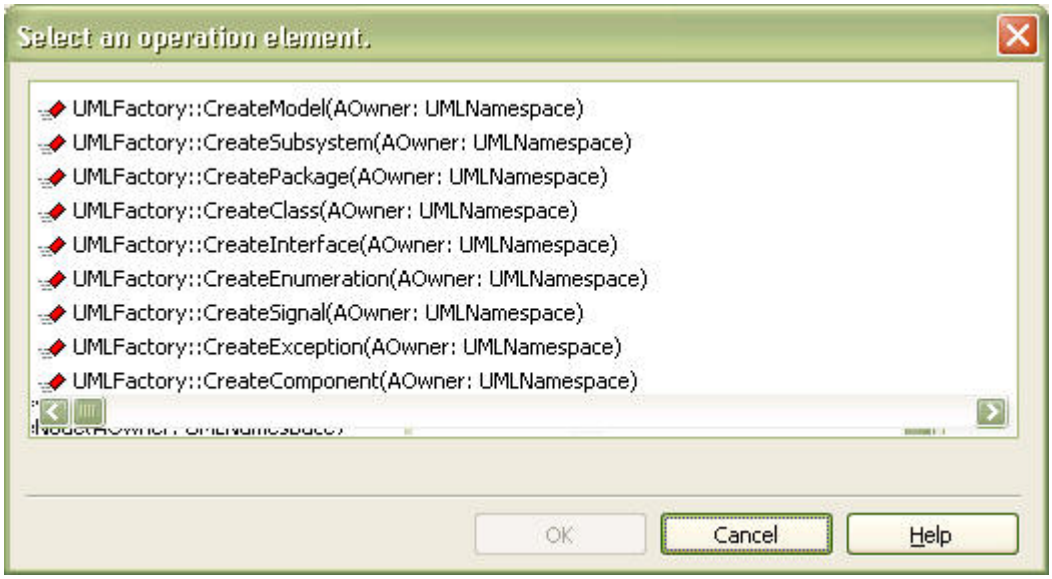
Следующий пример показывает весь процесс выполнения диалога ElementSelector и получение

выбранных элементов.

```
fkClassifiers = 2;
var app = new ActiveXObject("StarUML.StarUMLApplication");
var sel_dlg = app.ElementSelector;
sel_dlg.AllowNull = false;
sel_dlg.Filter(fkClassifiers)
sel_dlg.ClearSelectableModels();
sel_dlg.AddSelectableModel("UMLModel");
sel_dlg.AddSelectableModel("UMLSubsystem");
sel_dlg.AddSelectableModel("UMLPackage");
if (sel_dlg.Execute("Select a classifier type element.")){
var elem = sel_dlg.GetSelectedModel;
...
}
else{
// If canceled, ...
}
```

Работа с объектом ElementListSelector

ElementListSelector - диалог, который отображает элементы в виде одномерного списка и позволяет просматривать их и выбрать элемент.



Ссылка на объект диалога ElementListSelector может быть получена через объект StarApplication как показано в примере ниже.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var sel_dlg = app.ElementListSelector;
```

Диалог ElementListSelector предоставляет следующее свойство и методы.

Свойство	Описание
AllowNull: Boolean	Определяет, позволено ли возвращать пустое значение.

Метод	Описание
ClearListElements	Очищает список.
AddListElement(AModel: IModel)	Добавляет элемент, определенный параметром в список.

Метод	Описание
AddListElementsByCollection(AModel: IModel; CollectionName: String; ShowInherited: Boolean)	Добавляет элементы указанной коллекции указанного элемента, в список. Параметр 'ShowInherited' определяет, нужно ли проследить структуру наследования указанного элемента и добавить элементы аналогичных коллекций из элементов верхних уровней в список.
AddListElementsByClass(MetaClassName: String; IncludeChildInstances: Boolean)	Добавляет элементы типов, определенных параметром, в список. Если параметр 'IncludeChildInstances'='true', дочерние элементы выбранного типа, также добавляются в список.
Execute(Title: String): Boolean	Выполняет диалог. Устанавливает указанный заголовок диалога.
GetSelectedModel: IModel	Возвращает ссылку на выбранный пользователем элемент.

Следующий пример вызывает диалог ElementListSelector, и запрашивает у пользователя операцию указанного класса. Так как параметр "ShowInherited" установлен в "true", то любая операция из коллекций родительских классов (если таковые есть у выбранного класса) также может быть выбрана.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var sel_dlg = app.ElementListSelector;
sel_dlg.AllowNull = false;
sel_dlg.ClearListElements();
var class = ... // Получение ссылки на класс.
sel_dlg.AddListElementsByCollection(class, "Operations", true);
if (sel_dlg.Execute("Select an operation element.")){
var selElem = sel_dlg.GetSelectedModel;
...
}
else{
// If canceled, ...
}
```

Пример выше использовал метод AddListElementsByCollection. Следующий пример использует метод AddListElementsByClass. Так как параметр "IncludeChildInstances" "истинен", элементы выбранных типов, и все их дочерние элементы добавляются в список.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var sel_dlg = app.ElementListSelector;
sel_dlg.AllowNull = false;
sel_dlg.ClearListElements();
sel_dlg.AddListElementsByClass("UMLClassifier", true);
if (sel_dlg.Execute("Select a classifier type element.")){
var selElem = sel_dlg.GetSelectedModel;
...
}
else{
// If canceled ...
}
```

Использование API для метаобъектов

Этот раздел описывает концепцию метамодельных элементов и их использование. Как отмечалось в "Главе 2. Архитектура StarUML" метамодельные элементы - это элементы, которые принадлежат пакету "Non_Modeling Elements:: MetaModeling".

Основные понятия мета модели

Методы метамодели StarUML обеспечивает доступ на метауровне к модельным элементам, описанным в предыдущих разделах. Короче говоря, метамодельные элементы - это элементы, которые определяют эти элементы моделирования. Использование метамодельных элементов

позволяет получить доступ к номенклатуре каждого элемента моделирования и к информации о модельных элементах открытого проекта. Хотя концепция метамодели может показаться сложной для неопытных пользователей, строго рекомендуется, прочесть следующие описания, поскольку знание метамодели очень важно при использовании StarUML™.

Простой пример использования метамодели

Перед объяснением концепции метамодели, позвольте нам рассмотреть следующий простой пример, в качестве краткого обзора использования метамодельных элементов StarUML™. Сначала, предположим, что мы должны получить через API список всех классов в выполняющемся в настоящее время приложении StarUML™. Хотя поиск можно провести от проектного элемента самого верхнего уровня до всех элементов самого низкого уровня, использование метамодельных элементов может упростить процесс. Посмотрите на следующий код.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var meta = app.MetaModel;
var metaClass = meta.FindMetaClass("UMLClass");
for (var i = 0; i < metaClass.GetInstanceCount(); i++){
var AClassElem = metaClass.GetInstanceAt(i);
...
}
```

Этот пример использует метамодельные элементы, чтобы получить ссылки на все элементы классов. Модельный элемент называемый "UMLClass", передается как параметр в метод IMetaModel.FindMetaClass чтобы обращаться к элементам-классам. Параметр может быть заменен на "UMLAttribute", если требуется обработать список всех атрибутов. Другими словами, этот подход может быть применен ко всем модельным элементам тем же самым способом.

Обратите внимание: См. "Приложение B. Список модельных элементов UML" для ознакомления с именами элементов.

Второй пример показывает, как обратиться к информации о модельных элементах. Как узнать, какие свойства класса - как модельного элемента UML - присутствуют в коде реализации программы? Это вопрос не о том, какие атрибуты определены в разработанном пользователем классе, а какие атрибуты определены в элементе "Class", который является модельным элементом UML. Например, элемент "Class" имеет свойства "Name", "Visibility" и "IsAbstract".

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var meta = app.MetaModel;
var metaClass = meta.FindMetaClass("UMLClass");
for (var i = 0; i < metaClass.GetMetaAttributeCount(); i++){
var metaAttr = metaClass.GetMetaAttributeAt(i);
var attrName = metaAttr.Name;
...
}
```

Этот пример получает имена всех атрибутов, принадлежащих модельному элементу "UMLClass", т.е. имена свойств класса. Точно так же как в первом примере, параметр для метода IMetaModel.FindMetaClass может быть заменен, чтобы выполнить ту же самую задачу для других элементов UML.

Архитектура метамодели UML

Этот раздел кратко представляет архитектуру метамодели UML. Он нужен, чтобы понять метамодель StarUML™.

OMG (Ассоциация объектно-ориентированного управления) использует метод, называемый метамоделирование архитектуры, для того, чтобы определить спецификации элементов UML. Эта метамоделирующая архитектура состоит из следующих слоев.

- Meta-metamodel (метаметамодель)
- Metamodel (метамодель)
- Model (модель)
- User Objects (объекты пользователя)

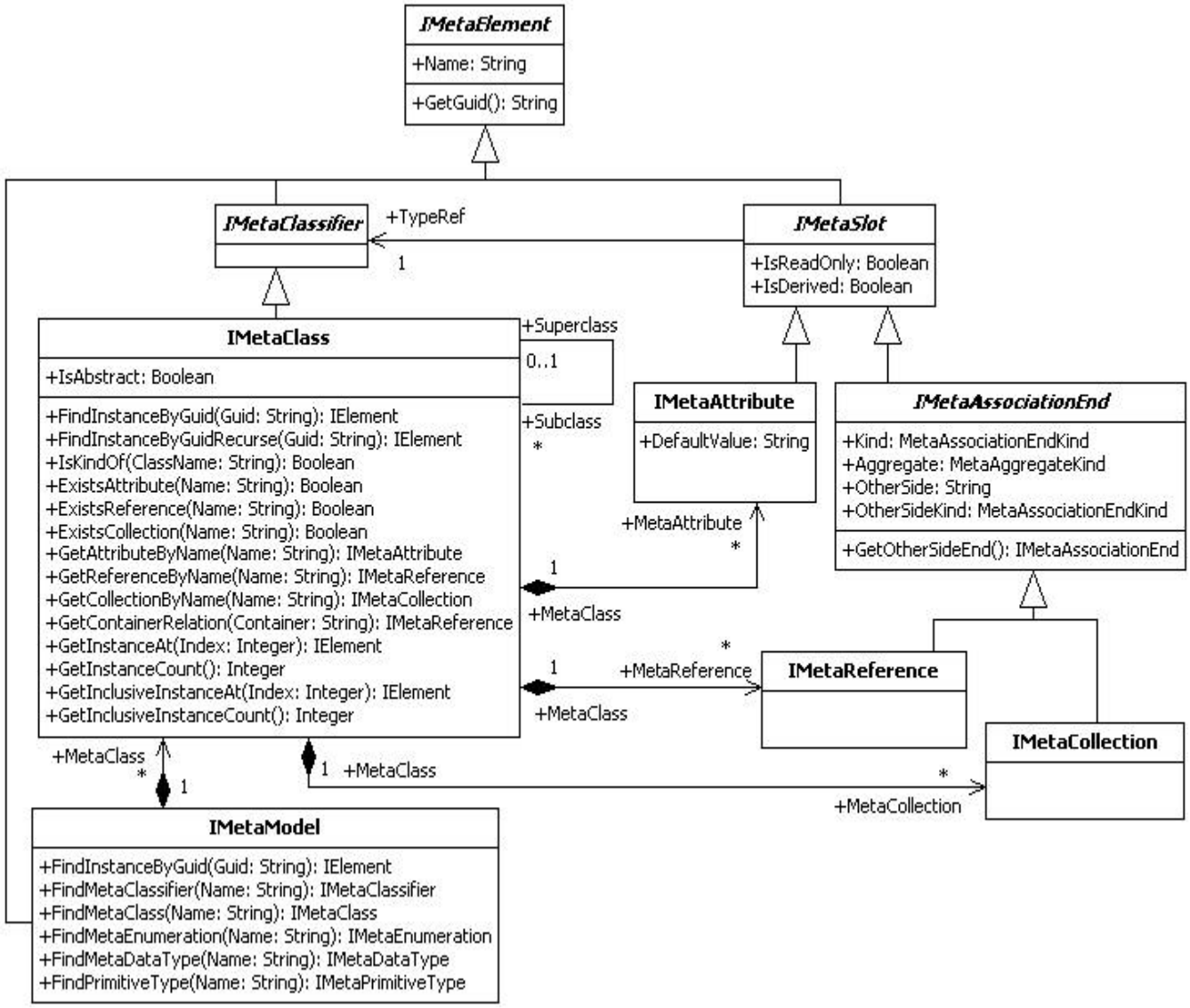
Определения элементов моделирования UML, описанные в Спецификации UML относятся к категории метамодели. Другими словами, обычные элементы UML подобные пакету, классу, варианту использования и актору - элементы метамодели. А элементы UML, которые созданы в процессе моделирования программного обеспечения, то есть, классы, которые называют как "Class1" или "Class2" - инстанции метамодели, которые относятся к категории модели. Более точно выражаясь, "Class1" и "Class2" - это инстанции метаэлемента "класс" ("UMLClass" в StarUML™).

Базовым слоем для того, чтобы определять метаэлементы UML подобные пакету, классу, варианту использования и актору, является метаметамодель; метаэлементы StarUML™ относятся к этому слою метаметамодели. Другими словами, все элементы моделирования могут быть интерпретированы как инстанции типа MetaClass, объясненного ниже. Метамодель StarUML™ облегчает доступ к модельным элементам на метауровне, исключая необходимость их определять.

Организация метамодели

Следующая диаграмма иллюстрирует компоненты и организацию элементов метамодели StarUML™. Некоторые компоненты опущены из-за пространственных ограничений.

Пожалуйста обращайтесь к пакету "::Application Model::Non_Modeling Elements::Metamodeling Elements" модели приложения StarUML для получения полной диаграммы.



Метамодель StarUML™ включает относительно небольшое количество метаэлементов, как показано на диаграмме. IMetaElement - элемент верхнего уровня элементов метамодели и имеет атрибуты Name и GUID. Так как элементы моделирования - инстанции элемента метамодели (метаэлемента IMetaClass), значение свойства Name каждого объекта типа IMetaElement должно быть одним из названий (имен) элементов моделирования, описанных в "Главе 5. Работа с модельными элементами". Примеры - "Model", "View", "UMLClass" и "UMLAttribute".

Потомками IMetaElement являются элементы метамодели типа IMetaClassifier и IMetaSlot. IMetaClassifier - это метаэлемент для непосредственного определения элементов моделирования, а IMetaSlot - для определения атрибутов элемента моделирования и атрибутов ссылки. Также, конкретные элементы подобные IMetaClass, IMetaAttribute, IMetaReference, IMetaCollection и IMetaModel наследованы от IMetaClassifier и IMetaSlot; они играют самую важную роль в архитектуре метамодели StarUML™.

Обработка элементов метамодели

IMetaModel

Элемент IMetaModel поддерживает и управляет элементами метамодели как коллекциями и предоставляет использование других элементов метамодели. Только один объект IMetaModel

существует в одном приложении StarUML™. Ссылка на этот объект может быть получена через интерфейс IStarUMLApplication.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var meta = app.MetaModel;
```

Ранее было упомянуто, что IMetaModel предоставляет для использования доступ к другим элементам метамодели. Следующий пример получает ссылку на элементы IMetaClass, используя IMetaModel. В разделе, посвящённом IMetaClass, будет объяснено, что число объектов типа IMetaClass и число элементов моделирования всегда совпадают (сверьтесь со следующим примером).

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var meta = app.MetaModel;
for (var i = 0; i < meta.GetMetaClassCount(); i++){
var metaClass = meta.GetMetaClassAt(i);
...
}
```

Хотя в диаграмме классов выше опущены типы IMetaEnumeration, IMetaDataType, и IMetaPrimitiveType и их отношения с IMetaClass, интерфейс IMetaModel предоставляет ссылку и на эти элементы. IMetaEnumeration - метаэлемент для определения типа перечисления, связанный с элементами моделирования. UMLVisibilityKind и UMLAggregationKind - примеры инстанций элемента IMetaEnumeration.

IMetaDataType - метаэлемент для определения других типов данных (кроме перечислимых) и примитивных типов. Тип Points -его единственная инстанция. IMetaPrimitiveType - метаэлемент для того, чтобы определять примитивные типы, подобные Integer, Real, Boolean и String.

Интерфейс IMetaModel предоставляет метод поиска метаэлементов. Следующий пример - повторение первого примера в этой главе. Он показывает получение ссылки на элемент IMetaClass для элемента моделирования "UMLClass" с использовнием метода IMetaModel.FindMetaClass (число объектов типа IMetaClass - равно числу элементов моделирования).

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var meta = app.MetaModel;
var metaClass = meta.FindMetaClass("UMLClass");
...
```

Подобно IMetaClass, интерфейс IMetaModel предоставляет методы поиска других метаэлементов: FindMetaClassifier, FindMetaEnumeration, FindMetaDataType и FindPrimitiveType.

Интерфейс IMetaModel - имеет свойство GUID и предоставляет метод FindInstanceByGuid, который возвращает ссылку на соответствующий элемент моделирования. Метод FindInstanceByGuid возвращает ссылку типа IElement. Следующий код может использоваться как расширение примера выше.

```
...
var guid = ...
var elem = meta.FindInstanceByGuid(guid);
...
```

IMetaClass

IMetaClass - метаэлемент, который предоставляет определение для каждого элемента моделирования, и поддерживает и управляет инстанциями каждого элемента моделирования как коллекцией. В приложении StarUML™, число элементов IMetaClass равно числу элементов

моделирования. Следующий код показывает получение ссылок на IMetaClass каждого элемента моделирования, используя метод IMetaModel.FindMetaClass.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var meta = app.MetaModel;
var metaClassOfPackage = meta.FindMetaClass("UMLPackage");
var metaClassOfClass = meta.FindMetaClass("UMLClass");
var metaClassOfAttribute = meta.FindMetaClass("UMLAttribute");
...
```

Другой способ получить ссылку на элементы типа IMetaClass состоит в том, чтобы использовать метод GetMetaClass интерфейса IElement, который является типом верхнего уровня элемента моделирования.

```
elem = ... // получить ссылку на модельный элемент
var metaClass = elem.GetMetaClass();
```

Интерфейс IMetaClass предоставляет методы получения суперклассов и подклассов в структуре наследования каждого элемента моделирования. Ссылка на суперкласс для типа IElement - который является элементом моделирования самого верхнего уровня - является пустой.

```
var metaClass = ... // Получаем ссылку на IMetaClass
var superCls = metaClass.Superclass;
...
for (var i = 0; i < metaClass.GetSubclassCount(); i++){
var subCls = metaClass.GetSubclassAt(i);
...
}
```

Интерфейс IMetaClass - содержит GUID для элементов моделирования и предоставляет метод FindInstanceByGuid, который является аналогом IMetaModel.FindInstanceByGuid, чтобы получить ссылку на соответствующий элемент моделирования. Метод типа IMetaClass более эффективен чем метод типа IMetaModel, так как ищет элементы моделирования только определенного типа. Если объект не найден в соответствующем типе, может использоваться метод FindInstanceByGuidRecurse, чтобы искать также среди всех производных элементов моделирования.

Первый пример в этом разделе иллюстрирует поиск инстанции определенного элемента моделирования, используя методы GetInstanceCount и GetInstanceAt интерфейса IMetaClass. Инстанции элементов моделирования соответствуют созданным пользователем элементам моделирования.

```
var metaClass = ... // Получаем ссылку на метакласс.
for (var i = 0; i < metaClass.GetInstanceCount(); i++){
var AElem = metaClass.GetInstanceAt(i);
...
}
```

IMetaAttribute

Интерфейс IMetaAttribute может использоваться, чтобы читать спецификации свойств каждого элемента моделирования. Ссылка на IMetaAttribute может быть получена через интерфейс IMetaClass как показано ниже. Интерфейс IMetaClass также предоставляет метод ExistsAttribute, который проверяет существование свойства с указанным именем, а также метод GetAttributeByName, который возвращает элемент IMetaAttribute с указанным именем.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var metaClass = app.MetaModel.FindMetaClass("UMLClass");
for (var i = 0; i < metaClass.GetMetaAttributeCount(); i++){
```

```
var metaAttr = metaClass.GetMetaAttributeAt(i);
...
}
```

Следующий пример показывает чтение спецификации атрибута.

```
var metaAttr = ... // Получаем ссылку на IMetaAttribute
var metaType = metaAttr.TypeRef;
var attrName = metaAttr.Name;
var attrType = metaType.Name;
...
```

Кроме того, IMetaSlot - интерфейс верхнего уровня для IMetaAttribute - предоставляет свойства IsReadOnly и IsDerived. IsReadOnly указывает, имеет ли атрибут статус "только для чтения", а IsDerived указывает, существует ли соответствующий атрибут фактически, или получен через другие атрибуты.

IMetaReference и IMetaCollection

Элементы IMetaReference и IMetaCollection определяют элементы ссылок, которые специфицируют ссылки между различными элементами моделирования. Все эти ссылки - специальные ассоциации. В то время как IMetaReference показывает ссылки мощностью '1' или меньше, IMetaCollection показывает ссылки, которые выражаются в виде коллекций. Это - единственное различие между интерфейсами IMetaReference и IMetaCollection (IMetaReference и IMetaCollection оба получены из интерфейса IMetaAssociationEnd).

Сначала, позвольте нам рассмотреть на примере получение ссылок на объекты IMetaReference и IMetaCollection. Для этого используется интерфейс IMetaClass.

```
var metaClass = ... // Получаем тип IMetaClass под указанный модельный элемент
// Получаем ссылки в объекте типа IMetaReference
for (var i = 0; i < metaClass.GetMetaReferenceCount(); i++){
var metaAttr = metaClass.GetMetaReferenceAt(i);
...
}
// Получаем ссылки на объекты типа IMetaCollection
for (var i = 0; i < metaClass.GetMetaCollectionCount(); i++){
var metaAttr = metaClass.GetMetaCollectionAt(i);
...
}
```

Интерфейс IMetaAssociationEnd, общий предок для IMetaReference и IMetaCollection, предоставляет свойства и методы для того, чтобы определять спецификации ссылок (ассоциаций) для соответствующего элемента моделирования.

Свойство Kind определяет, является ли соответствующая ассоциация простой ссылкой или коллекцией. Тип IMetaReference - простой тип ссылки, и тип IMetaCollection - тип ссылки коллекции. Свойство Aggregate показывает значение атрибута AggregationKind соответствующей ассоциации. Это свойство перечислимого типа и может принимать одно из следующих значений:

- makNone (0): Нет
- makAggregate (1): Агрегация
- makComposite (2): Композиция

Свойство OtherSide показывает имя AssociationEnd другой стороны ассоциации, а свойство OtherSideKind показывает, является ли AssociationEnd другой стороны простой ссылкой или коллекцией.

Метод GetOtherSideEnd возвращает ссылку типа IMetaAssociationEnd на AssociationEnd с другой стороны ассоциации. Следующий пример показывает, как использовать свойства и методы, предоставленные интерфейсом IMetaAssociationEnd, который является общим предком для IMetaReference и IMetaCollection.

```
var metaSlot = ... // Get IMetaReference or IMetaCollection type reference.
var kind = metaSlot.Kind;
var aggregate = metaSlot.Aggregate;
var otherSide = metaSlot.OtherSide;
var otherSideKind = metaSlot.OtherSideKind;
var otherSideEnd = metaSlot.GetOtherSideEnd();
...
```

Атрибут TypeRef является ссылкой на интерфейс IMetaSlot и может использоваться, чтобы узнать IMetaClass для объекта типа IMetaReference или IMetaCollection. Следующий пример показывает, как читать элемент на противоположной стороне ассоциации элемента моделирования.

```
var metaSlot = ... // Получаем ссылки на IMetaReference или IMetaCollection.
var otherSideEnd = metaSlot.GetOtherSideEnd();
var otherSideMetaClass = otherSideEnd.TypeRef;
...
```

Глава 5. Написание подходов

Основная концепция подхода

Существуют многочисленные методологии разработки программного обеспечения, и каждая компания или организация применяет собственную или существующую методологию, которая адаптирована под требования её группы разработчиков или проектировщиков. Предметные области, языки программирования и технические платформы также различны для каждой области разработки программного обеспечения. Следовательно, должно быть согласовано много аспектов на начальной стадии программного моделирования. Технология подходов призвана облегчить начальное согласование параметров разработки проекта, зависящих от методологии разработки программного обеспечения или требований платформы. Пользователи могут создавать собственные подходы, чтобы инициировать проекты в некоторых начальных формах.

Подходы выполняют следующие задачи при создании проектов.

- Подходы конфигурируют профили, используемые в проектах. Профили, определенные в подходах автоматически включаются в проект в момент создания проекта.
- Подходы определяют структуру пакетов. Структура пакетов обычно зависит от моделей процесса разработки программного обеспечения. Например, использование подхода "4+1 View Model Approach" предполагает наличие пяти пакетов "Logical View", "Physical View", "Process View", "Development View" и "Use Case View".
- Подходы конфигурируют фреймворки, на которые ссылаются. Для проектов, которые являются, зависимыми от определенных языков программирования или платформ, соответствующие фреймворки могут быть определены в подходах и будут загружены при создании проектов. Например, если текущий проект разрабатывается в Java, в подходе может быть определен фреймворк под JFC (Java Foundation Classes), так чтобы он был включен в проект как пакет для прямой ссылки.
- Подходы импортируют модельные фрагменты, чтобы подключить их к основной модели.

Выполняйте шаги, указанные ниже, чтобы создать новый подход.

1. Создайте файл описания подхода (.apr), чтобы определить новый подход.
2. Скопируйте файл описания подхода (.apr) в подкаталог каталога модулей.

Создание нового подхода

Общая структура файла описания подхода

Файл описания подхода создаётся согласно правилам XML и имеет расширение - .apr (approach file). Спецификация подхода помещается в пределах тега APPROACH и не должна содержать никаких ошибок в синтаксисе или структуре.

```
<?xml version=”1.0” encoding=”...” ?>
<APPROACH version="...">
    <HEADER>
        ...
    </HEADER>
    <BODY>
        ...
    </BODY>
```

```
</APPROACH>
```

- Свойство encoding: Определяет значение свойства кодировки для XML документа (например. UTF-8, EUC-KR). Для получения подробностей относительно этого свойства, см. документацию по ресурсам XML.
- Свойство version (атрибут APPROACH): информация о версии документа описания подхода (например 1.0).
- Элемент HEADER: См. раздел "Элемент Header".
- Элемент BODY: См. раздел "Элемент Contents".

Элемент Header

Секция HEADER документа описания подхода содержит общую информацию о подходе типа имени подхода или описания.

```
<HEADER>
    <NAME>...</NAME>
    <DISPLAYNAME>...</DISPLAYNAME>
    <DESCRIPTION>...</DESCRIPTION>
</HEADER>
```

- Элемент NAME: Содержит имя подхода. Это уникальное имя, идентифицирующее подход.
- Элемент DISPLAYNAME : Содержит название подхода, которое показывается пользователям в диалоге New Project.
- Элемент DESCRIPTION: Содержит детальное описание подхода.

Элемент BODY

Секция BODY описания подхода включает элементы IMPORTPROFILES и MODELSTRUCTURE. Элемент IMPORTPROFILES задаёт имена профилей, загружаемых при создании проекта, а элемент MODELSTRUCTURE содержит информацию о начальной структуре модели и загружаемых фреймворках.

```
<BODY>
    <IMPORTPROFILES>
        <PROFILE>...</PROFILE>
        ...
    </IMPORTPROFILES>
    <MODELSTRUCTURE>
        ...
    </MODELSTRUCTURE>
</BODY>
```

- Элемент IMPORTPROFILES: Перечисляет профили, включаемые в проект, используя последовательность элементов PROFILE.
- Элемент PROFILE: Содержит имя профиля, включаемого в проект.
- Элемент MODELSTRUCTURE: См. раздел "Структура модели".

Структура модели

Элемент MODELSTRUCTURE описывает начальную структуру пакетов проекта. Модель, Подсистема, Пакет и Фреймворки обычно образуют некоторую иерархию. Например, элементы модели, подсистемы, пакета или фреймворка могут быть помещены под элементом SUBSYSTEM. В то время как фреймворк сам является пакетом, он не может содержать другие пакеты.

Ниже показан синтаксис описания структур в элементе MODELSTRUCTURE.

```
<MODELSTRUCTURE>
    model_expression*
</MODELSTRUCTURE>

model_expression ::= model_element
    | package_element
    | subsystem_element
    | import_framework
    | import_model_fragment.

model_element ::= <MODEL name=”...” stereotypeProfile=”...”
stereotypeName=”...”>model_expression</MODEL>.

package_element ::= <PACKAGE name=”...” stereotypeProfile=”...”
stereotypeName=”...”>model_expression</PACKAGE>.

subsystem_element ::= <SUBSYSTEM name=”...” stereotypeProfile=”...”
stereotypeName=”...”>model_expression</SUBSYSTEM>.

import_framework ::= <IMPORTFRAMEWORK name=”...”/>.

import_model_fragment ::= <IMPORTMODELFRAGMENT fileName=”...”/>.
```

- Свойство name (элементов MODEL, PACKAGE, SUBSYSTEM): имя каждого модельного элемента UML.
- Свойство stereotypeProfile (элементов MODEL, PACKAGE, SUBSYSTEM): Имя профля, который оределяет стереотип применяемый к модельному элементу.
- Свойство stereotypeName (элементов MODEL, PACKAGE, SUBSYSTEM): Имя стереотипа, который применяется к модельному элементу.
- Свойство name (элементов IMPORTFRAMEWORK): Имя поключаемого зарегистрированного фреймворка.
- Свойство fileName (элементов IMPORTMODELFRAGMENT): Имя файла модельногофрагмента импортруемого в родтельский модельный элемент.

Пример описания подхода

Ниже приведён пример описания подхода 4+1 View Model.

```
<?xml version=”1.0” encoding=”UTF-8” ?>
<APPROACH version="1">
    <HEADER>
        <TITLE>4+1 View Model</TITLE>
        <DESCRIPTION>This is an approach to support 4+1 View Model in .NET
platform.</DESCRIPTION>
    </HEADER>
    <BODY>
        <IMPORTPROFILES>
            <PROFILE>4+1Profile</PROFILE>
            <PROFILE>CSharpProfile</PROFILE>
        </IMPORTPROFILES>
        <MODELSTRUCTURE>
            <MODEL name=”UseCase View”/>
            <MODEL name=”Logical View”>
                <IMPORTFRAMEWORK name=”dot_net_framework”/>
           </MODEL>
            <MODEL name=”Development View”/>
            <MODEL name=”Process View”/>
            <MODEL name=”Deployment View”/>
        </MODELSTRUCTURE>
    </BODY>
</APPROACH>
```

Регистрация нового подхода

Чтобы подход был автоматически распознан програмой StarUML, его нужно поместить в отдельный подкаталог каталога модулей StarUML (<install-dir> \modules). StarUML открывает и читает все подходы, находящиеся в каталоге модулей, и автоматически регистрирует их при своей инициализации. Если файл подхода не открывается или его расширение - не .apr, StarUML не будет читать подход и проигнорирует его. Рекомендуется, помещать каждый подход в отдельный подкаталог каталога модулей, во избежание их беспорядочной загрузки.

Обратите внимание: Чтобы зарегистрировать иконку подхода, создайте файл с иконкой и поместите его в каталог подхода. Иконка подхода будет отображена рядом с названием подхода в списке подходов в диалоге New Project. Если файла иконки с тем же именем, что и у подхода, не обнаруживается, подходу присваивается значок, принятый по умолчанию.

Обратите внимание: Удалите файлы подхода из каталога модулей StarUML (<install-dir> \modules), чтобы подход больше не использовался.

Использование методов, связанных с подходом

Чтение информации о подходах, установленных в системе

Так как подходы предназначены только для инициализации проектных конфигураций, к ним обычно нет необходимости обращаться непосредственно из программ. Поэтому, StarUML™ не предоставляет объекты COM-automation для того, чтобы работать с подходами. Однако, методы GetAvailableApproachCount () и GetAvailableApporachAt () интерфейса IProjectManager могут использоваться, чтобы получить количество и имена подходов, установленных в системе.

```
IProjectManager.GetAvailableApproachAt(Index: Integer): String
IProjectManager.GetAvailableApproachCount(): Integer
```

Создание проекта с подходом

Новый проект с заданным подходом может быть создан с помощью метода IProjectManager. NewProjectByApproach (). ApproachName, указанный как параметр, должен являться именем одного из подходов, установленных в системе. Иначе, будет открыт пустой проект. Сигнатура метода NewProjectByApproach () следующая.

IProjectManager.NewProjectByApproach(ApproachName: String)

Следующий пример на Jscript создаёт новый проект с подходом "UMLComponents".

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var prjMgr = app.ProjectManager;
prjMgr.NewProjectByApproach("UMLComponents");
```

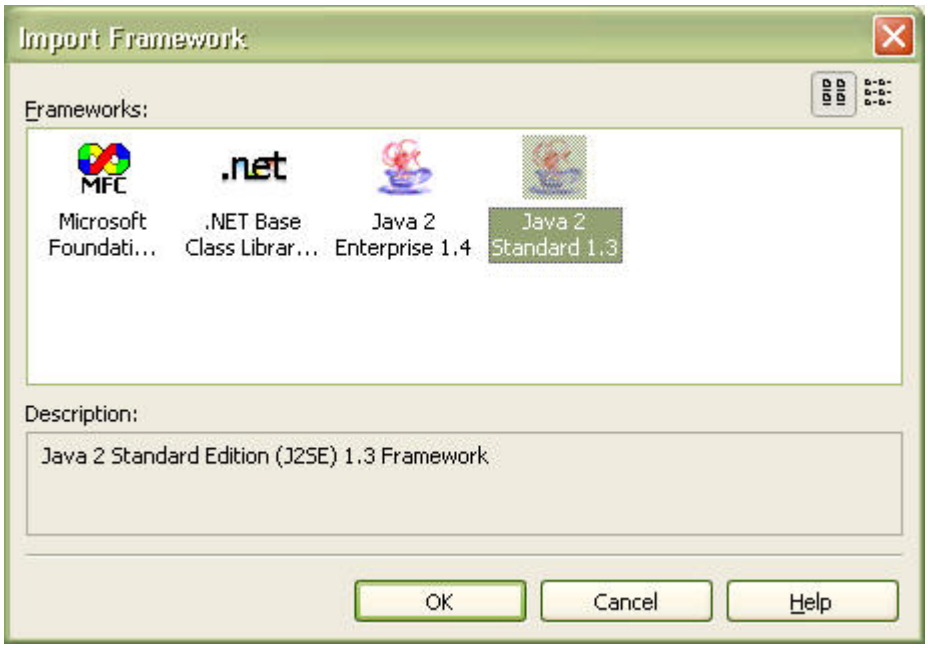
Глава 6. Написание фреймворков

Основные концепции модельного фреймворка

Концепция модельного фреймворка предполагает возможность использования прикладных инструментариев или библиотек классов в StarUML™. Например, JFC (Java Foundation Classes), MFC (Microsoft Foundation classes) и VCL (Visual Component Library) могут быть объектами моделирования при создании фреймворков.

Как будет показано позже в этой главе, пользователь может также создать свой собственный фреймворк. Самое большое преимущество использования фреймворков состоит в том, что они обеспечивают общедоступность и возможность многократного использования основных элементов моделирования и модельных структур.

Диалог "Import Model Framework" (показанный ниже), вызываемый меню [File]-[Import]-[Framework…], отображает список фреймворков, установленных в системе. Выбор элемента из этого списка и его выполнение приводит к автоматическому включению модельной структуры, определенной фреймворком, в указанное место модели. Фреймворк состоит из многих секций и обрабатывается программой StarUML тем же самым способом, что и секция.



Список фреймворков, установленных в системе, можно получить, равно как и подгрузить в проект любые специфические фреймворки, используя API StarUML. Подробно эти операции будут обсуждены позже.

Создание нового фреймворка

Фреймворк состоит из многих файлов секций (.unt) и одного файла определения фреймворка

(.frw), он также может иметь дополнительный файл значка (.ico).

Выполните шаги, указанные ниже, чтобы определить новый фреймворк.

1. Создайте файлы секций, которые содержат модельную информацию для фреймворка (см. " Главу 4. Использование открытого API").
2. Создайте файл описания модельного фреймворка (.frw), который специфицирует модельный фреймворк.
3. Скопируйте файлы секций, файл описания и файл значка модельного фреймворка в подкаталог каталога модулей.

Общая структура файла описания модельного фреймворка

Файл описания модельного фреймворка создаётся в соответствии с правилами разработки XML-документов и должен иметь расширение .frw (Framework File). Информация о модельном фреймворке содержится в пределах тега FRAMEWORK, она не должна содержать никаких ошибок в синтаксисе или структуре.

```
<?xml version="1.0" encoding="..." ?>
<FRAMEWORK version="...">
    <HEADER>
        ...
    </HEADER>
    <BODY>
        ...
    </BODY>
</FRAMEWORK>
```

- Свойство encoding: Определяет тип кодировки документа XML (например. UTF-8, EUC-KR). Для получения подробной информации об этом свойстве, см. описание ресурсов XML.
- Свойство version (элемент FRAMEWORK): Информация о версии формата документа фреймворка (напр. 1.0).
- Элемент HEADER : См. раздел "Элемент Header".
- Элемент BODY: См. раздел "Элемент Body".

Элемент Header

Секция HEADER содержит общую информацию о модельном фреймворке типа имени модельного фреймворка и его описания.

```
<HEADER>
    <NAME>...</NAME>
    <DISPLAYNAME>...</DISPLAYNAME>
    <DESCRIPTION>...</DESCRIPTION>
</HEADER>
```

- Элемент NAME: Содержит имя модельного фреймворка. Оно является уникальным идентификатором модельного фреймворка и должно быть идентично имени соответствующего ключа системного реестра.
- Элемент DISPLAYNAME: Содержит отображаемое название фреймворка, используемое в диалоге "Import Model Framework" и т.д.
- Элемент DESCRIPTION: Содержит описание модельного фреймворка.

Элемент Body

Секция BODY содержит актуальную информацию о модельном фреймворке и состоит главным образом из секций IMPORTPROFILES и FRAMEWORKMODELS.

```
<BODY>
    <IMPORTPROFILES>
        <PROFILE>...</PROFILE>
        ...
    </IMPORTPROFILES>
    <FRAMEWORKMODELS>
        <UNIT>...</UNIT>
        ...
    </FRAMEWORKMODELS>
</BODY>
```

- Элемент IMPORTPROFILES: Перечисляет профили UML, загружаемые при подключении модельного фреймворка.
- Элемент PROFILE: Указывает имя каждого загружаемого профиля UML.
- Элемент FRAMEWORKMODELS: Перечисляет файлы секций, которые составляют модельный фреймворк.
- Элемент UNIT: Определяет имя каждого файла секции. Указываются только имена файлов, без пути. Файлы секций, которые составляют модельный фреймворк, должны быть расположены по тому же маршруту, что и файл документа модельного фреймворка.

Обратите внимание: Элемент "UNIT" указывает только те файлы секций, которые являются секциями верхнего уровня. Как сообщалось в "Главе 4. Использование открытого API", когда секция содержит подсекции более низкого уровня, все они автоматически загружаются вместе с корневой секцией.

Пример документа модельного фреймворка

Следующий пример документа модельного фреймворка описывает фреймворк для Java 2 Standard Edition (J2SE) 1.3.

```
<?xml version="1.0" encoding="UTF-8" ?>
<FRAMEWORK version="1.0">
    <HEADER>
        <NAME>J2SE1.3</NAME>
        <DISPLAYNAME>Java 2 Standard 1.3</DISPLAYNAME>
        <DESCRIPTION>Java 2 Standard Edition (J2SE) 1.3 Framework.</DESCRIPTION>
    </HEADER>
    <BODY>
        <FRAMEWORKMODELS>
            <UNIT>J2SE13 (java).pux</UNIT>
            <UNIT>J2SE13 (javax).pux</UNIT>
            <UNIT>J2SE13 (org).pux</UNIT>
        </FRAMEWORKMODELS>
    </BODY>
</FRAMEWORK>
```

Регистрация нового модельного фреймворка

Чтобы фреймворк был автоматически распознан программой StarUML, он должен размещаться в подкаталоге каталога модулей StarUML (<install-dir>\modules). StarUML находит и читает все фреймворки, расположенные в каталоге модулей, и автоматически регистрирует их при своей инициализации. Если файл фреймворка не корректен или имеет расширение не .frw, StarUML игнорирует его. Рекомендуется размещать фреймворк в отдельном подкаталоге каталога модулей

StarUML, во избежание их беспорядочного хранения.

Обратите внимание: Чтобы зарегистрировать иконку для фреймворка, создайте её и поместите в каталог фреймворка. Иконка фреймворка будет отображена рядом с его названием в списке фреймворков в диалоге [Import Framework]. Если иконка с именем, совпадающим с именем фреймворка, не обнаруживается, используется иконка, заданная по умолчанию.

Обратите внимание: Просто удалите файлы фреймворка из каталога модулей StarUML (<install-dir> \modules), чтобы фреймворк больше не использовался .

Использование методов обработки фреймворков

Чтение информации о модельных фреймворках, установленных в системе

Список модельных фреймворков, установленных в системе, может быть получен через внешнее API с помощью методов GetAvailableFrameworkCount и GetAvailableFrameworkAt интерфейса IProjectManager.

Ниже приведены сигнатуры этих методов.

```
IProjectManager.GetAvailableFrameworkAt(Index: Integer): String
IProjectManager.GetAvailableFrameworkCount(): Integer
```

Импортирование модельного фреймворка

Метод IProjectManager.ImportFramework может использоваться, чтобы включить зарегистрированный модельный фреймворк в текущий проект. Сигнатура метода приведена ниже. Параметр OwnerPackage определяет модельный элемент верхнего уровня, в который будет включен модельный фреймворк. Он должен быть модельным элементом типа IUMLPackage. Параметр FrameworkName задаёт имя загружаемого модельного фреймворка. Это строковое значение должно содержать точное имя (идентификатор) модельного фреймворка.

```
IProjectManager.ImportFramework(OwnerPackage: IUMLPackage; FrameworkName: String)
```

Следующий пример показывает импорт модельный фреймворка "J2SE1.3" с использованием метода IProjectManager.ImportFramework.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var prjMgr = app.ProjectManager;
var owner = ... // Get reference to IUMLPackage type element.
prjMgr.ImportFramework(owner, "J2SE1.3");
```

Глава 7. Написание профилей UML

Основные концепции профилей UML

Механизмы расширения UML

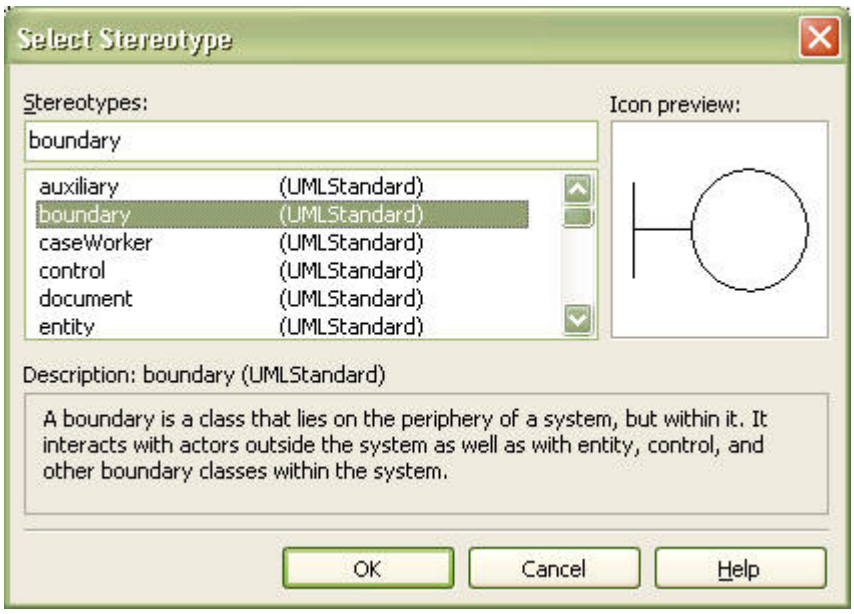
UML - универсальный язык моделирования, который предоставляет множество четких концепций и графических нотаций, удовлетворяющих общим требованиям моделирования программного обеспечения. Однако, средства моделирования/разработки программного обеспечения сегодня могут принимать множество различных форм и требовать наличия элементов или семантики, которая не существует в стандарте UML. UML предоставляет стандартную концепцию поддержки таких требований, которую называют Механизмами расширения UML.

Механизм расширения UML использует стереотипы, ограничения, определения и значения тегов, чтобы расширять семантику стандартных элементов моделирования UML или определять элементы моделирования с новой семантикой.

Стереотип

Стереотип - элемент моделирования, который предусмотрен для того, чтобы добавлять новые свойства и ограничения к стандартному элементу моделирования UML. Стереотип может также использоваться, чтобы предоставить новые графические нотации для элементов моделирования. Ниже показан диалог выбора стереотипа, который появляется при нажатии на кнопку выбора стереотипа в приложении StarUML.

Диалог выбора стереотипа показывает список доступных стереотипов, определенных в профиле UML, который включен в текущий проект. Стереотипы могут также быть сконфигурированы или изменены посредством API. Детально эти операции будут описаны позже.

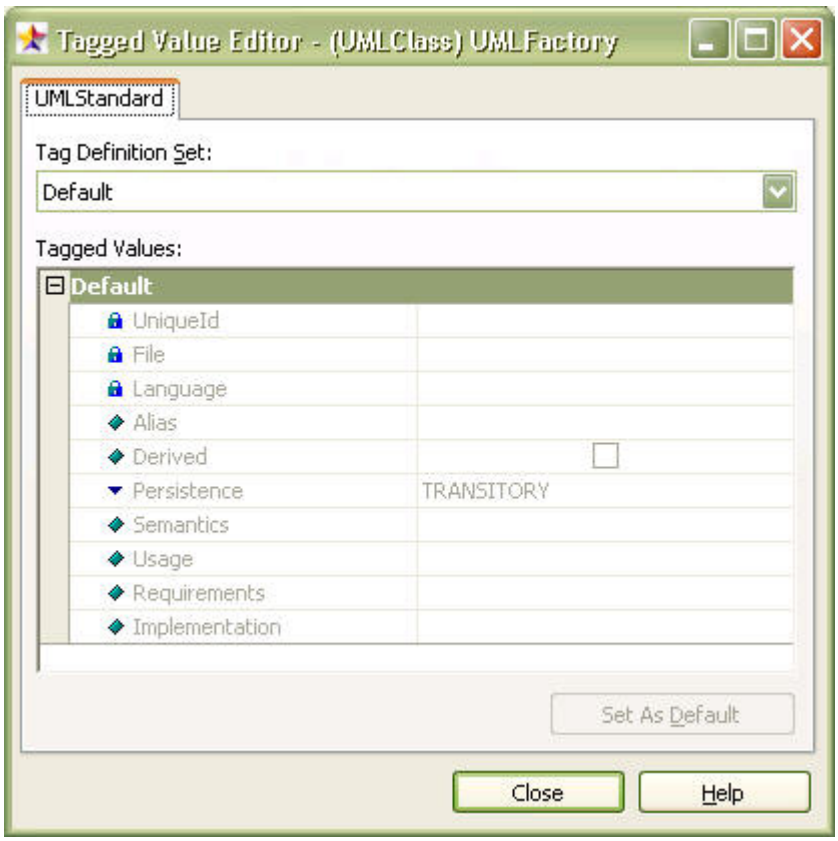


Обратите внимание: Хотя стандарт UML позволяет каждому расширяемому элементу моделирования иметь несколько стереотипов, StarUML™ ограничивает каждый элемент моделирования только одним стереотипом.

Определения тегов

Определение тега - элемент, который специфицирует новое свойство, которое можно добавить некоторым элементам моделирования. Определение тега включает определение возможных значений тега. Значение тега может быть значением скалярного типа данных, ссылкой на другой элемент моделирования или коллекцией.

Следующая иллюстрация показывает экран редактора дополнительных свойств в приложении StarUML™. Редактор свойств показывает список тегов, которые принадлежат выбранному элементу моделирования в соответствии с профилем UML. Значения тегов элемента моделирования также могут быть сконфигурированы или изменены через внешнее API. Детально эти возможности будут описаны позже.



Констрэйнты

Констрэйнты добавляют специфические ограничения к некоторому элементу моделирования, позволяющие переопределить его семантику. Для ознакомления с описаниями ограничений см. радел "Элементы ExtCore" в "Главе 4. Использование открытого API".

Обратите внимание: UML профиль в StarUML не содержит определений ограничений.

Профиль UML

Профиль UML - пакет механизмов расширения UML. Другими словами, это - коллекция стереотипов, ограничений, определений тегов и типов данных, которые требуются для некоторой области программного обеспечения или платформы разработки.

Профиль UML состоит из элементов стереотипа, ограничения, определения тега и типа данных. Хотя стандарт UML требует, чтобы профиль специфицировался как пакет со стереотипом "<<profile>>", StarUML™ реализует его через XML-файл, для более легкого использования.

Дополнительный механизм расширения в StarUML

Профиль StarUML поддерживает несколько дополнительных механизмов расширения наряду с определенными в UML. Это - тип диаграммы, прототип элемента, прототип модельного элемента, расширение палитры. Эти механизмы расширения дополняют семантику стандартных элементов и предоставляют регулярные методы создания расширенных элементов и их интеграции в пользовательский интерфейс.

Тип диаграммы

Тип диаграммы - элемент расширения, предназначенный для определения нового типа диаграмм, реализующих дополнительную семантику на базе стандартных диаграмм UML. Полезно создавать специализированные диаграммы на каждой стадии проекта, например диаграмму модели данных, диаграмму анализа устойчивости и так далее, а также сделать несколько видов диаграмм, применяемых в различных областях, доступными для использования на StarUML. Имя типа диаграммы присваивается свойству "DiagramType" диаграммы. Свойство DiagramType не может быть изменено в отличие от стереотипа. Когда профиль включается в проект, он расширяет меню [Add Diagram], что позволяет создавать диаграммы дополнительных типов.

Прототип элемента

Прототип элемента определяет некоторый образец создания элемента, в котором заранее заданы некоторые свойства. Пользователь может создать элемент, копируя образец, зарегистрированный как прототип элемента в палитре, или используя внешнее API.

Прототип модельного элемента

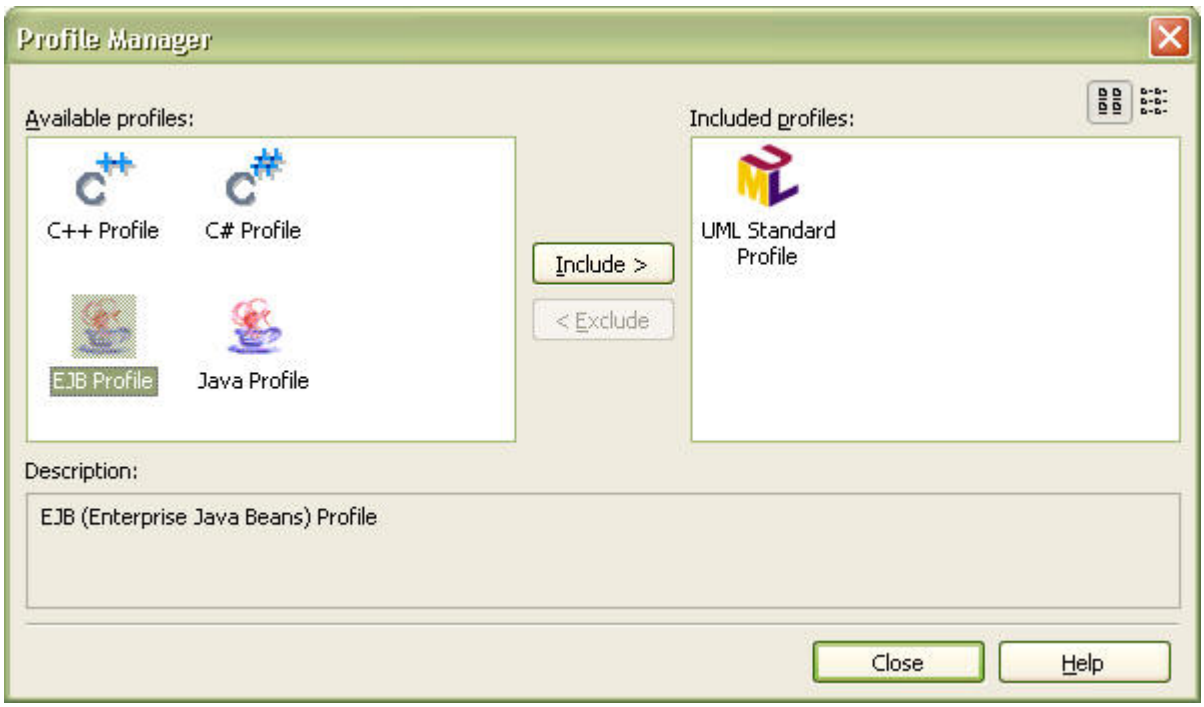
Прототип модельного элемента может быть применен только к модельному элементу, даже если он аналогичен прототипу элемента моделирования. Модельный элемент, который является копией данного прототипа, может быть создан только с помощью меню. Он добавляется в подменю [Add Model] подобно прототипу элемента в палитре.

Расширение палитры

Расширение палитры позволяет вставлять дополнительную палитру, которая появляется слева на главной форме. Добавленная палитра может содержать прототипы элементов или элементы UML, определенные в профиле как элементы палитры.

Включение и отключение профиля

Если профиль UML требуется для текущего проекта, его нужно добавить в проект, потому что никакие профили кроме "UML Standard Profile" не добавляются в проект автоматически. Чтобы добавить профиль, используйте диалог профиля (показан ниже), который может быть вызван через меню [Model] -> [Profiles…]. Список "Available Profiles" слева показывает профили, зарегистрированные в настоящее время в пользовательской системе, а список "Included Profiles" справа показывает профили, уже включенные в текущий проект. Добавление профиля может быть сделано простым выбором профиля в списке доступных профилей и нажатием кнопки "Include" в центре. Как только профиль добавится в проект, стереотипы и определения тегов, заданные в новом профиле, добавятся в диалог выбора стереотипов и редактор свойств расширения, которые были показаны выше. Если профиль больше не требуется в текущем проекте, просто щелкните кнопку "Exclude", чтобы удалить его из проекта. Но соблюдайте осторожность, так как исключение профиля повлечёт удаление всей информации проекта, относящейся к исключённому профилю. Профили могут также быть включены или исключены через внешнее API. Детально эти операции будут описаны позже.



Создание профиля UML

Общая структура файла документа профиля

Файл документа профиля создаётся в формате XML и имеет расширение - .prf (PLASTIC Profile File). Содержимое профиля располагается внутри тега PROFILE. Не должно быть никаких ошибок в синтаксисе или структуре этого файла.

Общая структура документа профиля следующая.

```
<?xml version="1.0" encoding="..." ?>
<PROFILE version="...">
    <HEADER>
        ...
    </HEADER>
    <BODY>
        ...
    </BODY>
</PROFILE>
```

- Свойство encoding: Определяет тип кодировки документа XML (например. UTF-8, EUC-KR). Для получения информации об этом параметре см. документацию по XML.
- Свойство version (элемент PROFILE): Версия PRF документа (например 1.0).
- Элемент HEADER: См раздел Элемент HEADER.
- Элемент BODY: См раздел Элемент BODY.

Элемент HEADER

Секция HEADER документа профиля содержит общую информацию о профиле, типа имени профиля и описания.

```
<HEADER>
    <NAME>...</NAME>
    <DISPLAYNAME>...</DISPLAYNAME>
```

```
    <DESCRIPTION>...</DESCRIPTION>
    <AUTOINCLUDE>...</AUTOINCLUDE>
</HEADER>
```

- Элемент NAME: Содержит имя профиля. Имя идентифицирует профиль.
- Элемент DISPLAYNAME : Название, используемое в диалоге профилей и других пользовательских интерфейсах.
- Элемент DESCRIPTION: Содержит описание профиля.
- Элемент AUTOINCLUDE: Определяет, подключается ли профиль автоматически, когда создается новый проект.

Элемент BODY

Секция BODY документа профиля содержит собственно профиль. Эта информация может включать спецификации стереотипов, типов данных, наборов тегов дополнительных элементов расширения.

```
<BODY>
    <STEREOTYPELIST>
        ...
    </STEREOTYPELIST>
    <TAGDEFINITIONSETLIST>
        ...
    </TAGDEFINITIONSETLIST>
    <DATATYPELIST>
        ...
    </DATATYPELIST>
    <ELEMENTPROTOTYPELIST>
        ...
    </ELEMENTPROTOTYPELIST>
    <MODELPROTOTYPELIST>
        ...
    </MODELPROTOTYPELIST>
    <PALETTELIST>
        ...
    </PALETTELIST>
    <DIAGRAMTYPELIST>
        ...
    </DIAGRAMTYPELIST>
</BODY>
```

- Элемент STEREOTYPELIST: Определяет перечень стереотипов (элементы STEREOTYPE). Об определении стереотипов, см. раздел Стереотипы.
- Элемент TAGDEFINITIONSETLIST: Определяет перечень тегов (элементы TAGDEFINITIONSET). Об определении тегов, см. раздел Определение тегов.
- Элемент DATATYPELIST: Определяет перечень типов данных (элементы DATATYPE). Об определении типов данных, см. раздел Типы данных.
- Элемент ELEMENTPROTOTYPELIST: Определяет прототипы элементов (элементы ELEMENTPROTOTYPE). Об определении прототипа элемента, см. раздел Прототипы элемента.
- Элемент MODELPROTOTYPELIST: Определяет перечень прототипов модельных элементов (элементы MODELPROTOTYPE). Об определении прототипа модельного элемента, см. раздел Прототипы модельных элементов.
- Элемент PALETTELIST: Определяет перечень палитр (элементы PALETTE). Об определении расширений палитры, см. секцию Палитры.
- Элемент DIAGRAMTYPELIST: Определяет перечень типов диаграмм (элементы DIAGRAMTYPE). Об определении типов диаграмм, см. раздел Типы диаграмм.

Стереотип

Элемент STEREOTYPE определяет информацию о стереотипе и структуре наследования.

```
<STEREOTYPE>
    <NAME>...</NAME>
    <DESCRIPTION>...</DESCRIPTION>
    <BASECLASSES>
        <BASECLASS>...</BASECLASS>
        ...
    </BASECLASSES>
    <PARENT>...</PARENT>
    <RELATEDTAGDEFINITIONSET>...</RELATEDTAGDEFINITIONSET>
    <ICON minWidth="..." minHeight="...">...</ICON>
    <NOTATION>...</NOTATION>
</STEREOTYPE>
```

- Элемент NAME: Содержит имя стереотипа. Оно должно быть уникальным в пределах профиля.
- Элемент DESCRIPTION: Содержит описание стереотипа.
- Элемент BASECLASSES: Может содержать имена модельных элементов UML, к которым может быть применен стереотип (например, UMLClass, UMLClassifier, UMLAttribute, UMLPackage...).

Обратите внимание: Если используется имя абстрактного класса, подобное UMLClassifier, все элементы, унаследованные от этого класса, будут совместимы со стереотипом. Если определен стереотип верхнего уровня (элемент PARENT), этот элемент не указывается; а если он указан, то он игнорируется и используется значение BASECLASSES стереотипа верхнего уровня.

- Элемент PARENT: Стереотипы могут иметь отношения наследования. Элемент PARENT содержит имя стереотипа-предка. Отношения наследования должны связывать стереотипы в пределах одного профиля. Этот элемент можно опустить, если стереотип верхнего уровня не определён.
- Элемент RELATEDTAGDEFINITIONSET: Содержит имя набора тегов, связанного со стереотипом. Он может интерпретироваться как набор дополнительных свойств, предоставленных стереотипом элементу. Может быть опущен, если нет ни одного тега. Набор определений тегов, указанный здесь, должен быть специфицировано в этом же самом профиле.
- Элемент ICON: Стереотип может обозначаться иконкой, указанной пользователем. Этот элемент содержит имя файла иконки стереотипа. Иконки могут быть представлены форматах .WMF, .EMF или .BMP. Файлы иконок должны быть расположены в том же самом каталоге, что и документ профиля. Документ профиля содержит имена файлов иконок без путей.
- Свойство minWidth (элемент ICON): Определяет минимальную ширину стереотипной иконки.
- Свойство minHeight (элемент ICON): Определяет минимальную высоту стереотипной иконки.
- Элемент NOTATION: Стереотип может отображаться не только как иконка, он также может рисоваться специальным методом, определённым на языке описания графических нотаций. Этот элемент содержит имя файла расширения нотации (.nxt), определяющего нотацию. Элемент с расширенной нотацией будет прорисован так, как определено в файле расширения нотации, а не как стандартный элемент UML. Файл расширения нотации должен быть помещен в каталог документа профиля. Для него должно быть указано только имя без пути.

Набор определений тегов (TagDefinitionSet)

Элемент TAGDEFINITIONSET содержит основную информацию, касающуюся набора определений тегов. Он включает последовательность элементов TAGDEFINITION, располагающихся внутри элемента TAGDEFINITIONLIST, перечисляющую определения тегов, включенные в набор.

```
<TAGDEFINITIONSET>
    <NAME>...</NAME>
    <BASECLASSES>
        <BASECLASS>...</BASECLASS>
        ...
    </BASECLASSES>
    <TAGDEFINITIONLIST>
        ...
    </TAGDEFINITIONLIST>
</TAGDEFINITIONSET>
```

- Элемент NAME: Содержит имя набора тегов. Если определения тегов относятся к конкретному стереотипу, рекомендуется использовать имя этого стереотипа (в этом случае, набор будет показывается первым в пользовательском интерфейсе).
- Элемент BASECLASSES: Содержит имена элементов UML, к которым применяются определения тегов (используется так же как BASECLASSES элемента STEREOTYPE). Если набор тегов определен для конкретного стереотипа, этот элемент не указывается; любое определение этого элемента игнорируется и используется значение BASECLASSES соответствующего стереотипа.
- Элемент TAGDEFINITIONLIST: Содержит перечень определений тегов, включенных в набор. См. раздел Определения тегов.

Определение тега

Элемент TAGDEFINITION содержит определение тега, включенное в набор.

```
<TAGDEFINITION lock="...">
    <NAME>...</NAME>
    <TAGTYPE referenceType="...">...</TAGTYPE>
    <DEFAULTDATAVALUE>...</DEFAULTDATAVALUE>
    <LITERALS>
        <LITERAL>...</LITERAL>
        ....
    </LITERALS>
</TAGDEFINITION>
```

- Свойство lock (элемент TAGDEFINITION): Определят, можно ли изменять значение тега средствами интерфейса пользователя. Если установлено в "True", значения тега могут быть изменены только через внешний COM-интерфейс, а редактор тегов использоваться не может. Это свойство может быть опущено, поскольку имеется заданное по умолчанию значение "False".
- Элемент NAME: Имя тега. Должно быть уникально в пределах набора.
- Элемент TAGTYPE: Тип тега. Может быть любым из: Integer, Boolean, Real, String, Enumeration, Reference или Collection.
- Свойство referenceType (элемент TAGTYPE): Указывает, какой тип объектной ссылки допускается, когда тип тега - Reference или Collection. Например, определив это свойство как "UMLClass", вы разрешаете ссылку только на Class. Если свойство опущено, используется заданное по умолчанию значение - "UMLModelElement". Это свойство игнорируется, если тип тега - Integer, Boolean, Real, String или Enumeration.
- Элемент DEFAULTVALUE: Содержит значение тега по умолчанию. Этот элемент игнорируется, а заданное по умолчанию значение устанавливается в null для тегов типа

ссылки или коллекции.
- Элемент LITERALS: Определяет допустимый набор литералов, если тип тега - Enumeration. Игнорируется для тегов других типов.

Тип данных

Элемент DATATYPE определяет тип данных. Этот элемент имеет один суб-элемент с именем NAME.

```
<DATATYPE>
    <NAME>...</NAME>
</DATATYPE>
```

- Элемент NAME: Содержит имя типа данных.

Прототип элемента

Элемент ELEMENTPROTOTYPE описывает прототип элемента, который определяет образец для создания модельного элемента.

```
<ELEMENTPROTOTYPE>
    <NAME>....</NAME>
    <DISPLAYNAME>....</DISPLAYNAME>
    <ICON>....</ICON>
    <DRAGTYPE>....</DRAGTYPE>
    <BASEELEMENT argument="...">....</BASEELEMENT>
    <STEREOTYPENAME>....</STEREOTYPENAME>
    <STEREOTYPEDISPLAY>....</STEREOTYPEDISPLAY>
    <SHOWEXTENDEDNOTATION>....</SHOWEXTENDEDNOTATION>
    <MODELPROPERTYLIST>
        <MODELPROPERTY name="...">....</MODELPROPERTY>
        ....
    </MODELPROPERTYLIST>
    <VIEWPROPERTYLIST>
        <VIEWPROPERTY name="...">....</VIEWPROPERTY>
        ....
    </VIEWPROPERTYLIST>
    <TAGGEDVALUELIST>
        <TAGGEDVALUE profile="..." tagDefinitionSet="..." tagDefinition="..."> </TAGGEDVALUE>
        ....
    </TAGGEDVALUELIST>
</ELEMENTPROTOTYPE>
```

- Элемент NAME: Имя прототипа элемента. Оно должно быть уникальным в пределах профиля.
- Элемент DISPLAYNAME: Содержит отображаемое имя, используемое в пользовательском интерфейсе, например в палитре.
- Элемент ICON: Этот элемент содержит имя файла иконки, используемой для прототипа элемента в пользовательском интерфейсе, например в палитре. Файл иконки прототипа элемента должен быть в формате .BMP размером 16 X 16. Файл иконки должен быть помещен в каталог документа профиля. Должно быть определено только имя файла без пути.
- Элемент DRAGTYPE: определяет, как показывать элемент при изменении его местоположения или размера, когда пользователь тянет его мышью на диаграмме. Может принимать одно из значений: Rect или Line.
- Элемент BASEELEMENT: Определяет имя стандартного элемента UML, на базе которого создаётся прототип. Этот элемент не может быть опущен. Если он не определен, прототип будет игнорирован. Допустимые имена стандартных элементов UML следующие.

Имена элементов			
Text	CollaborationInstanceSet	CallEvent	
Note	Interaction	TimeEvent	
NoteLink	InteractionInstanceSet	ChangeEvent	SignalAcceptState
Model	CompositeState	ClassifierRole	SignalSendState
Subsystem	State	Object	Artifact
Package	ActionState	Transition	AttributeLink
Class	Activity	Dependency	Port
Interface	SubactivityState	Association	Part
Enumeration	Pseudostate	AssociationClass	Connector
Signal	FinalState	Generalization	CombinedFragment
Exception	Partition	Link	InteractionOperand
Component	Swimlane	AssociationRole	Frame
ComponentInstance	SubmachineState	Stimulus	ExtensionPoint
Node	Attribute	Message	Rectangle
NodeInstance	Operation	Include	Ellipse
Actor	Parameter	Extend	RoundRect
UseCase	TemplateParameter	Realization	Line
StateMachine	EnumerationLiteral	ObjectFlowState	Image
ActivityGraph	UninterpretedAction	FlowFinalState	
Collaboration	SignalEvent	SystemBoundary	

- свойство argument: Некоторым типам элементов, которые базируются на ассоциации, псевдосостоянии и так далее, требуется аргумент при создании. С помощью этого аргумента устанавливаются некоторые их специфические свойства. Заданное по умолчанию значение этого свойства - 0. В большинстве случаев его не нужно определять.

Значения аргумента, используемые в StarUML, следующие.

Имя элемента	**Смысл и значение**
Pseudostate	Decision = 0, InitialState = 1, Synchronization = 2, Junction Point = 3, Choice Point = 4, Deep History = 5, Shallow History = 6
UninterpretedAction	Entry Action = 0, Do Activity = 1, Exit Action = 2
Stimulus	Stimulus with Call Action = 0, Stimulus with Send Action = 1, Stimulus with Return Action = 2, Stimulus with Create Action = 3, Stimulus with Destroy Action = 4, Reverse Stimulus with Call Action = 5, Reverse Stimulus with Send Action = 6, Reverse Stimulus with Return Action = 7, Reverse Stimulus with Create Action = 8, Reverse Stimulus with Destroy Action = 9
Message	Message with Call Action = 0, Message with Send Action = 1, Message with Return Action = 2, Message with Create Action = 3, Message with Destroy Action = 4, Reverse Message with Call Action = 5, Reverse Message with Send Action = 6,

Имя элемента	Смысл и значение
	Reverse Message with Return Action = 7, Reverse Message with Create Action = 8, Reverse Message with Destroy Action = 9
Association	Assocation = 0, Directed Association = 1, Aggregation = 2, Composition = 3;
Swimlane	Vertical Swimlane = 0, Horizontal Swimlane = 1;

- Элемент STEREOTYPENAME: Определяет имя стереотипа для прототипа элемента. Если значение этого элемента определено, оно устанавливается в качестве значения свойства "Stereotype", когда модельный элемент создаётся. Этот элемент может быть опущен.
- Элемент STEREOTYPEDISPLAY: Определяет, как будет показан стереотип, когда элемент создаётся. Значением этого элемента должно быть sdkText (показывать как текст), sdkIcon (показывать иконку), sdkNone (не показывать), sdkDecoration (показывать декорацию). Этот элемент может быть опущен. Заданное по умолчанию значение - sdkText.
- Элемент SHOWEXTENDEDNOTATION: Определяет, нужно ли использовать для прорисовки файл расширения нотации (.nxt), указанный в элементе STEREOTYPENAME, если таковой существует. Если указано значение True, StarUML прорисовывает элемент согласно файлу расширения нотации. Этот элемент может быть опущен. Заданное по умолчанию значение - False.
- Элемент MODELPROPERTYLIST: Содержит список элементов MODELPROPERTY.
- Элемент MODELPROPERTY: Определяет значение свойства при создании элемента. Свойство имени, которое определяет имя устанавливаемого свойства, должно быть обязательно определено. Если имя свойства не указано или указано некорректно, элемент не будет создан. См. "Главу 4. Использование открытого API" для информации об именах свойств и диапазонах их значений.
- Элемент VIEWPROPERTYLIST: Содержит список элементов VIEWPROPERTY.
- Элемент VIEWPROPERTY: Определяет значение свойства визуального представления при создании элемента. Свойство имени, которое определяет имя свойства представления, должно быть обязательно определено. Если имя свойства не указано или указано некорректно, элемент не будет создан. См. "Главу 4. Использование открытого API" для информации об именах свойств и диапазонах их значений.
- Элемент TAGGEDVALUELIST: Содержит список элементов TAGGEDVALUE.
- Элемент TAGGEDVALUE: Определяет значение тега модельного элемента при создании элемента. Чтобы присвоить значение тегу, Вы должны указать определяющее его свойство tagDefinition .
- Свойство profile (элемент TAGGEDVALUE): Определяет имя профиля, который содержит определение тега. Этот элемент может быть опущен, в этом случае предполагается профиль, определяющий ELEMENENTPROTOTYPE.
- Свойство tagDefinitionSet - имя набора, содержащее tagDefinition.
- Свойство tagDefinition (элемент TAGGEDVALUE): Определяет имя тега.

Прототип модельного элемента

Элемент MODELPROTOTYPE описывает прототип модельного элемента, который определяет его шаблон при создании.

```
<MODELPROTOTYPE>
    <NAME>....</NAME>
    <DISPLAYNAME>....</DISPLAYNAME>
    <ICON>....</ICON>
    <BASEMODEL argument="...">....</BASEMODEL>
```

```
    <STEREOTYPENAME>....</STEREOTYPENAME>
    <PROPERTYLIST>
        <PROPERTY name=”...”>....</PROPERTY>
        ....
    </PROPERTYLIST>
    <TAGGEDVALUELIST>
        <TAGGEDVALUE profile=”...” tagDefinitionSet=”...” tagDefinition=”...”> </TAGGEDVALUE>
        ....
    </TAGGEDVALUELIST>
    <CONTAINERMODELLIST>
        <CONTAINERMODEL type="...." stereotype="...."/>
        ....
    </CONTAINERMODELLIST>
</MODELPROTOTYPE>
```

- Элемент NAME: Имя прототипа модели. Должно быть уникально в пределах профиля.
- Элемент DISPLAYNAME: Содержит название прототипа, используемое в пользовательском интерфейсе, например в меню [Add Model].
- Элемент ICON: Этот элемент содержит имя файла иконки прототипа в пользовательском интерфейсе, например в меню [Add Model]. Файл иконки прототипа должен иметь формат .BMP, размером 16 X 16. Файл иконки должен быть помещен в каталог документа профиля. Имя должно быть указано без пути.
- Элемент BASEMODEL: Определяет имя стандартного элемента UML, на основе которого создаётся прототип. Этот элемент (BASEMODEL) не может быть опущен. Если этот элемент не определен, прототип не может быть распознан. Доступные имена элементов UML те же самые, что перечислены при описании элемента BASEELEMENT раздела "Прототип элемента". Элементы, которые имеют только визуальное представление, не могут быть указаны.
- Свойство argument: Некоторым типам элементов, которые базируются на ассоциации, псевдо состоянии и так далее, требуется аргумент при создании. С помощью этого аргумента устанавливаются некоторые их специфические свойства. Заданное по умолчанию значение этого свойства - 0. В большинстве случаев его не нужно определять. Доступные значения аргумента - те же самые, что перечислены для аналогичного свойства в разделе Прототип элемента.
- Элемент STEREOTYPENAME: Определяет стереотип прототипа. Если значение этого элемента задано, оно вводится как значение свойства "Stereotype", при создании модельного элемента. Этот элемент может быть опущен.
- Элемент PROPERTYLIST: Содержит список элементов PROPERTY.
- Элемент PROPERTY: Определяет значение свойства при создании модельного элемента. Свойство имени, которое определяет имя свойства модельного элемента, должно быть обязательно определено. Если имя свойства не указано или указано некорректно, элемент не будет создан. См. "Главу 4. Использование открытого API" для информации об именах свойств и диапазонах их значений.
- Элемент TAGGEDVALUELIST: Содержит список элементов TAGGEDVALUE.
- Элемент TAGGEDVALUE: Определяет значение тега модельного элемента при создании элемента. Чтобы присвоить значение тегу, Вы должны указать определяющее его свойство tagDefinition .
- Свойство profile (элемент TAGGEDVALUE): Определяет имя профиля, который содержит определение тега. Этот элемент может быть опущен. Если он опущен, применяется профиль, который создаёт MODELPROTOTYPE.
- Свойство tagDefinitionSet (элемент TAGGEDVALUE): Определяет имя набора, содержащего тег.
- Свойство tagDefinition (элемент TAGGEDVALUE): Определяет имя тега.
- Элемент CONTAINERMODELLIST: Содержит список элементов CONTAINERMODEL.
- Элемент CONTAINERMODEL: Ограничивает контекст модельного элемента, определенного прототипом. Если значение указано, то пункт меню создания прототипа в меню [Add Model], будет активизирован только тогда, когда текущим является указанный модельный элемент.

Палитра

Элемент PALETTE описывает дополнительную палитру и её элементы.

```
<PALETTE>
    <NAME>....</NAME>
    <DISPLAYNAME>....</DISPLAYNAME>
    <PALETTEITEMLIST>
        <PALETTEITEM>....</PALETTEITEM>
        ....
    </PALETTEITEMLIST>
</PALETTE>
```

- Элемент NAME: Имя палитры. Должно быть уникально в пределах профиля.
- Элемент DISPLAYNAME: Отображаемое название название.
- Элемент PALETTEITEMLIST: Список элементов палитры.
- Элемент PALETTEITEM: Определяет имя элемента палитры. Значение этого элемента должно совпадать с именем прототипа элемента, определенного в профиле, или именем стандартного элемента UML. Доступные имена элементов UML - те же, что перечислены при описании BASEELEMENT в разделе Прототип элемента.

Тип диаграмм

Элемент DIAGRAMTYPE содержит спецификацию типа диаграммы.

```
<DIAGRAMTYPELIST>
    <DIAGRAMTYPE>
        <NAME>....</NAME>
        <DISPLAYNAME>....</DISPLAYNAME>
        <BASEDIAGRAM>....</BASEDIAGRAM>
        <ICON>....</ICON>
        <AVAILABLEPALETTELIST>
            <AVAILABLEPALETTE>....</AVAILABLEPALETTE>
            ....
        </AVAILABLEPALETTELIST>
    </DIAGRAMTYPE>
</DIAGRAMTYPELIST>
```

- Элемент NAME: Имя типа диаграммы. Должно быть уникально в пределах профиля.
- Элемент DISPLAYNAME: Отображаемое название, используемое в пользовательском интерфейсе, например в меню [Add Diagram].
- Элемент ICON: Этот элемент содержит имя файла иконки типа диаграммы в пользовательском интерфейсе (в меню [Add Diagram]). Файл иконки типа диаграммы должен иметь формат .BMP, размером 16 X 16. Файл иконки должен быть помещен в каталог документа профиля. Элемент должен указывать только имя файла без пути.
- Элемент BASEDIAGRAM: Определяет имя стандартной диаграммы UML, на базе которой создан данный тип диаграммы. Имена стандартных диаграмм UML указаны ниже.

Имена дограмм
ClassDiagram
UseCaseDiagram
SequenceDiagram
SequenceRoleDiagram
CollaborationDiagram
CollaborationRoleDiagram
StatechartDiagram
ActivityDiagram
ComponentDiagram

Имена доргамм
DeploymentDiagram CompositeStructureDiagram

- Элемент AVAILABLEPALETTELIST: Определяет список активных палитр для типа диаграммы.
- Элемент AVAILABLEPATTE: Определяет палитру, активируемую для типа диаграммы. Значением этого элемента должно быть имя палитры, определенной в профили, или встроенной палитры StarUML.

Имена встроенных палитр
UseCase Class SequenceRole Sequence CollaborationRole Collaboration Statechart Activity Component Deployment CompositeStructure Annotation

Регистрация профиля UML

Чтобы профиль автоматически распознавался программой StarUML, его следует разместить в подкаталоге каталога модулей StarUML (<install-dir> \modules). StarUML находит и читает все профили, расположенные в каталоге модулей, и автоматически регистрирует их когда стартует. Если файл профиля некорректен или имеет расширение не .prf, StarUML игнорирует его. Рекомендуется размещать профили в отдельном подкаталоге каталога модулей StarUML, во избежание их смешивания.

Обратите внимание: Чтобы зарегистрировать иконку профиля, создайте файл иконки (с таким же именем) и поместите его в каталог профиля. Иконка профиля будет показана рядом с названием профиля в списке диалога [Profiles]. Если файл иконки не найден, ему присваивается заданная по умолчанию иконка.

Обратите внимание: Просто удалите файл профиля из каталога модулей StarUML (<install-dir> \modules), чтобы он больше не использовался.

Управление объектом элемента расширения

Описание элементов расширения

К элементам расширения, определенным в профиле, можно обратиться через StarUML™ API. COM-интерфейс, связанный с расширением StarUML™, организован тем же самым способом как и актуальная структура расширения UML, и управляется через интерфейс IExtensionManager. Разработчики редко непосредственно управляют объектами элемента расширения. Напротив, разработчику намного удобнее получать стереотипы или значения тегов от актуальных расширенных модельных элементов. В этом случае могут использоваться методы, предоставленные интерфейсом IExtensionModel. Для получения детальной информации

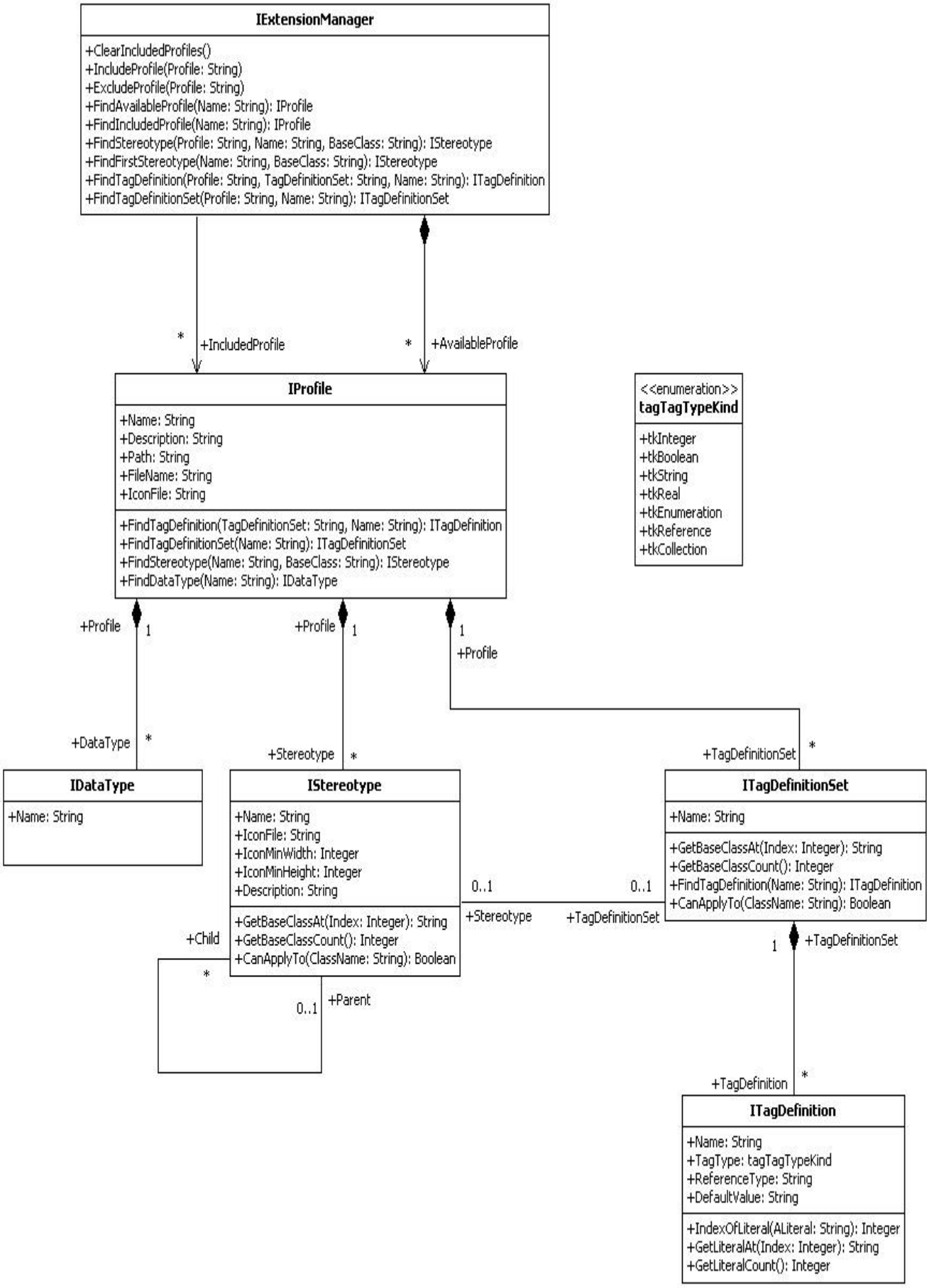
относительно интерфейса IExtensibleModel и элементов моделирования, см. "Главу 4. Использование открытого API".

Как было упомянуто ранее, элементы расширения не создаются в процессе собственно моделирования, они предварительно определены структурами расширения. Поскольку они не должны изменяться после загрузки программы или проекта, большинство свойств, определенных в этих интерфейсах, только читается.

Следующие интерфейсы доступны для обработки объектов элемента расширения.

- IExtensionManager: Управляет профилями, зарегистрированными в программе, и предоставляет методы поиска элементов расширения. IExtensionManager - первый интерфейс, который получает доступ к профилю или элементам расширения, определенным в профиле.
- IProfile: Управляет элементами расширения, определенными в профиле, и предоставляет методы доступа к ним. Он также содержит информацию о профиле. IProfile поддерживает элементы расширения, определенные в профиле, как коллекции элементов IStereotype, ITagDefinition и IDataType.
- IStereotype: Предоставляет информацию о стереотипах.
- ITagDefinitionSet: Предоставляет информацию о наборе тегов, и управляет определениями тегов, содержащимися в наборе, как коллекцией элементов ITagDefintion.
- ITagDefintion: Предоставляет информацию о теге.
- IDataType: Предоставляет информацию о типе данных.

Следующая диаграмма иллюстрирует организацию COM-интерфейсов для элементов расширения StarUML.


IExtensionManager
+ClearIncludedProfiles()
+IncludeProfile(Profile: String)
+ExcludeProfile(Profile: String)
+FindAvailableProfile(Name: String): IProfile
+FindIncludedProfile(Name: String): IProfile
+FindStereotype(Profile: String, Name: String, BaseClass: String): IStereotype
+FindFirstStereotype(Name: String, BaseClass: String): IStereotype
+FindTagDefinition(Profile: String, TagDefinitionSet: String, Name: String): ITagDefinition
+FindTagDefinitionSet(Profile: String, Name: String): ITagDefinitionSet
* +IncludedProfile
* +AvailableProfile
IProfile
+Name: String
+Description: String
+Path: String
+FileName: String
+IconFile: String
+FindTagDefinition(TagDefinitionSet: String, Name: String): ITagDefinition
+FindTagDefinitionSet(Name: String): ITagDefinitionSet
+FindStereotype(Name: String, BaseClass: String): IStereotype
+FindDataType(Name: String): IDataType
<<enumeration>>
tagTagTypeKind
+tkInteger
+tkBoolean
+tkString
+tkReal
+tkEnumeration
+tkReference
+tkCollection
+Profile 1
+DataType *
+Profile 1
+Stereotype *
1 +Profile
+TagDefinitionSet *
IDataType
+Name: String
IStereotype
+Name: String
+IconFile: String
+IconMinWidth: Integer
+IconMinHeight: Integer
+Description: String
+GetBaseClassAt(Index: Integer): String
+GetBaseClassCount(): Integer
+CanApplyTo(ClassName: String): Boolean
0..1 +Stereotype
0..1 +TagDefinitionSet
ITagDefinitionSet
+Name: String
+GetBaseClassAt(Index: Integer): String
+GetBaseClassCount(): Integer
+FindTagDefinition(Name: String): ITagDefinition
+CanApplyTo(ClassName: String): Boolean
+Child *
0..1 +Parent
1 +TagDefinitionSet
+TagDefinition *
ITagDefinition
+Name: String
+TagType: tagTagTypeKind
+ReferenceType: String
+DefaultValue: String
+IndexOfLiteral(ALiteral: String): Integer
+GetLiteralAt(Index: Integer): String
+GetLiteralCount(): Integer

Вызов IExtensionManager

Чтобы управлять профилями и элементами расширения, сначала должна быть получена ссылка на интерфейс IExtensionManager. IStarUMLApplication предоставляет свойство для обращения к объекту ExtensionManager. Следующий код - пример получения ссылки к IExtensionManager на Jscript.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var ext = app.ExtensionManager;
```

Включение и отключение профиля

IExtensionManager предоставляет методы для включения или исключая профиля в проекте. Метод IncludeProfile() включает указанный профиль в текущий проект, а ExcludeProfile() исключает указанный профиль из текущего проекта. Профиль, указанный как параметр в этих методах, должен быть зарегистрирован в системе. Если указанный профиль не существует или не зарегистрирован в системе, происходит ошибка. Сигнатуры методов следующие.

```
IExtensionManager.IncludeProfile(Profile: String)
IEXtensionManager.ExcludeProfile(Profile: String)
```

Ниже представлен пример на JScript исключения профиля "UMLStandard" из текущего проекта.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var ext = app.ExtensionManager;
ext.ExcludeProfile("UMLStandard");
```

Вызов элементов расширения, определенных в профиле

Объект расширения, определённый в профиле, доступен через интерфейс IProfile. IProfile предоставляет следующие методы доступа к коллекциям, позволяющие обратиться к интерфейсам объектов расширения (IStereotype, ITagDefinitionSet, и IDataType). Аргумент индекса, используемый в GetStereotypeAt (), GetTagDefinitionSetAt (), GetDataType (), и т.д. должен быть меньше или равен чем (Count - 1) для коллекции.

```
IProfile.GetStereotypeCount(): Integer
IProfile.GetStereotypeAt(Index: Integer): IStereotype
IProfile.GetTagDefinitionSetCount(): Integer
IProfile.GetTagDefinitionSetAt(Index: Integer): ITagDefinitionSet
IProfile.GetDataTypeCount(): Integer
IProfile.GetDataTypeAt(Index: Integer): IDataType
```

Следующий пример на Jscript показывает перебор всех стереотипов, определенных в профиле.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var ext = app.ExtensionManager;
var prf = ext.FindIncludedProfile("UMLStandard");
if (prf != null) {
    var st;
    for (i = 0; i <= prf.GetStereotypeCount() - 1; i++) {
        st = prf.GetStereotypeAt(i);
        // do something...
    }}
```

Поиск элементов расширения

Интерфейс IProfile предоставляет методы поиска интерфейсов элементов расширения, определенных в профиле.

```
FindTagDefinition(TagDefinitionSet: String, Name: String): ITagDefinition
FindTagDefinitionSet(Name: String): ITagDefinitionSet
FindStereotype(Name: String, BaseClass: String): IStereotype
FindDataType(Name: String): IDataTypeManaging
```

Обработка стереотипов

Интерфейс IStereotype предоставляет информацию о стереотипах, определенных в профиле. Основная информация о стереотипе, имя, описание и имя файла иконки, может быть получена через неизменяемые свойства интерфейса IStereotype.

IStereotype предоставляет методы для получения списка элементов UML, к которым могут быть применены стереотипы: GetBaseClassCount (), GetBaseClassAt (), CanApplyTo (), и т.д. Методы GetBaseClassCount () и GetBaseClassAt () позволяют получить имена элементов UML, которые могут быть использованы с данным стереотипом. Метод CanApplyTo() указывает, может ли элемент UML, указанный как аргумент, принять текущий стереотип, возвращая значение типа Boolean.

Свойство BaseClass стереотипа может указывать не только на элементы UML, отображаемые на диаграммах, но также и на абстрактные элементы верхнего уровня, подобные UMLClassifier. В этом случае, стереотип может быть применен ко всем элементам-потомкам указанного элемента верхнего уровня. Например, предположим, что UMLClassifier определен как BaseClass, тогда это означает то же самое, как будто бы все его потомки UMLClass, UMLInterface, UMLUseCase, и UMLActor определены как BaseClass для этого стереотипа. Для изучения структуры наследования между модельными элементами, см. Plastic Application Model.

Метод GetStereotype () интерфейса IExtensibleModel возвращает объект IStereotype указанного модельного элемента. Если стереотип элемента не определен в профиле (это допускается), будет возвращено пустое значение. В этом случае, свойство StereotypeName интерфейса IExtensibleModel может использоваться, чтобы получить незарегистрированное имя стереотипа.

Следующий пример на JScript выводит в окно сообщений описание стереотипа текущего модельного элемнта.

```
var selMgr = app.SelectionManager;
if (selMgr.GetSelectedModelCount() > 0) {
    var selModel = selMgr.GetSelectedModelAt(0);
    var st = selModel.GetStereotype();
    if (st != null) {
        WScript.Echo(st.Описание)
    }}
```

Работа с определениями тегов

Интерфейс ITagDefinition предоставляет информацию о тегах, определенных в профиле. ITagDefinition предоставляет следующие свойства.

Свойство	Описание
Name: String	Имя определения тега. Имя тега дожно быть уникальным в пределах набора.

Свойство	Описание
TagType: TagTypeKind	Тип тега. Допустимы следующие типы. • tkInteger = 0 (integer) • tkBoolean = 1 (boolean) • tkString = 2 (string) • tkReal = 3 (real number) • tkEnumeration = 4 (enumeration) • tkReference = 5 (reference) • tkCollection = 6 (collection) В зависимости от типа тега, используются различные методы получения его значения. Интерфейс IExtensibleModel включает методы получения значения тега под каждый допустимый тип.
ReferenceType: String	Указывает тип объектной ссылки, допустимой для определения значения тега, когда TagType - tkReference или tkCollection. Например, установив это свойство в "UMLClass", вы разрешаете присвоить тегу только ссылку на класс. Если определение для ReferenceType опущено в документе профиля, то по умолчанию используется "UMLModelElement". Если TagType - не tkReferece или tkCollection, это свойство не имеет никакого эффекта.
DefaultValue: String	Определяет заданное по умолчанию значение тега. Если TagType - tkEnumeration, то оно содержит строковое значение. Если TagType - tkReference или tkCollection, то заданное по умолчанию значение установлено как пустой указатель, и это свойство не имеет никакого эффекта.

Следующий пример на JScript выводит в окно сообщений заданное по умолчанию значение тега Derived из набора Default профиля UMLStandard.

```
var ext = app.ExtensionManager;
var tag = ext.FindTagDefinition("UMLStandard", "Default", "Derived");
WScript.Echo(tag.DefaultValue);
```

Глава 8. Расширение меню

Основные концепции расширения меню

Чтобы предоставить возможность пользователю вызывать функции аддинов, система меню StarUML™ может быть расширена. Для этого разработчики аддинов должны предоставить файлы расширения меню. Это подразумевает следующие шаги.

1. Создание файла расширения меню.
2. Регистрация файла расширения меню.

Файл расширения меню аддина (*.mnu) - это текстовый XML-файл. Каждый аддин должен содержать только один файл расширения меню. StarUML™ использует этот файл, чтобы расширять главное меню приложения и всплывающие меню: добавлять новые пункты меню, выполнять определенные действия, или посылать сообщения аддинам. Файл расширения меню StarUML™ может содержать следующие определения.

- Новые пункты меню для добавления
- Деление элементов главного меню и пунктов всплывающих меню
- Пункты основного меню StarUML, в которые добавляются новые пункты меню
- Отображение названий и вызывающих клавиш для пунктов меню
- Точки активации и дезактивации пунктов меню
- Файлы скриптов, которые выполняются, когда выбираются пункты меню
- Идентификаторы пунктов меню, которые посылаются объектам аддина при выборе
- Местоположения пунктов меню в меню верхнего уровня
- Файлы иконок для пунктов меню

Файл расширения меню пишется в формате XML. Это должен быть правильно построенный документ, и его содержание должно быть корректно. Эта глава обсуждает DTD XML (Document Type Definition - Определение типа документа, которое должно быть соблюдено, чтобы гарантировать корректность файлов расширения меню), структуру файлов расширения меню, а также предоставляет соответствующие примеры.

Обратите внимание: файл расширения меню должен иметь расширение *.mnu и находится в некотором подкаталоге каталога модулей StarUML™ (<insall-директор> \modules).

Создание файла расширения меню

DTD файла расширения меню

Файл расширения меню должен содержать XML, который соответствует определенному DTD. Ниже показано полное содержание DTD, определенное для файла расширения меню.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT VERSION (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ELEMENT COMPANY (#PCDATA)>
<!ELEMENT COPYRIGHT (#PCDATA)>
```

```
<!ELEMENT MAINITEM (MAINITEM)*>
<!ATTLIST MAINITEM
          base (FILE|EDIT|FORMAT|MODEL|VIEW|TOOLS|HELP|UNITS|IMPORT|EXPORT|NEW_TOP) #IMPLIED
          caption CDATA #REQUIRED
          index CDATA #IMPLIED
          beginGroup CDATA #IMPLIED
          script CDATA #IMPLIED
          actionId CDATA #IMPLIED
          availableWhen (ALWAYS|PROJECT_OPENED|MODEL_SELECTED|VIEW_SELECTED|UNIT_SELECTED|
DIAGRAM_ACTIVATED) "PROJECT_OPENED"
          iconFile CDATA #IMPLIED>
<!ELEMENT POPUPITEM (POPUPITEM)*>
<!ATTLIST POPUPITEM
          location (EXPLORER|DIAGRAM|BOTH) "BOTH"
          caption CDATA #REQUIRED
          index CDATA #IMPLIED
          beginGroup CDATA #IMPLIED
          script CDATA #IMPLIED
          actionId CDATA #IMPLIED
          availableWhen (ALWAYS|PROJECT_OPENED|MODEL_SELECTED|VIEW_SELECTED|UNIT_SELECTED|
DIAGRAM_ACTIVATED) "PROJECT_OPENED"
          iconFile CDATA #IMPLIED>
<!ELEMENT MAIMENU (MAINITEM)*>
<!ELEMENT POPUPMENU (POPUPITEM)*>
<!ELEMENT HEADER (NAME?, VERSION?, DESCRIPTION?, COMPANY?, COPYRIGHT?)>
<!ELEMENT BODY (MAINMENU?, POPUPMENU?)>
<!ELEMENT ADDINMENU (HEADER?, BODY)>
<!ATTLIST ADDINMENU addInID CDATA #REQUIRED>
```

Обратите внимание: Имена всех элементов XML должны быть написаны в символах верхнего регистра, а имена всех атрибутов начинаться со строчных букв. Предопределенные значения символов представляются на верхнем регистре с символами '_' (символ подчеркивания). Эти соглашения должны быть соблюдены всюду в файле меню, а предопределенные значения символов должны использоваться должным образом.

Полная структура файла расширения меню

Файлы расширения меню соответствуют правилам составления документов XML, а определяемые пользователем пункты меню описываются внутри элемента 'ADDINMENU'.

```
<?xml version="1.0" encoding="..."?>
<ADDINMENU addInID="...">
  <HEADER>...</HEADER>
  <BODY>...</BODY>
</ADDINMENU>
```

- Свойство encoding: Определяет кодировку документа XML (например. UTF-8, EUC-KR). Для подробного ознакомления с назначением этого свойства см. описание XML.
- Свойство addInID: Уникальный идентификатор аддина. Это должно быть уникальное значение, которое идентифицирует текущий аддин. Рекомендуется, чтобы название компании или название изделия использовались как часть этого значения (например, StarUML.StandardAddIn).
- Элемент HEADER: Содержит общую информацию об аддине. См. раздел Элемент HEADER.
- Элемент BODY: Содержит информацию об актуальных пунктах меню. См. раздел Элемент BODY.

Элемент HEADER

Элемент заголовка файла расширения меню содержит информацию о файле меню. Содержание заголовка не окзывает никакого влияния на фактическую структуру пунктов меню. Хотя эта секция

может быть опущена, рекомендуется включать её, чтобы файлы расширения меню были легко идентифицируемы.

```
<HEADER>
  <NAME>...</NAME>
  <VERSION>...</VERSION>
  <DESCRIPTION>...</DESCRIPTION>
  <COMPANY>...</COMPANY>
  <COPYRIGHT>...</COPYRIGHT>
</HEADER>
```

- Элемент NAME: Содержит название адина (строковое значение).
- Элемент VERSION: Содержит информацию о версии (строковое значение).
- Элемент DESCRIPTION: Содержит краткое описание аддина (строковое значение).
- Элемент COMPANY: Содержит информацию о разработчике (строковое значение).
- Элемент COPYRIGHT: Содержит объявление об авторском праве (строковое значение).

Элемент BODY

Элемент BODY файла расширения меню содержит спецификации добавляемых пунктов меню. Информация этой секции должна быть точной.

```
<BODY>
  <MAINMENU>
    <MAINITEM>...</MAINITEM>
    <MAINITEM>...</MAINITEM>
  </MAINMENU>
  <POPUPMENU>
    <POPUPITEM>...</POPUPITEM>
    <POPUPITEM>...</POPUPITEM>
  </POPUPMENU>
</BODY>
```

Элемент BODY может быть разделен на определения главного меню и определения контекстного меню.

- Элемент MAINMENU: Описывает добавляемые элементы главного меню.
- Элемент POPUPMENU: Описывает добавляемые элементы всплывающих меню.
- Элемент MAINITEM: Содержит актуальную информацию о пункте меню (главное меню).
- Элемент POPUPITEM: Содержит актуальную информацию о пункте меню (контекстное меню).

Элементы главного меню и пункты всплывающих меню описываются отдельно. В соответствии с функциями, предоставляемыми каждым аддином, пункты меню можно добавлять в главное меню или в контекстные меню. Поэтому элемент MAINMENU или элемент POPUPMENU может быть опущен, но не оба сразу. Если же одни и те же функциональные возможности нужно добавить и в главное меню и в контекстное меню, соответствующая информация должна быть введена как в MAINMENU, так и в POPUPMENU. В этом случае эти два элемента должны вызывать один скрипт или иметь одинаковые значения свойства actionID. Однако, при добавлении пункта меню низшего уровня к основному пункту меню типа [Format] и [Unit], который доступен и в главном меню и в контекстных меню, информация должна добавляться в только MAINMENU.

Элемент MAINMENU

Элемент MAINMENU может содержать несколько элементов MAINITEM. Каждый элемент MAINITEM составляет один элемент главного меню. Чтобы определить суб-меню с пунктами

меню, элемент MAINITEM может в свою очередь содержать несколько элементов MAINITEM.

```
<MAINITEM base=”...” caption=”...” index=”...” beginGroup=”...” script=”...” actionId=”...”
availableWhen=”...” iconFile=”...”>
  <MAINITEM>...</MAINITEM>
  <MAINITEM>...</MAINITEM>
</MAINITEM>
```

Обратите внимание: Если пункт меню не соответствует субменю, для него должна быть указана ссылка на выполняемый скрипт или идентификатор вызываемой акции.

Свойство	Описание	Диапазон значений	Пропуск
base	Основное свойство добавляемого пункта меню. Это свойство игнорируется, если элемент MAINITEM находится внутри другого элемента MAINITEM.	Допускается: FILE, EDIT, FORMAT, MODEL, VIEW, TOOLS, HELP, UNITS, IMPORT, EXPORT, or NEW_TOP. *	Если опущено, новый пункт добавляется как субменю меню **[Tools]** .
caption	Указывает отображаемое название пункта меню. Это значение может содержать код клавиши. Чтобы определить горячую клавишу, добавьте '&' и код символа за этим значением. Учтите, что StarUML™ не проверяет дублирования горячих клавиш в других пунктах меню.	Строковое значение	Не может быть опущено.
index	Указывает порядковый номер пункта меню. Например, если это значение '0', пункт меню будет следовать первым за стандартными пунктами. Если данное значение конфликтует с другими аналогичными значениями, меню может быть отображено некорректно.	Целое, большее 0.	Обычно опущен. Если опущено, добавляется в порядке регистрации аддина.
beginGroup	Указывает, помещать ли сепаратор перед пунктом меню	TRUE или FALSE.	Если опущено, принимается FALSE.
script	Содержит полное имя выполняемого скрипта, если указано. Допускается относительный маршрут от каталога аддина. Может содержать URL.	Строковое значение	Может быть опущено.
actionId	Содержит положительное целое, которое передаётся COM объекту аддина. Если аддин добавляет несколько пунктов меню, каждый пункт может отличаться уникальным значением идентификатора.	Целое, большее 0.	Может быть опущено.
availableWhen	Указывает когда пункт меню доступен.	Допускается: ALWAYS, PROJECT_OPENED, MODEL_SELECTED, VIEW_SELECTED, UNIT_SELECTED, or DIAGRAM_ACTIVATED. **	Если опущено, принимается PROJECT_OPENED

Свойство	Описание	Диапазон значений	Пропуск
iconFile	Содержит полное имя файла иконки, если указано. Допускается относительный маршрут от каталога аддина.	Строковое значение	Может быть опущено.

***диапазон значений свойства Base**

- FILE: Пункт меню добавляется как пункт субменю меню [File].
- EDIT: Пункт меню добавляется как пункт субменю меню [Edit].
- FORMAT: Пункт меню добавляется как элемент подменю меню [Format].
- MODEL: Пункт меню добавляется как элемент подменю меню [Model].
- VIEW: Пункт меню добавляется как элемент подменю меню [View].
- TOOLS: Пункт меню добавляется как элемент подменю меню [Tools]. (значение по умолчанию)
- HELP: Пункт меню добавляется как элемент подменю меню [Help].
- UNITS: Пункт меню добавляется как элемент подменю меню [Файл]-> [Unit].
- IMPORT: Пункт меню добавляется как элемент подменю меню [Файл]-> [Import].
- EXPORT: Пункт меню добавляется как элемент подменю меню [Файл]->[Export].
- NEW_TOP: Пункт меню создаётся как новый элемент главного меню верхнего уровня.

**** диапазон значений свойства availableWhen**

- ALWAYS: Доступно, пока выполняется приложение StarUML™.
- PROJECT_OPENED: Доступно, когда присутствует элемент проекта. (значение по умолчанию)
- MODEL_SELECTED: Доступно, когда выбран модельный элемент.
- VIEW_SELECTED: Доступно, когда выбран элемент представления.
- UNIT_SELECTED: Доступно, когда выбрана секция.
- DIAGRAM_ACTIVATED: Доступно, когда открыта диаграмма.

Элемент POPUPMENU

Элемент POPUPMENU может содержать несколько элементов POPUPITEM. Каждый элемент POPUPITEM соответствует одному пункту всплывающего меню. Чтобы определить пункт меню с подпунктами, элемент POPUPITEM может в свою очередь содержать несколько элементов POPUPITEM.

```
<POPUPITEM location="..." caption="..." index="..." beginGroup="..." script="..."
actionId="..." availableWhen="..." iconFile="...">
  <POPUPITEM>...</POPUPITEM>
  <POPUPITEM>...</POPUPITEM>
</POPUPITEM>
```

Обратите внимание: Если пункт меню не соответствует субменю, для него должна быть указана ссылка на выполняемый скрипт или идентификатор вызываемой акции.

Property	Описание	Range of Value	Omission
location	Указывает тип контекстных меню, в который добавляется пункт. Это свойство игнорируется, если элемент POPUPITEM находится внутри другого элемента POPUPITEM.	Допускается: EXPLORER, DIAGRAM, or BOTH. *	Если опущено, принимается BOTH.

Property	Описание	Range of Value	Omission
caption	Указывает отображаемое название пункта меню. Это значение может содержать код клавиши. Чтобы определить горячую клавишу, добавьте '&' и код символа за этим значением. Учтите, что StarUML™ не проверяет дублирования горячих клавиш в других пунктах меню.	Строковое значение	Не может быть опущено.
index	Указывает порядковый номер пункта меню. Например, если это значение '0', пункт меню будет следовать первым за стандартными пунктами. Если данное значение конфликтует с другими аналогичными значениями, меню может быть отображено некорректно.	Целое, боьшее 0.	Обычно опущен. Если опущено, добавляется в порядке регистрации аддина.
beginGroup	Указывает, помещать ли сепаратор перед пунктом меню	TRUE или FALSE.	FALSE if omitted.
script	Содержит полное имя выполняемого скрипта, если указано. Допускается относительный маршрут от каталога аддина. Может содержать URL.	Строковое значение	Может быть опущено.
actionId	Содержит положительное целое, которое передаётся COM объекту аддина. Если аддин добавляет несколько пунктов меню, каждый пункт может отличаться уникальным значением идентификатора.	Целое, большее 0.	Может быть опущено.
availableWhen	Указывает когда пункт меню доступен.	Допускается ALWAYS, PROJECT_OPENED, MODEL_SELECTED, VIEW_SELECTED, UNIT_SELECTED, or DIAGRAM_ACTIVATED **	Если опущено, принимается PROJECT_OPENED.
iconFile	Содержит полное имя файла иконки, если указано. Допускается относительный маршрут от каталога аддина.	Строковое значение	Может быть опущено.

*** диапазон значений свойства location**

- EXPLORER: Пункт меню добавляется к контекстному меню Навигатора модели.
- DIAGRAM: Пункт меню добавляется к контекстному меню Диаграммы.
- BOTH: Пункт меню добавляется к обоим контекстные меню Диаграммы и Навигатора модели. (значение по умолчанию)

**** диапазон значений свойства availableWhen - Тот же самый, что и для элемента MAINMENU.**

Пример файла расширения меню

Следующий пример показывает реальный файл описания меню для одного из стандартных расширений, которое устанавливается вместе с программой StarUML™ .

```
<?xml version="1.0" encoding="UTF-8"?>
<ADDINMENU addInID="StarUML.StandardAddIn">
    <HEADER>
        <NAME>Default module of StarUML</NAME>
        <VERSION>1.0.0</VERSION>
        <DESCRIPTION>Default extension pack of Agora Plastic to convert diagram</DESCRIPTION>
        <COMPANY>Plastic Software, Inc.</COMPANY>
        <COPYRIGHT>Copyright (C) 2005 Plastic Software, Inc. All rights reserved.</COPYRIGHT>
    </HEADER>
    <BODY>
        <MAINMENU>
            <MAINITEM base="MODEL" caption="Convert Diagram" beginGroup="TRUE"
availableWhen="MODEL_SELECTED">
            <MAINITEM caption="Convert Sequence(Role) to Collaboration(Role)"
script="ConvSeq2Col.vbs"/>
            <MAINITEM caption="Convert Collaboration(Role) to Sequence(Role)"
script="ConvCol2Seq.vbs"/>
        </MAINMENU>
    </BODY>
</ADDINMENU>
```

Регистрация файла расширения меню

Чтобы расширение меню было распознано автоматически, файл нужно поместить подкаталог каталога модулей StarUML (<install-dir> \modules). StarUML находит и читает все файлы расширения меню, расположенные в каталоге модулей и автоматически регистрирует их во время своей инициализации. Если файл расширения меню некорректен, или его расширение не .mnu, он будет игнорирован. Рекомендуется создать подкаталог в каталоге модулей StarUML и поместить файл туда, чтобы избежать путаницы.

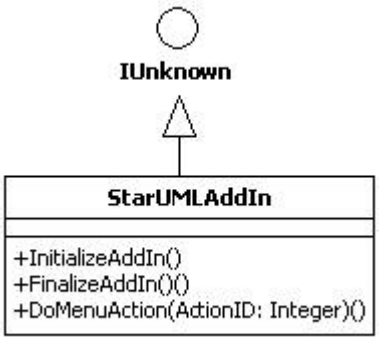
Обратите внимание: Просто удалите файл расширения меню из каталога модулей StarUML (<install-dir> \modules), чтобы данное расширение меню больше не использовалось.

Глава 9. Написание дополнительных COM-объектов

Основные концепции дополнительных COM-объектов

Как отмечалось в "Главе 3. Пример Здравствуй мир", чтобы добавить новые функциональные возможности к StarUML™ могут быть использованы несложные скрипты. Однако, чтобы реализовать более сложные и полезные функциональные возможности, лучше использовать среду разработки программ, которая поддерживает COM-объекты. Если Вы хотите создать аддин для StarUML™, не имеет значения, будет ли использоваться Delphi, Visual Basic или любая другая среда программирования, лишь бы она поддерживала технологию COM.

Наиболее важный аспект создания аддина для StarUML™ заключается в том, что должен использоваться интерфейс IStarUMLAddIn, определенный в StarUML™.



Как показано выше, интерфейс IStarUMLAddIn наследован от IUnknown и определяет три дополнительных метода: InitializeAddIn (), FinalizeAddIn (), и DoMenuAction ().

Методы интерфейса IStarUMLAddIn

Методы, определенные для реализации интерфейса IStarUMLAddIn следующие.

Метод	Описание
InitializeAddIn()	Метод InitializeAddIn() используется объектом StarUMLApplication, чтобы инициализировать COM-объект аддина, когда он будет создан. Как будет показано ниже, он используется, чтобы выполнить действия, требуемые при инициализации аддина, такие как регистрация подписки на события.
FinalizeAddIn()	Метод FinalizeAddIn() вызывается объектом StarUMLApplication непосредственно перед отсоединением от объекта аддина. Как будет

Метод	Описание
	показано ниже, он используется, чтобы выполнить действия, требуемые при завершении аддина, такие как удаление подписки на события.
DoMenuAction(ActionID: Integer)	Как отмечено в 'Главе 8. Расширение меню' метод DoMenuAction () вызывается, когда пользователь выбирает дополнительный пункт меню, определенный аддином. Значение 'actionId' пункта меню, определенного файлом расширения меню, передаётся как параметр.

Пример COM-объекта аддина

Ниже показан простой пример объявления COM-объекта аддина, реализующего интерфейс IStarUMLAddIn. Пример написан на Delphi Pascal.

```
type
    AddInExample = class(TComObject, IStarUMLAddIn)
    private
        StarUMLApp: IStarUMLApplication;
    protected
        function InitializeAddIn: HResult; stdcall;
        function FinalizeAddIn: HResult; stdcall;
        function DoMenuAction(ActionID: Integer): HResult; stdcall;
        ...
    public
        procedure Initialize; override;
        destructor Destroy; override;
        ...
    end;

...

implementation

procedure AddInExample.Initialize;
begin
    inherited;
    StarUMLApp := CreateOleObject('StarUML.StarUMLApplication') as IStarUMLApplication;
    ...
end;

destructor AddInExample.Destroy;
begin
    ...
    StarUMLApp := nil;
    inherited;
end;

function AddInExample.InitializeAddIn: HResult;
begin
    ...
    Result := S_OK;
end;

function AddInExample.FinalizeAddIn: HResult;
begin
    ...
    Result := S_OK;
end;

function AddInExample.DoMenuAction(ActionID: Integer): HResult; stdcall;
begin
    Result := S_OK;
    ...
end;
```

Разработка файла описания аддина

Основная концепция файла описания аддина

Файл описания аддина (*.aid) - это текстовый XML-файл. Все аддины, подсоединяемые к StarUML, должны предоставлять собственный файл описания. StarUML регистрирует объект аддина в системном реестре и инициализирует объект аддина, а также связанный с ним файл расширения меню, на который ссылается файл описания.

Обратите внимание: Файл описания должен иметь расширение *.aid и находится в подкаталоге каталога модулей StarUML (<install-dir>\modules).

Структура файла описания аддина

Файл описания аддина создаётся согласно правилам формата XML, определяемые пользователем реквизиты аддина, должен располагаться в пределах элемента 'ADDIN'.

```
<?xml version="1.0" encoding="..."?>
<ADDIN>
    <NAME>...</NAME>
    <DISPLAYNAME>...</DISPLAYNAME>
    <COMOBJ>...</COMOBJ>
    <FILENAME>...</FILENAME>
    <COMPANY>...</COMPANY>
    <COPYRIGHT>...</COPYRIGHT>
    <HELPFILE>...</HELPFILE>
    <ICONFILE>...</ICONFILE>
    <ISACTIVE>...</ISACTIVE>
    <MENUFILE>...</MENUFILE>
    <VERSION>...</VERSION>
    <MODULES>
        <MODULEFILENAME>...</MODULEFILENAME>
    </MODULES>
</ADDIN>
```

- Свойство encoding: Определяет кодировку XML документа (например. UTF-8, EUC-KR). Для получения информации об этом свойстве, см. документацию на XML.
- Элемент NAME: Определяет имя аддина. (строковое значение)
- Элемент DISPALYNAME: Определяет название аддина, которое отображается в интерфейсе пользователя. (строковое значение)
- Элемент COMOBJ: Определяет ProgID COM объекта. Этот элемент используется только для аддинов на базе COM объекта. (строковое значение)
- Элемент FILENAME: Указывает имя файла аддина. (строковое значение)
- Элемент COMPANY: Содержит информацию о разработчике аддина. (строковое значение)
- Элемент COPYRIGHT: Описывает объявление об авторском праве. (строковое значение)
- Элемент HELPFILE: Определяет URL, который содержит справку аддина. (строковое значение)
- Элемент ICONFILE: Содержит имя файла иконки аддина. (строковое значение)
- Элемент ISACTIVE: Определяет, активизируется ли аддин автоматически при старте программы. (булево значение)
- Элемент MENUFILE: Содержит имя файла расширения меню, связанного с аддином. (строковое значение)
- Элемент MODULES/MODULEFILENAME: Указывает имена файлов дополнительных COM-объектов в случае, если аддин использует другие COM-объекты. StarUML регистрирует все дополнительные COM-объекты, указанные в этом элементе при выполнении. (строковое значение)

Регистрация файла описания аддина

Чтобы файл описания аддина был автоматически распознан программой StarUML, он должен находиться в подкаталоге каталога модулей StarUML (<install-dir>\modules). StarUML находит и читает все файлы описания аддинов, расположенные в каталоге модулей и автоматически регистрирует их при инициализации. Если файл описания аддина некорректен или имеет расширение не .aid, StarUML игнорирует его. Рекомендуется создать подкаталог в каталоге модулей StarUML и поместить файл описания аддина туда, чтобы избежать беспорядочного хранения модулей.

Обратите внимание: Просто удалите файл описания аддина из каталога модулей StarUML (<install-dir>\modules), чтобы аддин больше не использовался.

Расширение опций

Основная концепция расширения опций

StarUML поддерживает опции настройки, чтобы корректировать среду разработки и настраивать функции StarUML. Опции необходимы не только приложению StarUML, но также и аддинам сторонних разработчиков. Расширение опций StarUML дает возможность аддинам выполнять функции конфигурирования без дополнительной реализации. Для того, чтобы использовать расширение опций, разработчику аддина нужно только определить опции в текстовом файле и поместить этот текстовый файл в каталог аддина. Эти определения будут загружены программой при инициализации и отображены в диалоге опций. Разработчик аддина может экономить время и усилия при реализации аддина, который сам предоставляет соответствующий интерфейс пользователям.

Выполните шаги, указанные ниже, чтобы обеспечить поддержку опций настройки в аддине.

1. Создайте файл описания схемы опций (.opt), чтобы определить элементы опции для аддина.
2. Скопируйте файл описания схемы опции (.opt) в подкаталог каталога модулей.

Иерархия схемы опций

StarUML создает показанную ниже иерархическую структуру опций, чтобы управлять множеством опциональных параметров, которые определяют в совокупности приложение и аддины.

- Схема опций: Самый высокий уровень структуры опций, задаваемый секцией файла схемы опций. Схема изображается как значок папки на верхнем уровне дерева, которое помещается в левой части диалога опций.
- Категория опций: Раздел схемы опций, отображаемый на уровень ниже в дереве, которое помещается в левой части диалога опций.
- Класс опций: Более детальный раздел классификации опций, соответствующий группировочной строке инспектора параметров в диалоге опций. Содержит несколько элементов опций (параметров), которые могут быть отредактированы.
- Элемент опций: Редактируемая опция, соответствует одной строке инспектора параметров в диалоге опций.

Написание схемы опций

Файл схемы опций, определяющий опциональные элементы, - это текстовый XML-файл, имеющий расширение *.opt. Содержимое схемы опций находится в пределах элемента OPTIONSCHEMA, и не должно содержать никаких ошибок в синтаксисе или структуре.

```
<?xml version=”1.0” encoding=”...” ?>
<OPTIONSCHEMA id="...">
    <HEADER>
    ...
    </HEADER>
    <BODY>
    ...
    </BODY>
</OPTIONSCHEMA>
```

- Свойство encoding: Определяет кодировку XML документа (например. UTF-8, EUC-KR).
- Свойство ID (элемент OPTIONSCHEMA): Определяет имя схемы опции. Это - уникальное имя, чтобы отличать данную схему опции от других.
- Элемент HEADER: См., раздел Элемент HEADER.
- Элемент BODY: См., раздел Элемент BODY.

Элемент HEADER

Раздел HEADER описания схемы опции содержит общую информацию о схеме опций типа заголовка схемы опций и её описания. Структура раздела заголовка следующая.

```
<HEADER>
    <CAPTION>...</CAPTION>
    <DESCRIPTION>...</DESCRIPTION>
</HEADER>
```

- Элемент CAPTION: Это - название схемы опций, отображаемое как заголовок узла в дереве диалога опций.
- Элемент DESCRIPTION: Содержит описание схемы опции.

Элемент BODY

Раздел BODY описания схемы опции содержит иерархическое определение всех опциональных элементов.

```
<BODY>
    <OPTIONCATEGORY>
        <CAPTION>...</CAPTION>
        <DESCRIPTION>...</DESCRIPTION>
        <OPTIONCLASSIFICATION>
            <CAPTION>...</CAPTION>
            <DESCRIPTION>...</DESCRIPTION>
            <OPTIONITEM>
               ...
            </OPTIONITEM>
            ...
        </OPTIONCLASSIFICATION>
          ...
    </OPTIONCATEGORY>
     ...
</BODY>
```

- Элемент OPTIONCATEGORY: Определяет структуру категории опций.

- Элемент CAPTION: Определяет название категории опции, отображаемое как заголовок узла в дереве диалога опций.
- Элемент DESCRIPTION: Содержит краткое описание категории опций, которое отображается в поле описания опции диалога опций.
- Элемент OPTIONITEM: Определяет множество элементов опций. См. раздел Определение элемента опции.

Определение элемента опции

Элемент OPTIONCLASSIFICATION может содержать множество определений элементов опций. Тип элемента опции определяется как один из: integer, real, boolean, enumeration и т.п.. Диалог опций поддерживает ввод значений и ограничивает значение опции согласно типу элемента.

Допустимые типы элементов опций показаны ниже.

Тип опции	Имя элемента XML	Ввод в диалоге опций
Integer	OPTIONITEM-INTEGER	Вводится только целе число.
Real	OPTIONITEM-REAL	Вводится только действительное число.
String	OPTIONITEM-STRING	Вводится только строка.
Boolean	OPTIONITEM-BOOLEAN	Вводится только true или false через чек-бокс.
Text	OPTIONITEM-TEXT	Вводится многострочный текст через текст-бокс.
Enumeration	OPTIONITEM-ENUMERATION	Выбирается один элемент из списка OPTION-ENUMERATIONITEM.
Font name	OPTIONITEM-FONTNAME	Выбирается шрифт, инсталлированный в системе.
File name	OPTIONITEM-FILENAME	Вводится имя файла или выбирается файл в диалоге открытия файла.
Path name	OPTIONITEM-PATHNAME	Вводится имя каталога или выбирается каталог в диалоге открытия каталога.
Color	OPTIONITEM-COLOR	Выбирается цвет в диалоге или комбо-боксе.
Range	OPTIONITEM-RANGE	Вводится целое число в указанном диапазоне. Его значение можно изменять на фиксированную величину спин-кнопкой.

Следующий пример представляет формат определений элементов опций разных типов, которые помещаются в элемент OPTIONCLASSIFICATION в файле схемы опций.

```
<OPTIONCLASSIFICATION>
    <OPTIONITEM-INTEGER key="...">
        <CAPTION>...</CAPTION>
        <DESCRIPTION>...</DESCRIPTION>
        <DEFAULTVALUE>...</DEFAULTVALUE>
    </OPTIONITEM-INTEGER>
    <OPTIONITEM-REAL key="...">
        <CAPTION>...</CAPTION>
        <DESCRIPTION>...</DESCRIPTION>
        <DEFAULTVALUE>...</DEFAULTVALUE>
    </OPTIONITEM-REAL>
    <OPTIONITEM-STRING key="...">
        <CAPTION>...</CAPTION>
        <DESCRIPTION>...</DESCRIPTION>
        <DEFAULTVALUE>...</DEFAULTVALUE>
    </OPTIONITEM-STRING>
    <OPTIONITEM-BOOLEAN key="...">
        <CAPTION>...</CAPTION>
        <DESCRIPTION>...</DESCRIPTION>
        <DEFAULTVALUE>...</DEFAULTVALUE>
    </OPTIONITEM-BOOLEAN>
    <OPTIONITEM-TEXT key="...">
        <CAPTION>...</CAPTION>
```

```
        <DESCRIPTION>...</DESCRIPTION>
        <DEFAULTVALUE>...</DEFAULTVALUE>
    </OPTIONITEM-TEXT>
    <OPTIONITEM-ENUMERATION key="...">
        <CAPTION>...</CAPTION>
        <DESCRIPTION>...</DESCRIPTION>
        <DEFAULTVALUE>...</DEFAULTVALUE>
        <ENUMERATIONITEM>...</ENUMERATIONITEM>
        ...
    </OPTIONITEM-ENUMERATION>
    <OPTIONITEM-FONTNAME key="...">
        <CAPTION>...</CAPTION>
        <DESCRIPTION>...</DESCRIPTION>
        <DEFAULTVALUE>...</DEFAULTVALUE>
    </OPTIONITEM-FONTNAME>
    <OPTIONITEM-FILENAME key="...">
        <CAPTION>...</CAPTION>
        <DESCRIPTION>...</DESCRIPTION>
        <DEFAULTVALUE>...</DEFAULTVALUE>
    </OPTIONITEM-FILENAME>
    <OPTIONITEM-PATHNAME key="...">
        <CAPTION>...</CAPTION>
        <DESCRIPTION>...</DESCRIPTION>
        <DEFAULTVALUE>...</DEFAULTVALUE>
    </OPTIONITEM-PATHNAME>
    <OPTIONITEM-COLOR key="...">
        <CAPTION>...</CAPTION>
        <DESCRIPTION>...</DESCRIPTION>
        <DEFAULTVALUE>...</DEFAULTVALUE>
    </OPTIONITEM-COLOR>
    <OPTIONITEM-RANGE key="...">
        <CAPTION>...</CAPTION>
        <DESCRIPTION>...</DESCRIPTION>
        <DEFAULTVALUE>...</DEFAULTVALUE>
        <MINVALUE>...</MINVALUE>
        <MAXVALUE>...</MAXVALUE>
        <STEP>...</STEP>
    </OPTIONITEM-RANGE>
     ...
</OPTIONITEMCLASSIFICATION>
```

- Свойство key (все элементы OPTIONITEM): Определяет собственное ключевое значение элемента опции, которое должно быть уникальным в схеме опций. Оно используется при чтении значения опции COM-объектом.
- Элемент CAPTION: Определяет пояснение к элементу, используемое в диалоге опций.
- Элемент DESCRIPTION: Содержит краткое описание элемента, которое отображается в поле описания опции диалога опций.
- Элемент DEFAULTVALUE: Определяет значение опции по умолчанию. Оно должно находиться в диапазоне допустимых значений опции. Если значение по умолчанию не соответствует типу опции, опция не может редактироваться в диалоге опций.

Тип опции	Диапазон допустимых значений
OPTIONITEM-INTEGER	Целое в пределах -2147483648 ~ 2147483647
OPTIONITEM-REAL	Целое число или значение с плавающей точкой
OPTIONITEM-STRING	Строковое значние
OPTIONITEM-BOOLEAN	True или False
OPTIONITEM-TEXT	Строковое значение
OPTIONITEM-ENUMERATION	Строка, определённая в элементе ENUMERATIONITEM
OPTIONITEM-FONTNAME	Имя шрифта, напр. Tahoma
OPTIONITEM-FILENAME	Полное имя файла с маршрутом

Тип опции	Диапазон допустимых значений	
	напр. C:\My Document\Default.xml	
OPTIONITEM-PATHNAME	Допустимый маршрут или пустая строка напр. C:\My Document	
OPTIONITEM-COLOR	Строка следующего формата ${W}{B}{G}{R}	
	{W}	Зарезервировано . Может быть 00
	{B}	Синий тон цвета. Шестнадцатиричное значение 0 ~ 255 (00 ~ FF)
	{G}	Зелёноый тон цвета. Шестнадцатиричное значение 0 ~ 255 (00 ~ FF)
	{R}	Красный тон цвета. Шестнадцатиричное значение 0 ~ 255 (00 ~ FF)
	напр. $00FF0000 , $00A0A0A0, $00FF00FF	
OPTIONITEM-RANGE	Целое значение, лежащее между минмальным MINVALUE и максимальным MAXVALUE	

- Элемент ENUMERATIONITEM: Перечисляет значения, которые может принимать опция перечислимого типа (OPTION-ENUMERATION). Элемент OPTION-ENUMERATION должен иметь не менее одного элемента ENUMERATIONITEM.
- Элемент MINVALUE: Определяет, минимальное целочисленное значение интервальной опции (OPTION-RANGE).
- Элемент MAXVALUE: Определяет, максимальное целочисленное значение интервальной опции (OPTION-RANGE).
- Элемент STEP: Определяет, величину приращения значения интервальной опции, которая применяется при щелчке по spin-кнопке.

Следующий пример показывает часть файла схемы опций для StarUML.

```
<?xml version="1.0" encoding="UTF-8" ?>
<OPTIONSCHEMA id="ENVIRONMENT">
    <HEADER>
        <CAPTION>Environment</CAPTION>
        <DESCRIPTION> </DESCRIPTION>
    </HEADER>
    <BODY>
        <OPTIONCATEGORY>
            <CAPTION>General</CAPTION>
            <DESCRIPTION>General Configuration is a group of the basic and general option
items for the program. This category includes the [General], [Browser], [Collection Editor]
and [Web] subcategories.</DESCRIPTION>
            <OPTIONCLASSIFICATION>
                <CAPTION>General</CAPTION>
                <DESCRIPTION></DESCRIPTION>
                <OPTIONITEM-RANGE key="UNDO_LEVEL">
                    <CAPTION>Max. number of undo actions</CAPTION>
                    <DESCRIPTION>Specifies the maximum number of actions for undo and
redo.</DESCRIPTION>
                    <DEFAULTVALUE>30</DEFAULTVALUE>
                    <MINVALUE>1</MINVALUE>
                    <MAXVALUE>100</MAXVALUE>
                    <STEP>1</STEP>
                </OPTIONITEM-RANGE>
                <OPTIONITEM-BOOLEAN key="CREATE_BACKUP">
                    <CAPTION>Create backup files</CAPTION>
                    <DESCRIPTION>Specifies whether to create backup files when saving
changes.</DESCRIPTION>
                    <DEFAULTVALUE>True</DEFAULTVALUE>
                </OPTIONITEM-BOOLEAN>
            </OPTIONCLASSIFICATION>
        </OPTIONCATEGORY>
    </BODY>
</OPTIONSCHEMA>
```

Регистрация схемы опций

Чтобы схема опций была распознаваема автоматически, необходимо поместить файл схемы в подкаталог каталога модулей (<install-dir>\modules). StarUML находит и читает все файлы схем опций в каталоге модулей и автоматически регистрирует их во время инициализации. Если файл схемы опций некорректен или его расширение не .opt, он будет игнорирован. Рекомендуется, создать подкаталог в каталоге модулей и поместить файл описания опций туда, во избежание беспорядка при хранении модулей.

Обратите внимание: Просто удалите файл схемы опций из каталога модулей (<install-dir>\modules), чтобы он больше не использовался.

Доступ к значениям опций

Доступ к значениям опции с использованием COM-интерфейса

Вы можете обратиться к значениям опций, которые пользователь изменял в диалоге опций, используя COM интерфейс StarUML.

Метод GetOptionValue () интерфейса IStarUMLApplication возвращает значение опции, определяемой по идентификатору схемы и ключу опции, как вариант. Формат метода следующий.

```
IStarUMLApplication.GetOptionValue(SchemaID: String, Key: String): Variant
```

- SchemaID: идентификатор схемы, который определён в файле схемы опций.
- Key: ключ элемента опции, который определён в файле схемы опций.

Используйте значение типа Variant, возвращаемое GetOptionValue(), приводя его к типу каждого элемента опции. Вы можете читать это значение непосредственно без дополнительного приведения типа на языках скриптов типа JScript и VBScript.

Следующий пример читает значение опции "UNDO_LEVEL", определенное в схеме опций окружения StarUML и выводит его в окно сообщений.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var undoLevel = app.GetOptionValue("ENVIRONMENT", "UNDO_LEVEL");
WScript.Echo("Max. number of undo actions : " + undoLevel);
```

Обработка события изменения значения опции

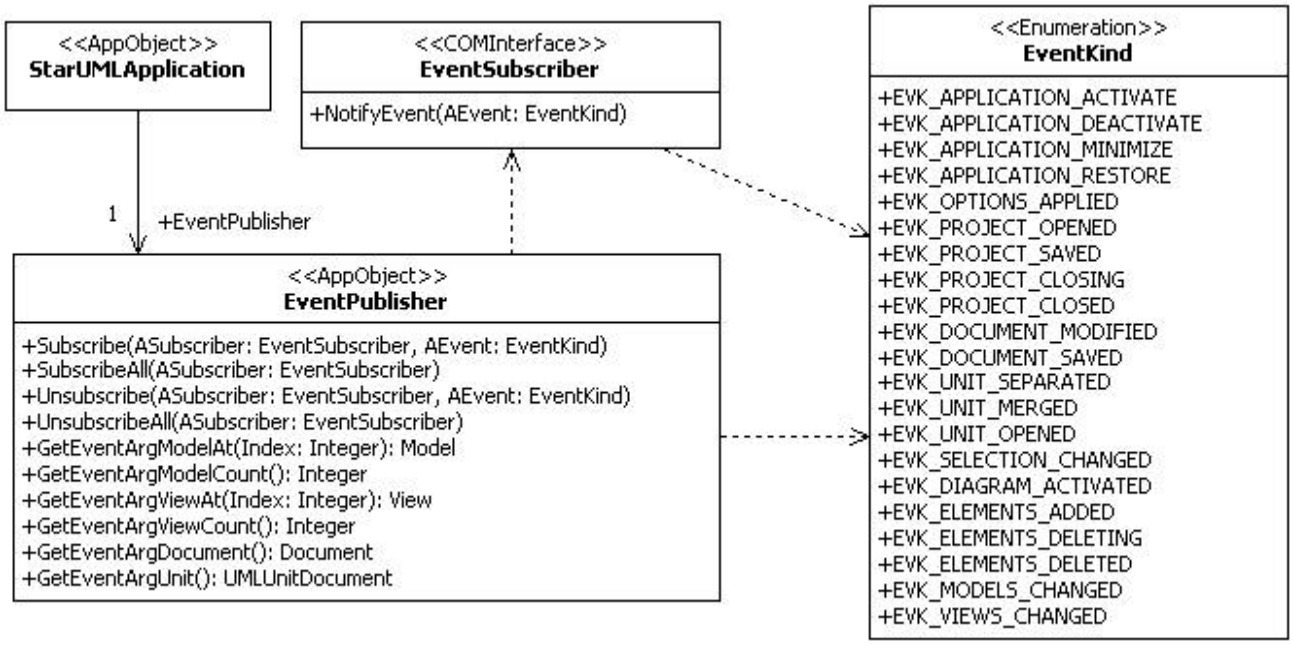
StarUML рассылает возникающие в его среде события аддинам, реализующим интерфейс IEventSubscriber. Если пользователь изменил значение опции в диалоге опций, приложение вызывает обработчик события - NotifyEvent() - тех аддинов, которые реализуют IEventSubscriber. Если Вы хотите контролировать значения опций в аддине сразу после их изменения пользователем, реализуйте интерфейс IEventSubscriber и метод NotifiyEvent(), чтобы читать значения опций методом IStarUMLApplication.GetOptionValue() в случае возникновения события EVK_OPTIONS_APPLIED. Аддины, использующие скрипт типа VBScript и JSCript, не могут контролировать изменение значений опций, так как они не могут реализовывать интерфейс IEventSubscriber.

Подробности, касающиеся обработки событий, будут изложены в следующем разделе.

Основные концепции обработки событий

Аддин, который реализовывает интерфейс IEventSubscriber, может подписываться на различные внутренние события приложения StarUML™. Всякий раз, когда внутреннее событие происходит, StarUML™ вызывает метод NotifyEvent зарегистрированных объектов типа IEventSubscriber.

Диаграмма классов, приведённая ниже, иллюстрирует организацию внешних интерфейсов API, связанных с обработкой событий.



Виды событий

Как показано выше, нумератор EventKind определяет виды внутренних событий StarUML™, на которые могут подписываться аддины, реализующие интерфейс IEventSubscriber. Таблица ниже описывает каждый литерал нумератора EventKind.

Вид события (литерал)	Целое значние	Описание события
EVK_APPLICATION_ACTIVATE	0	Возникает при активации окна приложения StarUML™ .
EVK_APPLICATION_DEACTIVATE	1	Возникает при деактивации окна приложения StarUML™.
EVK_APPLICATION_MINIMIZE	2	Возникает при минимизации окна приложения StarUML™.
EVK_APPLICATION_RESTORE	3	Возникает при восстановлении окна приложения StarUML™.
EVK_OPTIONS_APPLIED	4	Возникает, когда значение опции изменено
EVK_PROJECT_OPENED	5	Возникает, когда проект создан или открыт файл проекта.
EVK_PROJECT_SAVED	6	Возникает при сохранении проекта
EVK_PROJECT_CLOSING	7	Возникает, когда выбрано "Close Project".
EVK_PROJECT_CLOSED	8	Возникает, когда проект закрыт.
EVK_DOCUMENT_MODIFIED	9	Возникает, когда документ (проект или секция) изменён.
EVK_DOCUMENT_SAVED	10	Возникает, когда документ (проект или секция) сохранён.

Вид события (литерал)	Целое значние	Описание события
EVK_UNIT_SEPARATED	11	Возникает, когда секция выделена.
EVK_UNIT_MERGED	12	Возникает, когда секция присоединена.
EVK_UNIT_OPENED	13	Возникает, когда секция открыта.
EVK_SELECTION_CHANGED	14	Возникает, когда выбран другой элемент моделирования.
EVK_DIAGRAM_ACTIVATED	15	Возникает, когда диаграмма открывается.
EVK_ELEMENTS_ADDED	16	Возникает, когда новый модельный элемент создан.
EVK_ELEMENTS_DELETING	17	Возникает, когда модельный элемент удляется.
EVK_ELEMENTS_DELETED	18	Возникает, когда модельный элемент удалён.
EVK_MODELS_CHANGED	19	Возникает, когда изменено свойство модельного элемента
EVK_VIEWS_CHANGED	20	Возникает, когда изменяется свойство представления модельного элемента.

Подписка на события

Аддин, чтобы подписаться на события, должен реализовать интерфейс IEventSubscriber в дополнение к интерфейсу IStarUMLAddIn, который является общим интерфейсом для всех аддинов.

Следующий пример показывает определение класса аддина, который реализовывает интерфейсы IStarUMLAddIn и IEventSubscriber. Этот пример написан на Delphi.

```
type
    AddInExample = class(TComObject, IStarUMLAddIn, IEventSubscriber)
    private
        StarUMLApp: IStarUMLApplication;
        EventPub: IEventPublisher;
    protected
        function InitializeAddIn: HResult; stdcall;
        function FinalizeAddIn: HResult; stdcall;
        function DoMenuAction(ActionID: Integer): HResult; stdcall;
        function NotifyEvent(AEvent: EventKind): HResult; stdcall;
        ...
    public
        procedure Initialize; override;
        destructor Destroy; override;
        ...
    end;
```

Регистрация и удаление обработки событий

Чтобы аддин, который реализовывает интерфейс IEventSubscriber, мог подписаться на события, должен быть зарегистрирован обработчик событий . Регистрация обработчика событий и его удаление могут быть сделаны через объект типа IEventPublisher. Ссылка на объект типа IEventPublisher может быть получена через элемент IStarUMLApplication. Следующий пример на Delphi показывает, как получить ссылку на IStarUMLApplication и объект IEventPublisher.

```
implementation

procedure AddInExample.Initialize;
begin
    inherited;
    StarUMLApp := CreateOleObject('StarUML.StarUMLApplication') as IStarUMLApplication;
    EventPub := StarUMLApp.EventPublisher;
end;

destructor AddInExample.Destroy;
```

```
begin
    EventPub := nil;
    StarUMLApp := nil;
    inherited;
end;
```

Интерфейс IEventPublisher предоставляет следующие методы для регистрации и удаления обработчиков событий. Параметр "ASubscriber" для каждого метода представляет факический объект аддина, который реализовывает интерфейс IEventSubscriber.

Метод	Описание
Subscribe(ASubscriber: IEventSubscriber; AEvent: EventKind)	Регистрирует обработчик события, указанного аргументом AEvent.
SubscribeAll(ASubscriber: IEventSubscriber)	Регистрирует обработчик всех событий.
Unsubscribe(ASubscriber: IEventSubscriber; AEvent: EventKind)	Деактивирует обработчик события, указанного аргументом AEvent.
UnsubscribeAll(ASubscriber: IEventSubscriber)	Деактивирует обработку сех событий.

Используйте метод Subscribe, если аддин должен подписаться только на некоторые события. Например, для того, чтобы подписаться на два специфических события, вызовите метод Subscribe для каждого события. Используйте метод SubscribeAll, чтобы подписаться на все события. Вообще, методы Subscribe и SubscribeAll вызывает метод IStarUMLAddIn.InitializeAddIn.

Если аддину больше не нужно реагировать на события (например, когда объект уничтожен), все зарегистрированные события должны быть разрегистрированы. Использование метод Unsubscribe, если обработчик был зарегистрирован методом Subscribe, и метод UnsubscribeAll , если подписка была зарегистрирована методом SubscribeAll. Вообще, методы UnSubscribe и UnSubscribeAll вызывает метод IStarUMLAddIn.FinalizeAddIn.

Следующий пример показывает регистрацию и удаление подписки на события EVK_ELEMENTS_ADDED и EVK_ELEMENTS_DELETED.

```
implementation

function AddInExample.InitializeAddIn: HResult;
begin
    EventPub.Subscribe(Self, EVK_ELEMENTS_ADDED);
    EventPub.Subscribe(Self, EVK_ELEMENTS_DELETED);
    ...
    Result := S_OK;
end;

function AddInExample.FinalizeAddIn: HResult;
begin
    EventPub.Unsubscribe(Self, EVK_ELEMENTS_ADDED);
    EventPub.Unsubscribe(Self, EVK_ELEMENTS_DELETED);
    ...
    Result := S_OK;
end;
```

Доступ к параметрам события

Когда событие возникает, необходимо получить связанные с ним параметры. Например, когда происходит событие EVK_ELEMENTS_ADDED, связанное с созданием элемента моделирования, необходимо выяснить, какой элемент был создан. Интерфейс IEventPublisher предоставляет следующие методы получения параметров события.

Метод	Описание
GetEventArgModelCount(): Integer	Возвращает количество модельных элементов, относящихся к событию.
GetEventArgModelAt(Index: Integer): IModel	Возвращает по индексу ссылку на модельный элемент, относящийся к событию.
GetEventArgViewCount: Integer	Возвращает количество визуальных представлений, относящихся к событию.
GetEventArgViewAt(Index: Integer): IView	Возвращает по индексу ссылку на визуальное представление, относящееся к событию.
GetEventArgDocument: IDocument	Возвращает ссылку на элемент документа, относящегося к событию.
GetEventArgUnit: IUMLUnitDocument	Возвращает ссылку на документ секции, относящийся к событию.

Обработка событий

Когда происходит событие, на которое подписан аддин, он должен выполнить соответствующую обработку. Всякий раз, когда подписанное событие происходит, приложение StarUML™ вызывает метод NotifyEvent соответствующего Аддина и передает вид события как параметр. Вид события передаётся потому, что аддин может подписаться больше чем на одно событие. Каждый аддин должен реализовать в методе NotifyEvent логику запуска различных процессов в зависимости от вида события.

Следующий пример показывает реализацию метода NotifyEvent. Этот пример проверяет семантическую допустимость связей элемента, когда в приложении создаются элементы ассоциации (UMLAssociation) или обобщения (UMLGeneralization). (Этот пример - продолжение примеров, приведённых выше. Объявление аддина выполнено в примерах выше.)

```
implementation

function AddInExample.NotifyEvent(AEvent: EventKind): HResult;
var
    M: IModel;
    Assoc: IUMLAssociation;
    Gen: IUMLGeneralization;
    End1, End2: IUMLClassifier;
begin
    if AEvent = EVK_ELEMENTS_ADDED then
    begin
        if EventPub.GetEventArgModelCount = 1 then
        begin
            M := EventPub.GetEventArgModelAt(0);

            // Association
            if M.QueryInterface(IUMLAssociation, Assoc) = S_OK then
            begin
                End1 := Assoc.GetConnectionAt(0).Participant;
                End2 := Assoc.GetConnectionAt(1).Participant
                if End1.IsKindOf('UMLPackage') or End2.IsKindOf('UMLPackage') then
                    ShowMessage('Packages cannot have associations.')
                ...
            end;

          // Generalization
          if M.QueryInterface(IUMLGeneralization, Gen) = S_OK then
          begin
              if Gen.Child.IsRoot then
                  ShowMessage('Root elements cannot have parent elements.');
              if Gen.Parent.IsLeaf then
                  ShowMessage('Leaf elements cannot have child elements.');
           end;
        end;
    end;
```

```
    Result := S_OK;
end;
```

Глава 10. Расширение нотации

Эта глава является введением в технологию расширения графической нотации. Она представляет основные концепции расширения нотации и основные спецификации синтаксиса языка расширения нотации. Например, она показывает, как добавить новый вид диаграмм, в качестве примера возможного использования расширения нотации в своих интересах.

Зачем расширять нотацию?

Расширение нотации - механизм, предоставляющий пользователю возможность определять и использовать собственную графическую нотацию в моделях UML. StarUML поддерживает специальную платформу, позволяющую использовать возможность расширения нотации. Итак, зачем необходимо расширение нотации?

- Профиль поддерживает иконные и декоративные графические представления модельных элементов, но они не всегда могут точно реализовать графическую форму, требуемую для нотации.
- Для того, чтобы отобразить ER-диаграмму в UML, нужно корректно отобразить ER модель в модель UML, но отображение её нотации в нотацию UML невозможно.
- Мета модели UML достаточно, чтобы описать все виды контейнеров данных и их семантику. Если инструмент UML может свободно расширить свою нотацию, он может играть роль средства мета-моделирования во всех областях моделирования.

Использование старой графической нотации (формы) совместно со спецификациями UML дает пользователям дополнительную эффективность и совместимость между старой областью моделирования и UML.

Язык расширения нотации

Основной синтаксис

Синтаксис языка расширения нотации имитирует язык Scheme (диалект языка обработки списков LISP). Основные единицы - выражение и операция. Выражение состоит из значений и операций. Значения могут быть вещественные, целочисленные, строковые, булевые, идентификаторы. Операция начинается с "(" и заканчивается ")". Оператор и операнды (они представлены другими выражениями) заключаются в скобки. Операторы и идентификаторы регистро-независимы. Стиль комментариев аналогичен C++ и Java. Однострочные комментарии начинаются с "//", а многострочные заключаются в "/* */".

```
expr ::= flt | int | str | bool | nil | ident | "(" oper (expr)* ")" ;
```

Первая инструкция языка расширения нотации - выражение "notation". Оператором является "notation", а аргументами - выражения "onarrange" и "ondraw ". Выражение "notation" соответствует конкретному стереотипу в профиле. Выражение "notation" описывает, как будет показана фигура стереотипа. Изображение стереотипа - результат вычисления этого выражения. Сначала, вычисляется выражение "onarrange", выполняющее расчёт позиции элемента на основании полученных аргументов. А затем выполняется выражение "ondraw", рисующее элемент.

```
(notation   (onarrange ...)
    (ondraw ...)
)
```

Ниже перечислены доступные аргументы для выражений "onarrange" и "ondraw ".

- sequence
- if
- for
- set
- логический оператор сравнения
- встроенная функция

Выражение последовательности

Группирует "последовательность" выражений и выполняет аргументы в указанном порядке. Аргументы выражения "sequence" - такие же выражения, и их число не ограничено.

```
(sequence expr1 expr2 ...)
```

Следующий пример показывает одно выражение "sequence", группирующее 3 выражения.

```
(sequence
  (+ 10 20)       // 10 + 20
  (- 20 30 40)    // 20 - 30 - 40
  (/ 10 20)       // 10 / 20
)
```

Выражение if

Выражение "if" представляет условный синтаксис. Первый аргумент - условие, второй аргумент выполняется, если условие истинно, а третий аргумент выполняется, если условие ложно. Третий аргумент не обязателен. Если третий аргумент опущен, и условие является ложным, выражение "if" не выполняет ничего.

```
(if condition-expr on-true-expr on-false-expr? )
```

Следующий пример показывает, что, если значение "i" находится между 0 и 30, то выполнится выражение, увеличивающее переменную "count" на 1, в противном случае переменная "count" будет уменьшена.

```
(if (or (<= i 0) (>= i 30))   // if (i <= 0 || i >= 30)
   (set count (+ count 1))    //   count++;
   (set count (- count 1))    // else
)                             //   count--;
```

Выражение for

Выражение "for" повторяет некоторое выражение пока указанная переменная не переберёт все значения от начального до конечного. Первый аргумент - имя переменной, которая используется для подсчёта повторений. Второй - начальное значение переменной, а третий - её конечное значение. Последний аргумент - выражение, которое будет выполнено на каждой итерации.

```
(for identifier init-expr end-expr expr)
```

Следующий пример выводит на экран значения, изменяя i от 1 до 10.

```
(for i 1 10                              // for (int i = 1; i <= 10; i++)
    (textout 100 (+ 100 (* i 20)) i      //     textout(100, 100+(i*20), i);
             )
)
```

Выражение Set

Выражение "Set" присваивает значение переменной. Объявление переменной не требуется. Она объявляется автоматически и интерпретируется как глобальная переменная.

```
(set identifier value-expr)
```

Следующий пример присваивает значения переменным a и b, конкатенирует их, и присваивает результат переменной c.

```
(set a 'My name is ')        // a = "My name is ";
(set b 'foo')                // b = "foo";
(set c (concat a b))         // c = a + b;arthmetic, logical, comparison operator
```

Операторы: арифметические, логические, сравнения

Поддерживаемые операторы

- математические - "+", "-", "*", "/"
- логические операторы - "and", "or", "not".
- операторы сравнения - "=", "! = ", "<", " <= ", ">", "> = " .

```
(+ 1 (/ 10 5) (- (* 2 3) 6))      // 1 + (10/5) + (2*3-6)
(and (< i 10) (not (= j 20)))     // (i < 10) && (!(j == 20))
```

Встроенные функции

Встроенные функции, поддерживаемые в языке расширения нотации, сгруппированы следующим образом:

- Математические функции
- Строковые функции
- Функции списков
- Функции доступа к модели
- Графические функции

Математические функции

Ниже представлен список встроенных функций, связанных с математикой.

Сигнатура	Описание
(sin angle)	возвращает синус угла.
(cos angle)	возвращает косинус угла.
(tan angle)	возвращает танкгенс угла.

Сигнатура	Описание
(trunc val)	обрезает вещественное значение до целого. val - выражение вещественного типа.
(round val)	Округляет val до ближайшего целого. Если val лежит точно посередине между ближайшими целыми, результатом является чётноечисло.

Строковые функции

Ниже представлен список встроенных функций, связанных со строковой обработкой.

Сигнатура	Описание
(concat str1 str2...)	соединяет все аргументы в одну строку.
(trim str)	удаляет все начальные и конечные управляющие символы из строки.
(length str)	возвращает количество символов в строке
(tokenize str deli)	возвращает список строк, полученных разбиением исходной строки по указанному разделителю

Функции списков

Ниже представлен список встроенных функций, связанных с обработкой списков.

Сигнатура	Описание
(list val1 val2 ...)	возвращает список, составленный из аргументов.
(append lst lst)	присоединяет элемент к концу списка.
(append lst item)	
(itemat lst index)	возвращает элемент списка по индексу
(itemcount lst)	возвращает количество элементов в списке.

Функции доступа к модели

Ниже представлен список встроенных функций, связанных с доступом к модели.

Сигнатура	Описание
(mofattr elem attr)	возвращает значение указанного строкового свойства указанного атрибута.
(mofsetattr elem attr val)	присваивает значение "val" атрибуту "attr" модельного элемента.
(mofref elem ref)	возвращает значение свойства типа ссылки (object reference) для элемента, указанного аргументом.
(mofcolat elem col at)	возвращает значение типа ссылки (object reference) являющееся элементом с индексом "at" коллекции-свойства модельного элемента.
(mofcolcount elem col)	возвращает количество элементов в коллекции, определённой аргументами.
(constraintval elem name)	возвращает констрэйнт элемента.
(tagval elem tagset name)	возвращает значение тега примитивного типа указанного набора.
(tagref elem tagset name)	возвращает значение тега типа ссылки из указанного набора.
(tagcolat elem tagset name at)	возвращает указанный элемент тега-коллекции.
(tagcolcount elem tagset name)	возвращает количество элементов в теге коллекции

Графические функции

Ниже представлен список встроенных функций, связанных со стилями.

<table>
<tr><th>Сигнатура</th><th>Описание</th></tr>
<tr><td>(setpencolor color)</td><td>устанавливает цвет рисования линий окаймления фигур. Данный цвет используется пером, способ использования зависит от значений свойств Mode и Style.

Color может принимать следующие значения:
<table>
<tr><th>Значение</th><th>Смысл</th></tr>
<tr><td>clNone</td><td>Отсутствующий (белый)</td></tr>
<tr><td>clAqua</td><td>Цвет морской волны</td></tr>
<tr><td>clBlack</td><td>Чёрнй</td></tr>
<tr><td>clBlue</td><td>Глубой</td></tr>
<tr><td>clCream</td><td>Кремовый</td></tr>
<tr><td>clDkGray</td><td>Тёмно-серый</td></tr>
<tr><td>clFuchsia</td><td>Фуксия</td></tr>
<tr><td>clGray</td><td>Серый</td></tr>
<tr><td>clGreen</td><td>Зелёный</td></tr>
<tr><td>clLime</td><td>Известковый</td></tr>
<tr><td>clLtGray</td><td>Светло-серый</td></tr>
<tr><td>clMaroon</td><td>Красно-коричневый</td></tr>
<tr><td>clMedGray</td><td>Средне-серый</td></tr>
<tr><td>clMoneyGreen</td><td>Светло-зелёный</td></tr>
<tr><td>clNavy</td><td>Темно-синий</td></tr>
<tr><td>clOlive</td><td>Оливковый</td></tr>
<tr><td>clPurple</td><td>Пурпурный</td></tr>
<tr><td>clRed</td><td>Красный</td></tr>
<tr><td>clSilver</td><td>Серебряный</td></tr>
<tr><td>clSkyBlue</td><td>Небесно-голубой</td></tr>
<tr><td>clTeal</td><td>Теал</td></tr>
<tr><td>clWhite</td><td>Белый</td></tr>
<tr><td>clYellow</td><td>Жёлтый</td></tr>
</table></td></tr>
<tr><td>(setpenstyle style)</td><td>Используется при рисовке пунктирных или разорванных линий, или для отмены рисования линий, окаймляющих фигуры.

Style может принимать следующие значения:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>psSolid</td><td>сплошная линия.</td></tr>
<tr><td>psDash</td><td>пунктирная линия.</td></tr>
<tr><td>psDot</td><td>точечная линия.</td></tr>
<tr><td>psDashDot</td><td>точечно-пунктирная линия.</td></tr>
<tr><td>psDashDotDot</td><td>точечно-точечно-пунктирная линия.</td></tr>
<tr><td>psClear</td><td>Не рисовать линии (используется для отмены прорисовки окаймления фигур).</td></tr>
<tr><td>psInsideFrame</td><td>сплошная линия, не может менять цвет, если ширина линии больше 1.</td></tr>
</table></td></tr>
<tr><td>(setbrushcol</td><td>Устанавливает цвет кисти (из указанного ранее набора цветов).</td></tr>
</table>

Сигнатура	Описание
or color)	
(setbrushstyle style)	bsSolid, bsClear, bsHorizontal, bsVertical, bsFDiagonal, bsBDiagonal, bsCross, bsDiagCross
(setfontface font)	устанавливает гарнитуру шрифта.
(setfontcolor color)	устанавливает цвет шрифта (один из указанных ранее)
(setfontsize size)	устанавливает размер шрифта.
(setfontstyle style)	устанавливает стиль шрифта. Style формируется из указанных констант, разделённых символом "|".

Value	Meaning
fsBold	жирный.
fsItalic	курсив.
fsUnderline	подчёркнутый.
fsStrikeOut	зачёркнутый

(setdefaultstyle)	Возвращает свойствам Pen, Brush, Font значения по умолчанию

Ниже представлен список встроенных функций, связанных с графикой.

Сигнатура	Описание
(textheight str)	Возвращает высоту строки в пикселях, в зависимости от текущего шрифта.
(textwidth str)	Возвращает ширину строки в пикселях, в зависимости от текущего шрифта.
(textout x y str)	Отображает строку на экране с позиции (X,Y).
(textbound x1 y1 x2 y2 yspace text clipping)	Размещает строку в области экрана (x1, y1) - (x2, y2). yspace - размер строки. Если clipping установлен в "true", строка ограничивается областью вывода.
(textrect x1 y1 x2 y2 x y str)	Выводит строку в область экрана (x1, y1) - (x2, y2) от точки (X,Y).

Ниже представлен список встроенных функций, связанных с графическими фигурами.

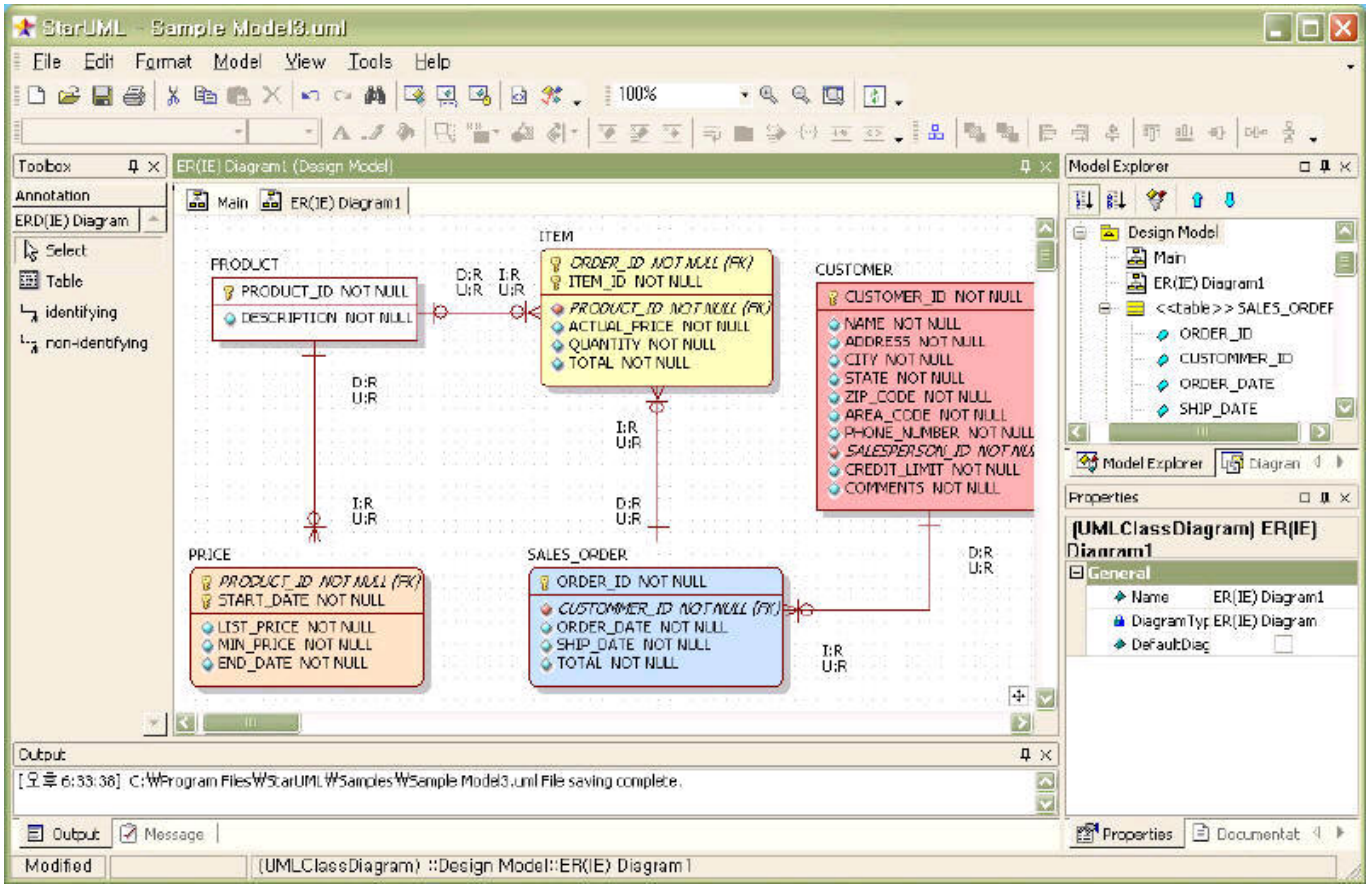
Signature	Описание
(rect x1 y1 x2 y2)	Рисует прямоугольник в области (X1,Y1) и (X2,Y2).
(filerect x1 y1 x2 y2)	Закрашивает четырёхугольник на канве согласно текущим атрибутам кисти.
(ellipse x1 y1 x2 y2)	Рисует эллипс на экране в указанной прямоугольной области.
(roundrect x1 y1 x2 y2 x3 y3)	Рисует прямоугольник с закруглёнными углами.
(arc x1 y1 x2 y2 x3 y3 x4 y4)	Рисует дугу внутри эллипса, ограниченного областью (X1,Y1) и (X2,Y2). Дуга начинается от точки пересечения эллипса с линией проходящей между центром эллипса ((X1+X2)/2.0, (Y1+Y2)/2.0) и точкой (X3,Y3) и рисуется против часовой стрелки до точки пересечения эллипса с линией, проходящей между центром эллипса и точкой (X4,Y4).
(pie x1 y1 x2 y2 x3 y3 x4 y4)	Рисует изображение выпуклого сектора. Сектор определяется эллипсом, ограниченным прямоугольником, определяемым точками (X1, Y1) и (X2, Y2). Секция прорисовки определяется

<table>
<tr><th>Signature</th><th>Описание</th></tr>
<tr><td></td><td>двумя радиальными линиями, проходящими между центром эллипса и точками (X3, Y3) и (X4, Y4) соответственно.</td></tr>
<tr><td>(drawbitmap x y img transparent)
(drawbitmap x1 y1 x2 y2 img transparent)</td><td>Вставляет изображение, определённое параметрами, в позицию экрана, определяемую координатами (X, Y). Используйте аргумент transparent чтобы изображение было прозрачным. Используйте аргументы x2, y2 чтобы растянуть изображение.</td></tr>
<tr><td>(moveto x y)</td><td>Изменяет текущую позицию рисования в точку (X,Y).</td></tr>
<tr><td>(lineto x y)</td><td>Рисует линию на канве от текущей позиции пера, в точку указанную X и Y, и устанавливает позицию пера в (X, Y).</td></tr>
<tr><td>(line x1 y1 x2 y2)</td><td>Рисует линию на канве от точки (x1, y1) в точку (x2, y2).</td></tr>
<tr><td>(pt x y)</td><td>Возвращает структуру точки по паре координат.</td></tr>
<tr><td>(polygon (pt x1 y1) (pt x2 y2) ...)</td><td>Рисует серию линий на канве, соединённых в указанных точках. Фигура замыкается линией от последней точки к первой.</td></tr>
<tr><td>(polyline (pt x1 y1) (pt x2 y2) ...)</td><td>Рисует серию линий на канве, начиная с текущей точки через указанные точки.</td></tr>
<tr><td>(polybezier (pt x1 y1) (pt x2 y2) ...)</td><td>рисует набор кривых Безье.</td></tr>
<tr><td>(ptatx index)</td><td>Допустима, когда текущий образ имеет "край". Возвращает значение "x" структуры точки по индексу краевой точки.</td></tr>
<tr><td>(ptaty index)</td><td>Допустима, когда текущий образ имеет "край". Возвращает значение "y" структуры точки по индексу краевой точки.</td></tr>
<tr><td>(ptcount)</td><td>Допустима, когда текущий образ имеет "край". Возвращает число краевых точек.</td></tr>
<tr><td>(drawedge headOrTail endStyle)</td><td>Допустима, кода текущий образ имеет "край". Рисует наконечник краевой точки в указанном стиле. Стиль задаётся как композиция следующих значений, разделённых символом "|".
<table>
<tr><th>Value</th><th>Shape</th></tr>
<tr><td>esStickArrow</td><td>  </td></tr>
<tr><td>esSolidArrow</td><td>  </td></tr>
<tr><td>esTriangle</td><td>  </td></tr>
<tr><td>esDiamond</td><td>  </td></tr>
<tr><td>esMiniDiamond</td><td>  </td></tr>
<tr><td>esArrowDiamond</td><td>  </td></tr>
<tr><td>esCrowFoot</td><td>  </td></tr>
<tr><td>esHalfStickArrow</td><td>  </td></tr>
<tr><td>esBar</td><td>  </td></tr>
<tr><td>esDoubleBar</td><td>  </td></tr>
<tr><td>esBelowCircle</td><td>  </td></tr>
<tr><td>esCircle</td><td>  </td></tr>
<tr><td>esRect</td><td>  </td></tr>
<tr><td>esFilledTriangle</td><td>  </td></tr>
<tr><td>esFilledDiamond</td><td>  </td></tr>
<tr><td>esMiniFilledDiamond</td><td>  </td></tr>
<tr><td>esArrowFilledDiamond</td><td>  </td></tr>
<tr><td>esFilledHalfStickArrow</td><td>  </td></tr>
<tr><td>esFilledCircle</td><td>  </td></tr>
</table></td></tr>
</table>

Signature	Описание
	esFilledRect 
	esMiniHalfDiamond 
(drawobject elem)	Прорисовывает элемент в оригинальном стиле.
(arrangeobject elem)	Устанавливает элемент в оригинальном стиле.

Создание нового типа диаграмм

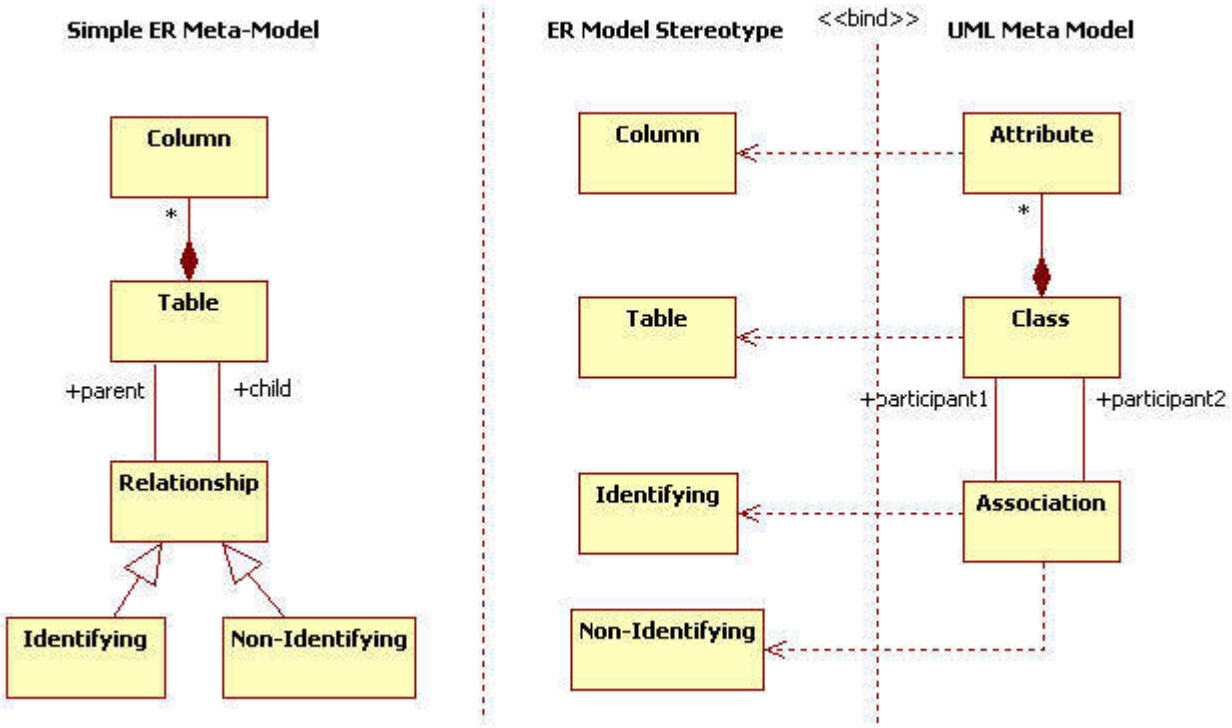
Существуют некоторые предварительные условия, которые нужно соблюсти, чтобы использовать расширение нотации. Прежде всего, необходим профиль. Он описывает, к какому стереотипу применяется расширение нотации. Во вторых, необходим файл расширения нотации (*.nxt). Он описывает, как нотация должна изображаться.



Вы должны подключить расширение нотации к стереотипу в профиле. Свойства, которые используются для расширения нотации, также должны быть указаны в тегах профиля. Ниже изложено, как описать нотацию ER-диаграммы, в качестве примера расширения нотации.

Определение профиля

Что касается ER-диаграммы, то она состоит из нескольких элементов (таблица, столбец, отношение, и т.д.).



Вы должны создать стереотипы для таблицы, столбца, отношений, и т.д. и применить эти стереотипы к модельным элементам UML (класс, ассоциация, атрибут), чтобы отобразить ER на UML. Это делается в профиле следующим образом. Вы добавляете XML-элемент <STEREOTYPE>, который называете "table", в <STEREOTYPELIST> и присваиваете элементу <BASECLASS> значение "UMLClass", т.е. применяете данный стереотип к типу "UMLClass". Чтобы класс со стереотипом "table" отображался как таблица в нотации ER, имя файла расширения нотации ("table.nxt") нужно указать в элементе .

Для стереотипа "column", необходимо определить дополнительные теги, чтобы указывать, является ли столбец ключевым (PK, FK, AK, или IK). Имя набора тегов ("table" в данном случае), в котором определён данный тег, указывается в элементе .

```
<PROFILE version="1.0">
    <HEADER>
        ...
    </HEADER>
    <BODY>
        <STEREOTYPELIST>
            <STEREOTYPE>
                <NAME>table</NAME>
                <BASECLASSES>
                    <BASECLASS>UMLClass</BASECLASS>
                </BASECLASSES>
                <NOTATION>table.nxt</NOTATION>
            </STEREOTYPE>
            <STEREOTYPE>
                <NAME>column</NAME>
                <BASECLASSES>
                    <BASECLASS>UMLAttribute</BASECLASS>
                    <RELATEDTAGDEFINITIONSET>table</RELATEDTAGDEFINITIONSET>
                </BASECLASSES>
            </STEREOTYPE>
            ...

        </STEREOTYPELIST>
```

Набор определений тегов описан в элементе <TAGDEFINITIONSET> элемента

<TAGDEFINITIONSETLIST>, а сам элемент <TAGDEFINITIONSET> составлен из элементов <TAGDEFINITION>, которые описывают свойства тега (имя, тип и значение по умолчанию), добавленного для стереотипа столбца. В следующем примере добавляются теги, идентифицирующие столбец как PK и FK, тип каждого тега является булевым, и значением по умолчанию для каждого является "ложь". (Это означает, что столбец сразу после создания не является ни первичным, ни внешним ключом).

```
        ...
        </STEREOTYPELIST>
        <TAGDEFINITIONSETLIST>
            <TAGDEFINITIONSET>
                <NAME>column</NAME>
                <BASECLASSES>
                    <BASECLASS>UMLAttribute</BASECLASS>
                </BASECLASSES>
                <TAGDEFINITIONLIST>
                    ...
                    <TAGDEFINITION lock="False">
                        <NAME>PK</NAME>
                        <TAGTYPE>Boolean</TAGTYPE>
                        <DEFAULTDATAVALUE>false</DEFAULTDATAVALUE>
                    </TAGDEFINITION>
                    <TAGDEFINITION lock="False">
                        <NAME>FK</NAME>
                        <TAGTYPE>Boolean</TAGTYPE>
                        <DEFAULTDATAVALUE>false</DEFAULTDATAVALUE>
                    </TAGDEFINITION>
                    ...
                </TAGDEFINITIONLIST>
            </TAGDEFINITIONSET>
        </TAGDEFINITIONSETLIST>
```

Чтобы создать диаграмму, которая показывает эти стереотипы после их определения, введите новый тип диаграмм, с именем "ER Diagram" в элементе <DIAGRAMTYPE> элемента <DIAGRAMTYPELIST>, указав значение элемента <BASEDIAGRAM> как "ClassDiagram", т.к. она будет основана на диаграмме классов, а также укажите в качестве ссылки на палитру "ERD (IE)" в элементе <AVALIABLEPALLETTE>.

```
        <DIAGRAMTYPELIST>
            <DIAGRAMTYPE>
                <NAME>ER(IE) Diagram</NAME>
                <DISPLAYNAME>ER(IE) Diagram</DISPLAYNAME>
                <BASEDIAGRAM>ClassDiagram</BASEDIAGRAM>
                <ICON>DataModelDiagram.bmp</ICON>
                <AVAILABLEPALETTELIST>
                    <AVAILABLEPALETTE>ERD(IE)</AVAILABLEPALETTE>
                </AVAILABLEPALETTELIST>
            </DIAGRAMTYPE>
        </DIAGRAMTYPELIST>
```

Информация о палитре указывается в элементе <PALLETTE>. Элемент <PALLETTE> - список ссылок на кнопки. Подробная информация о кнопках палитр приведена при описании элемента <ELEMENTPROTOTYPE>. Элемент <NAME> описывает название создаваемого элемента, элементы <DISPLAYNAME> и <ICON> описывают названия кнопок и имена файлов с изображениями на кнопках, элемент <DRAGTYPE> определяет стиль акций мыши, подобный прямоугольному или линейному, элементы <BASEELEMENT>, и <STEREOTYPENAME> подразумевают, что создаваемый элемент является "классом", которому присвоен стереотип "table". Чтобы элемент рисовался согласно расширению нотации, значение элемента <SHOWEXTENSION> должно быть установлено в "истину".

```
        <PALETTELIST>
            <PALETTE>
                <NAME>ERD(IE)</NAME>
                <DISPLAYNAME>ERD(IE) Diagram</DISPLAYNAME>
                <PALETTEITEMLIST>
```

```
                    <PALETTEITEM>Table</PALETTEITEM>
                    <PALETTEITEM>identifying</PALETTEITEM>
                    <PALETTEITEM>non-identifying</PALETTEITEM>
                </PALETTEITEMLIST>
            </PALETTE>
        </PALETTELIST>
        <ELEMENTPROTOTYPELIST>
            <ELEMENTPROTOTYPE>
                <NAME>Table</NAME>
                <DISPLAYNAME>Table</DISPLAYNAME>
                <ICON>Table.bmp</ICON>
                <DRAGTYPE>Rect</DRAGTYPE>
                <BASEELEMENT>Class</BASEELEMENT>
                <STEREOTYPENAME>table</STEREOTYPENAME>
                <SHOWEXTENDEDNOTATION>True</SHOWEXTENDEDNOTATION>
            </ELEMENTPROTOTYPE>
            ...
        <ELEMENTPROTOTYPELIST>
    ...
```

Написание расширения нотации

Хотя для создания спецификаций модельных элементов достаточно только определения профиля, однако, чтобы модель отображалась в нотации ER, должен быть написан файл расширения нотации (*.nxt), на который ссылается элемент <NOTATION> профиля. Следующие строки представляют скелет файла "table.nxt", они рисуют изображение стереотипа "table". Выражение "onarrange" конфигурирует состояние, требуемое чтобы рисовать "таблицу". Выражение "ondraw" рисует по частям имя таблицы, столбец PK и другие столбцы.

```
(notation
    (onarrange ...)

    (ondraw
        // draw  ...

        // draw PK column part ...

        // draw other column part ...
    )
)
```

Первая часть (name part) - устанавливает переменные для прорисовки и строку имени, полученную из модели и выводимую в позиции (x, y).

```
(set x left)
(set y top)
...
(set name (mofattr model 'Name'))
(textout x y name)
...
```

Здесь, переменные "left" и "top" - зарезервированные переменные. Они получают актуальные значения от StarUML при каждом обращении к расширению нотации, и могут возвращать значения в StarUML после завершения прорисовки. Все указанные ниже переменные, ведущие себя подобно этим, - зарезервированные переменные.

Переменная	Представление	Возвращается в StarUML	Описание
view	Node,Edge	not return	целевое представление для прорисовки
model	Node,Edge	not return	модельный элемент целевого представления для прорисовки
left	Node	return	левая крайняя позиция целевого представления

Переменная	Представление	Возвращается в StarUML	Описание
top	Node	return	верхняя крайняя позиция целевого представления
right	Node	return	правая крайняя позиция целевого представления
bottom	Node	return	нижняя крайняя позиция целевого представления
width	Node	return	ширина целевого представления
height	Node	return	высота целевого представления
minwidth	Node	not return	минимальная ширина целевого представления
minheight	Node	not return	минимальная высота целевого представления
points	Edge	not return	коллекция точек представления целевого края
head	Edge	not return	головной элемент представления целевого края
tail	Edge	not return	хвостовой элемент представления целевого края

Следующий код проверяет, зависит ли текущая таблица от других и рисует собственно фигуру таблицы. В цикле просматриваются ассоциации текущей таблицы (класса), если конец ассоциации соединяется с текущей таблицей, это означает, что таблица зависимая, и она прорисовывается как округленный прямоугольник. В противном случае, таблица рисуется как обычный прямоугольник, и это означает, что таблица независима от других.

```
(set isSuperType true)
(set c (mofcolcount model 'Associations'))
(for i 0 (- c 1)
    (sequence
        (set assocEnd (mofcolat model 'Associations' i))
        (if (= assocEnd (mofcolat (mofref assocEnd 'Association') 'Connections' 1))
            (set isSuperType false)
            nil)))
...
// outline
(setdefaultstyle)
(if isSuperType
    (rect x y right bottom)
    (roundrect x y right bottom 10 10))
```

При отображении столбцов, циклически перебираются все столбцы, которые содержит таблица. Элементы, значение тега PK у которых истинно, рисуются перед другими столбцами, иконка PK прорисовывается слева, а имя столбца прорисовывается с правой стороны.

```
...
(for i 0 (- (mofColCount model 'Attributes') 1)
    (sequence
        // select i-th column
        (set attr (mofColAt model 'Attributes' i))
        ...
        // column is PK?
        (if (tagVal attr 'ERD' 'column' 'PK')
            (sequence
                ...
                (set attrName (mofAttr attr 'Name'))
                ...
                (drawbitmap x y 'primarykey.bmp' true)
                (textout (+ x 16) y attrName)
                (setdefaultstyle)
                ... ))))
...
(line left y right y)
```

То же самое повторяется для всех столбцов. Элементы, значение тега PK у которых ложно, рисуются с иконкой обычной колонки под столбцами PK.

```
...
(for i 0 (- (mofColCount model 'Attributes') 1)
```

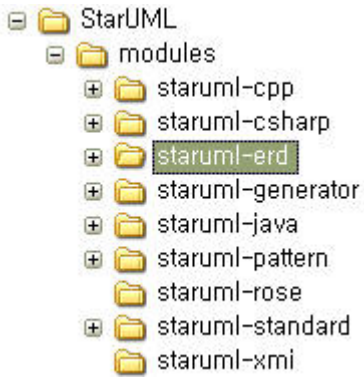
```
    (sequence
        // select i-th column
        (set attr (mofColAt model 'Attributes' i))
        (set keys '')
        ...
        // column is not PK?
        (if (= (tagVal attr 'ERD' 'column' 'PK') false)
        (sequence
            ...
            (set attrName (mofAttr attr 'Name'))
            ...
            // draw column
            (drawbitmap x y 'column.bmp' true))
            (textout (+ x 16) y attrName)
            (setdefaultstyle)
                ... ))))
```

Установка и использование расширения нотации

Файл расширения нотации должен располагаться согласно пути, указанному в профиле. В данном случае (при определении стереотипа "table"), так как путь не указан, а указано только имя файла, профиль и файл расширения нотации должны находиться в одной папке.

Если Вы сделали все, о чём говорилось выше, сделайте следующие шаги для инсталляции.

1. Создайте новую папку для модуля в каталоге модулей.
2. Поместите профиль, файл расширения нотации и связанные с ними файлы изображений в папку модуля.
3. Перезапустите StarUML.



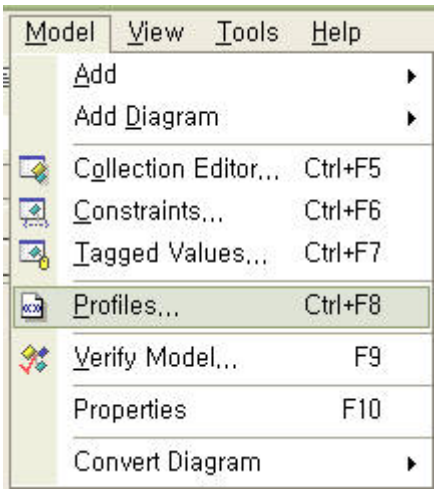
Ссылка

- Загрузите полный файл расширения нотации, профиль, и и всё остальное для поддержки

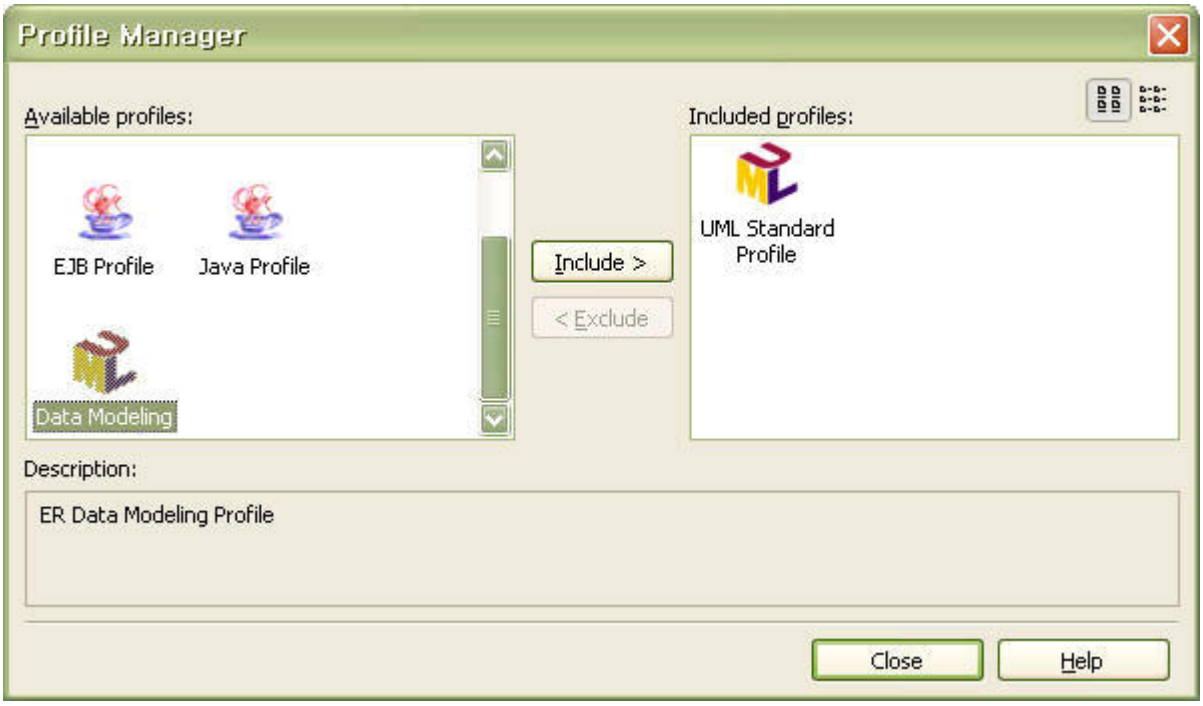
ER-диаграмм из раздела загрузки модулей официальной домашней страницы StarUML и установите всё согласно вышеуказанным шагам.

Ниже показано, как использовать расширение нотации.

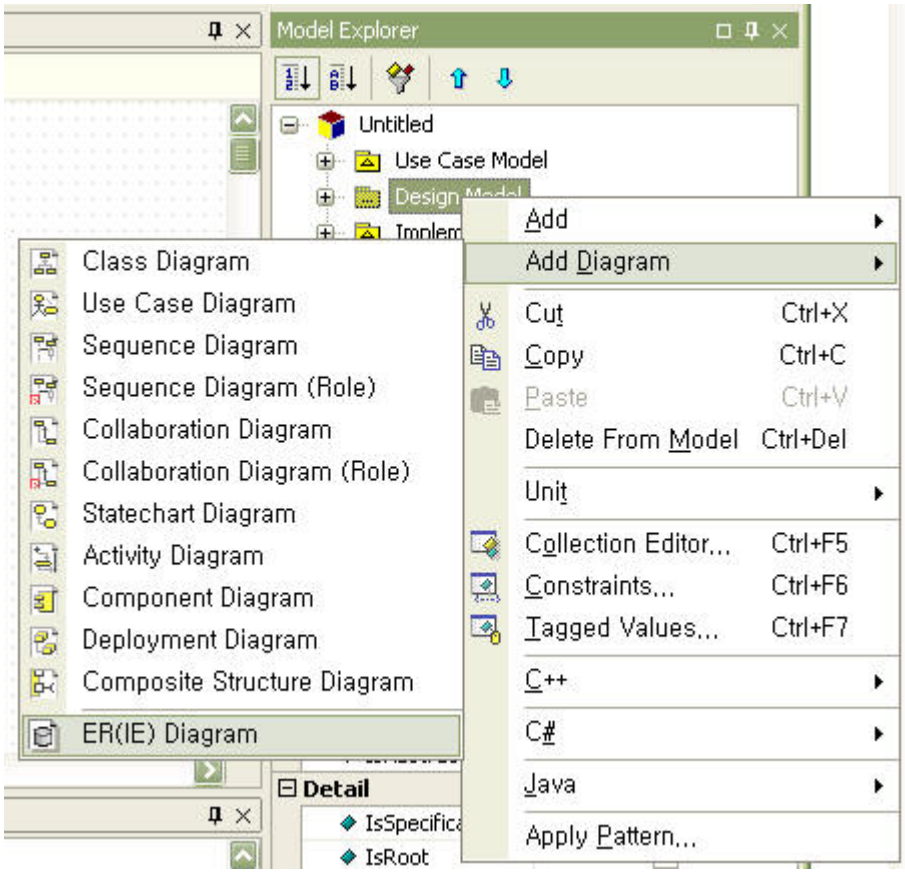
1. Запустите Start StarUML.
2. Щёлкните меню [Model] -> [Profiles...].



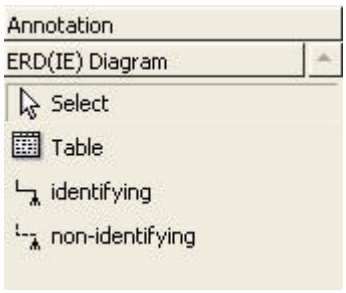
3. В диалоговом окне [Profile Manager] выберите профиль Data Modeling в списке [Available profiles] и нажмите кнопку [Include].



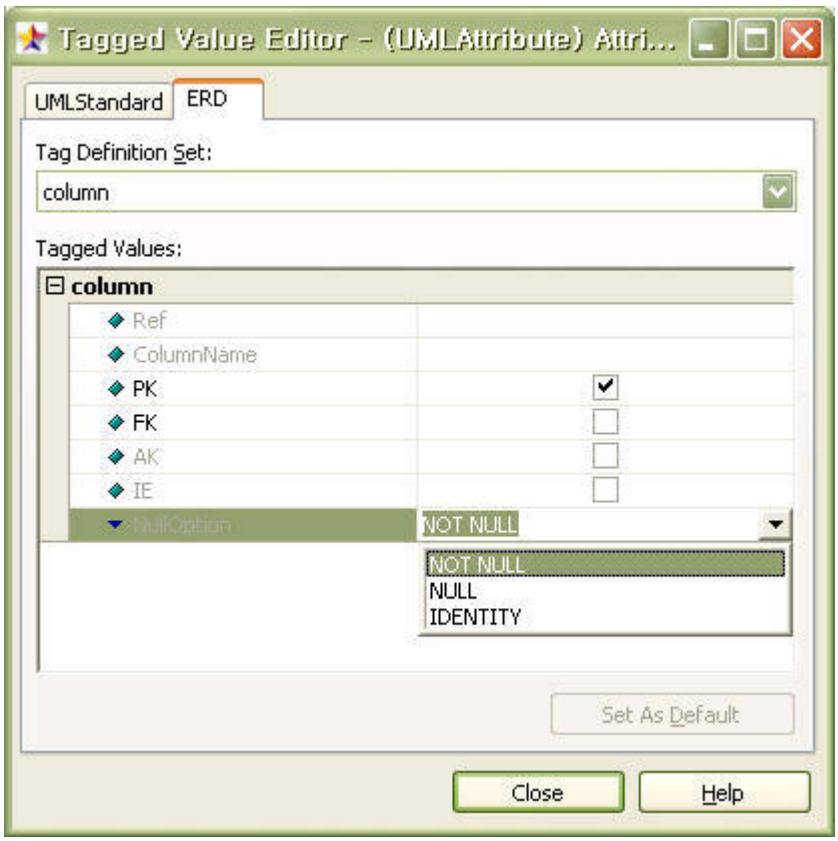
4. Выберите в [Model Explorer] пакет, в который нужно поместить ER-диаграмму , и нажмите всплывающее меню [Add Diagram] -> [ER(IE) Diagram] .



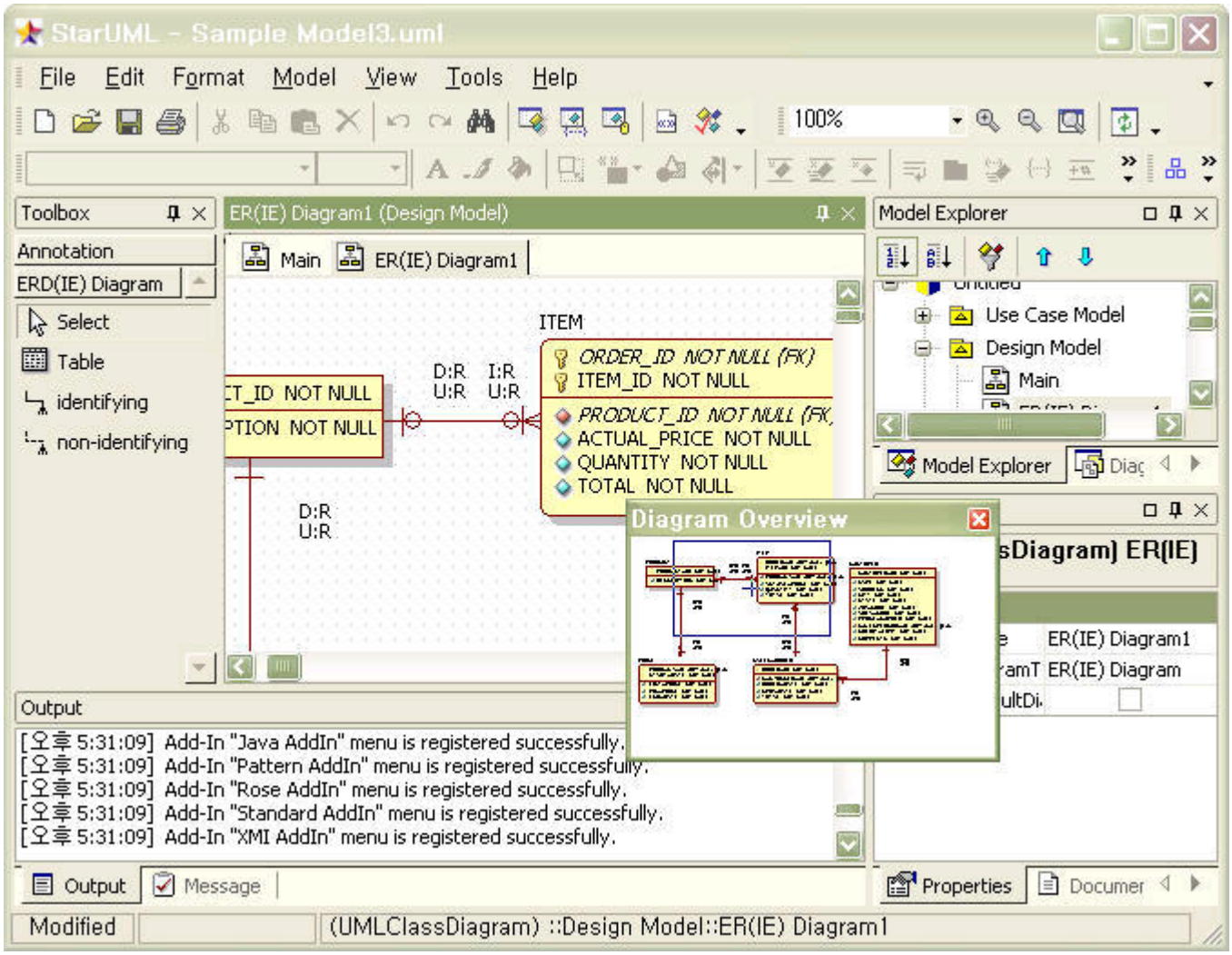
5. ER-диаграмма появится в окне [Main], а палитра моделирования для ER будет отображена в [инструментарии].



6. Используйте палитру для моделирования. Щелкните кнопку  и установите значения тегов на закладка [ERD] редактора тегов, чтобы определить свойства столбца.



7. Создавайте ER модель



Глава 11. Написание шаблонов

Эта глава представляет введение в композицию элементов, используемых для того, чтобы генерировать документы под Microsoft Word, Microsoft Excel, Microsoft PowerPoint, и текстовые файлы. Она показывает на примерах, как пользователь может создать, зарегистрировать и использовать собственный шаблон.

Компоненты шаблона

Шаблон StatUML включает текст двух типов. Фрагменты первого типа - область стиля, которая определяет стиль и формат документа, текст второго типа - область команд, определяющих, какие модельные элементы генератор получает из UML модели. Для представления команд шаблоны MS Office (Word, Excel, PowerPoint) используют комментарии MS Office, содержащие текстовый код определённого формата, заключённый в специальные символы. Область команд содержит команды типа циклов, сравнения, вычисления и прорисовки, обращающиеся к UML модели. Команды шаблонов под каждую платформу немного различаются, но в общем они подобны следующим.

- REPEAT ~ ENDREPEAT
- IF ~ ENDIF
- DISPLAY
- SCRIPT

Команда REPEAT

REPEAT - команда, которая выполняет перебор модельных элементов в соответствии со своими аргументами. Повторяется участок, расположенный между командами REPEAT И ENDREPEAT, генератор вставляет этот фрагмент в генерируемый документ на каждой итерации. Команда REPEAT имеет следующие четыре параметра.

Аргумент	Описание	Замечание
Pathname	Путь к перебираемым элементам	Необязательный
FilterType	Перебираются элементы типа FilterType.	Необязательный
CollectionName	Перебираются элементы коллекции CollectionName элементов, расположенных в Pathname и удовлетворяющих FilterType.	Необязательный
Condition	Перебираются элементы, удовлетворяющие Condition.	Необязательный

Первый параметр "Pathname" определяет отправную точку итерирования в модели UML. Он имеет форму имени пути, с разделителями "::". Имя пути элемента показывается в строке состояния. Есть два вида Pathname (абсолютный и относительный). Абсолютный путь начинается с символов "::". Например, "::A" означает элемент с именем "A", находящийся на верхнем уровне проекта. "A" означает элемент с именем "A", находящийся в контексте текущего элемента. Строка "{R} " означает, что итерации выполняются рекурсивно для всех суб-путей указанного пути. Если имя пути опущено, перебираются элементы по пути, указанному последней командой.

Второй параметр "FilterType" определяет тип перебираемых элементов. Если значение параметра - "UMLClass", то выполняется перебор только тех элементов, тип которых - "UMLClass". Если параметр опущен, то выполняется перебор всех элементов, независимо от типа.

Третий параметр "CollectionName" означает, что выполняется перебор элементов в коллекции выбранного элемента, которая имеет имя CollectionName. Например, если первый параметр - "::A", второй параметр - "UMLClass", и третий параметр - "OwnedElements", то это означает, что выполняется перебор элементов коллекции "OwnedElements" типа "UMLClass", располагающихся в контексте элемента "::A".

Четвертый параметр "Condition" определяет условие, которому должны удовлетворять перебираемые элементы.

Если значение параметра - "current().StereotypeName == 'boundary'", то выполняется перебор элементов, которые имеют стереотип "boundary". Значение по умолчанию для этого параметра - истина. Если параметр опущен, перебирются все элементы.

Ссылка

- current() - встроенная функция, которая используется в генераторе. См.ниже "Компоненты шаблона. Встроенные функции".

Вариации команды в шаблоне для WORD

Не только ENDREPEAT но также и инструкция ENDREPTR может закрывать команду REPEAT. Команда REPEAT с ENDREPTR используется для добавления строк таблиц. Например, чтобы поместить в таблицу список классов, поместите команду REPEAT в первой ячейке строки, а команду ENDREPTR поместите в последнюю ячейку строки. Это приведёт с созданию строки таблицы под каждый перебираемый элемент.

Команда IF

В случае удовлетворения указанного условия, команда IF отображает фрагмент, расположенный между командами IF и ENDIF. Команда IF имеет следующие параметры. Значения параметров выражаются как на JScript.

Аргумент	Описание	Замечания
Condition	контрольное условие	Обязательный

Ссылка

- Команда IF не доступна в шаблонах Excel и Powerpoint (но будет реализована в будущем)

Вариации команды в шаблоне WORD

- Здесь существует вариация команды IF "IF.. ENDIFTR". Она показывает строку таблицы в том единственном случае, когда условие является истинным. Аргументы "IF.. ENDIFTR" аналогичны аргументам "REPEAT.. ENDREPTR ". Поместите команду IF в первой ячейке строки, а ENDIFTR в последней ячейке строки.

Команда DISPLAY

Команда DISPLAY выводит значение модельного элемента. команда DISPLAY имеет следующие параметры.

Аргумент	Описание	Замечания
Pathname	Путь к выбираемому элементу	Optional
Expression	Выражение для выводимого значения	Optional

Первый параметр - имя пути, к которому относится второй параметр. Имя пути может быть выражено в форме абсолютного и относительного пути. Если имя пути опущено, используется последний путь, установленный предыдущей командой.

Второй параметр - выражение, определяющее значение, которое будет записано в документ. Если первый параметр - "::A", а второй параметр - "current().Documentation ", то будет выбран элемент, с именем A, расположенный непосредственно в главном проекте, и в документ будет помещено значение его свойства "Documentation".

Вариации команды в шаблоне для WORD

- В шаблоне для WORD, использование команды DISPLAY немного отличается. Если тип элемента, выбранного первым параметром - UMLDiagram, а второй параметр опущен, в генерируемый документ будет вставлено изображение указанной диаграммы.
- В шаблоне для WORD, команда DISPLAY имеет, в отличие от в других шаблонов, и третий параметр. Этот параметр указывает, помещать ли выводимое значение в индекс. Это необходимо, чтобы автоматически генерировать список индексов. Если параметр установлен в "I", генератор отмечает слово, выводимое DISPLAY, как индексное.

Вариации команды в шаблоне для POWERPOINT

- В шаблоне для POWERPOINT, существует два вида команды DISPLAY (DISPLAY-TEXT и DISPLAY-IMAGE). Команда DISPLAY-TEXT используется, чтобы выводить текст, а DISPLAY-IMAGE используется, чтобы выводить изображения диаграмм. Параметры настройки эквивалентны параметрам команды DISPLAY. В DISPLAY-IMAGE первый параметр должен быть именем пути, указывающим на диаграмму, а второй параметр должен быть опущен.

Команда SCRIPT

Используйте команду SCRIPT, чтобы расширить возможности вывода. Параметром команды является набор инструкций JScript. Параметр команды SCRIPT в отличие от других параметров включает несколько выражений (инструкций).

Встроенные функции

Ниже приведены встроенные функции, доступные в командах.

Сигнатура	Описание	Целевой шаблон
StarUMLApp():	Возвращает COM объект приложения	WORD,EXCEL,

Сигнатура	Описание	Целевой шаблон
IStarUMLApplication		POWERPOINT
StarUMLProject(): IUMLProject	Возвращает COM объект проекта	TEXT
MSWord(): WordApplication	Возвращает COM объект приложения Word.	WORD
MSExcel(): ExcelApplication	Возвращает COM объект приложения Excel.	EXCEL
MSPPT(): PowerpointApplication	Возвращает COM объект приложения Powerpoint.	POWERPOINT
findByFullpath(Path): IElement	Возвращает элемент по указанному пути.	WORD,EXCEL, POWERPOINT,TEXT
findByLocalpath(RootElem, Path): IElement	Возвращает элемент по относительному пути от RootElem.	WORD,EXCEL, POWERPOINT,TEXT
itemCount(RootElem, CollectionName): int	Возвращает количество элементов в коллекции CollectionName.	WORD,EXCEL, POWERPOINT,TEXT
item(RootElem, CollectionName, Index): IElement	Возвращает элемент коллекции ColletionName по индексу Index.	WORD,EXCEL, POWERPOINT,TEXT
attr(Elem, AttrName): Value	Возвращает свойство или значение ссылки с именем AttrName элемента Elem.	WORD,EXCEL, POWERPOINT,TEXT
current(): IElement	Возвращает последний выбранный элемент.	WORD,EXCEL, POWERPOINT,TEXT
pos(): int	Возвращает индекс текущего элемента в контейнере.	WORD,EXCEL, POWERPOINT
createFile(path): TextStream	Создаёт файл с указанным полным именем и возвращает объект файла.	TEXT
deleteFile(path)	Удаляет существующий файл с указанным полным именем.	TEXT
createFolder(path): Folder	Создаёт папку с указанным полным именем и возвращает объект папки.	TEXT
deleteFolder (path)	Удаляет папку с указанным полным именем.	TEXT
fileExists(path): Boolean	Проверяет, существует ли файл с указанным полным именем.	TEXT
folderExists(path): Boolean	Проверяет, существует ли папка с указанным полным именем.	TEXT
fileBegin(path)	Создаёт файл с указанным именем и вывод всех команд перенаправляется в этот файл, пока не встретится fileEnd.	TEXT
fileEnd(path)	Соответствует fileBegin и останавливает вывод в указанный ею файл.	TEXT
getTarget(path): String	Возвращает путь вывода StarUML Generator UI, установленный пользователем.	TEXT

Написание текстовых шаблонов

Перед написанием текстового шаблона, должны быть выполнены следующие шаги.

1. Скачайте образец текстового шаблона (template-text.zip) из раздела "downloads/templates" домашней страницы StarUML, для того, чтобы генерировать на его основе другие текстовые шаблоны.
2. Создайте новую папку с именем "template-text" и разархивируйте в неё скаченный файл.
3. Запустите StarUML.
4. Выберите меню [Tools] -> [StarUML Generator...].
5. Выберите шаблон "Default Code Template" на странице [Select templates for generator].

6. Щелкните кнопку [Clone Template], укажите название шаблона и путь, где он будет сохранен, и щелкните [OK].
7. Выберите новый созданный шаблон в [List of templates], щелкните кнопку [Open Template], и новый текстовый шаблон будут открыт в экране редактора.
8. Вводите команды в окне редактора, как указано ниже.

Команды, описанные в параграфе "Компоненты шаблона", представляются немного иначе, чем в других шаблонах. Команда в текстовом шаблоне заключается в символы "<@" и "@>". Название команды указывается сразу после "<@", первый параметр отделяется от названия одним пробелом, другие параметры разделяются символом ";". Текст, находящийся вне "<@" и "@>" обрабатывается как стилевая область, которая помещается в генерируемый документ "как есть".

Чтобы перебрать элементы типа "UMLClass", находящиеся по адресу "::Design Model", используется следующая команда

```
<@REPEAT {R}::Design Model;UMLClass;;@>
...
<@ENDREPEAT@>
```

Допустим, Вы хотите распечатать данные о классах модели. Между командами REPEAT и ENDREPEAT поместите текст "class", "{", "}" и команду DISPLAY для вывода имени класса и документации, как показано ниже.

```
<@REPEAT {R}::Design Model;UMLClass;;@>
class <@DISPLAY ;current().Name@> {
  // <@DISPLAY ;current().Documentation@>}
<@ENDREPEAT@>
```

В текстовом шаблоне, есть сокращенная команда, аналогичная команде DISPLAY, но не содержащая параметра пути. Она имеет форму "<@=expression@>" и использует только второй параметр команды DISPLAY. Если показанный выше шаблон выразить через "<@= ..@>", то получится следующий код.

```
<@REPEAT {R}::Design Model;UMLClass;;@>
class <@=current().Name@> {
  // <@=current().Documentation@>}
<@ENDREPEAT@>
```

Использование IF и ENDIF имеет смысл, если нужно что-то анализировать в просмтриваемом классе. В следующем примере документация класа печатается, если она не пустая.

```
<@REPEAT {R}::Design Model;UMLClass;;@>
class <@DISPLAY ;current().Name@> {
<@IF current().Documentation != ""@>
  // <@DISPLAY ;current().Documentation@>
  <@ENDIF@>}
<@ENDREPEAT@>
```

Выражение, используемое как параметр команды, пишется на JScript. В данном контексте может использоваться встроенная функция. Если Вы хотите использовать другую функцию вместо встроенной, определите новую функцию в команде SCRIPT и вызовите её в параметре другой команды. Следующий пример определяет функцию myfunc и отображает значение, возвращаемое этой функцией.

```
<@SCRIPT
function myfunc(a, b) {
  ...
}
@>
```

```
<@DISPLAY ;myfunc(1, 2)@>
```

Команда SCRIPT может использоваться и в других случаях. Ниже показаны другой пример использования команды SCRIPT, который сохраняет каждый класс в файл с тем же именем.

```
<@REPEAT {R}::Design Model;UMLClass;;@>
<@SCRIPT fileBegin(getTarget()+"\\"+current().Name+".java"); @>
class <@DISPLAY ;current().Name@> {
  // <@DISPLAY ;current().Documentation@>
}
<@SCRIPT fileEnd();@>
<@ENDREPEAT@>
```

Если редактирование шаблона завершено и он сохранен, Вы можете генерировать с его помощью ваши собственные тексты. См. главу "Генерация по шаблону" для детального ознакомления с этой операцией.

Написание шаблона для Word

Перед написанием шаблона для WORD, должны быть выполнены следующие шаги.

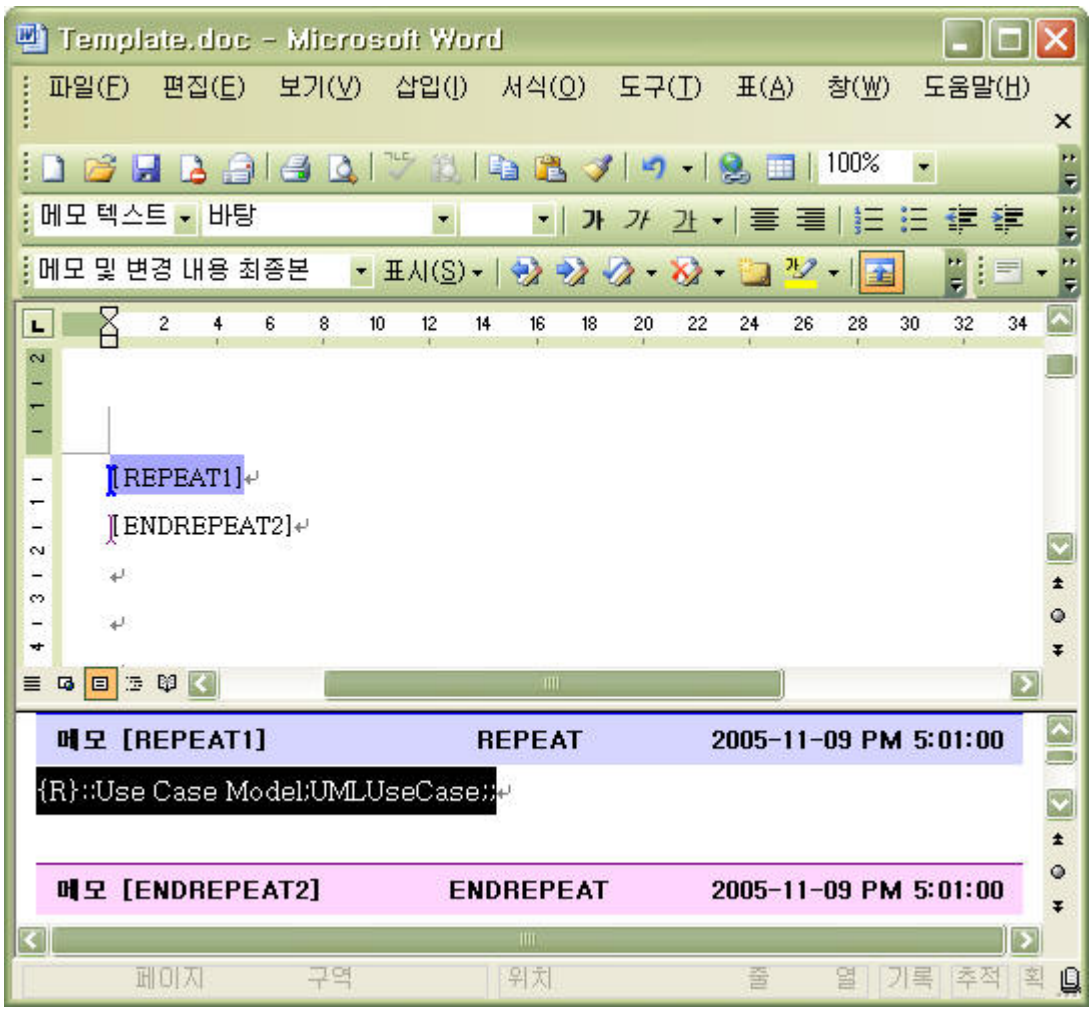
1. Загрузите типовой шаблон (template-word.zip), генерирующий документ WORD из раздела downloads/templates домашней страницы StarUML.
2. Создайте новую папку с именем "template-word" и разархивируйте загруженный файл в эту папку.
3. Запустите StarUML.
4. Выберите меню [Tools] -> [StarUML Generator...].
5. Выберите шаблон "Default Word Template" на странице [Select templates for generator].
6. Нажмите кнопку [Clone Template], укажите имя шаблона и маршрут для размещения, нажмите [OK].
7. Выберите вновь созданный шаблон в [List of templates], нажмите [Open Template], новый шаблон для WORD будет открыт в окне редактора.
8. Вводите команды, соответствующие шаблонам для MS Word.

В шаблоне для WORD команды представляются в виде комментариев WORD. Название команды укзывается в свойстве автора комментария, а аргументы определяются в тексте комментария. Разделитель аргументов - символ ";". Текст шаблона вне комментариев интерпретируется как область стиля, которая помещается в генерируемый документируют как есть.

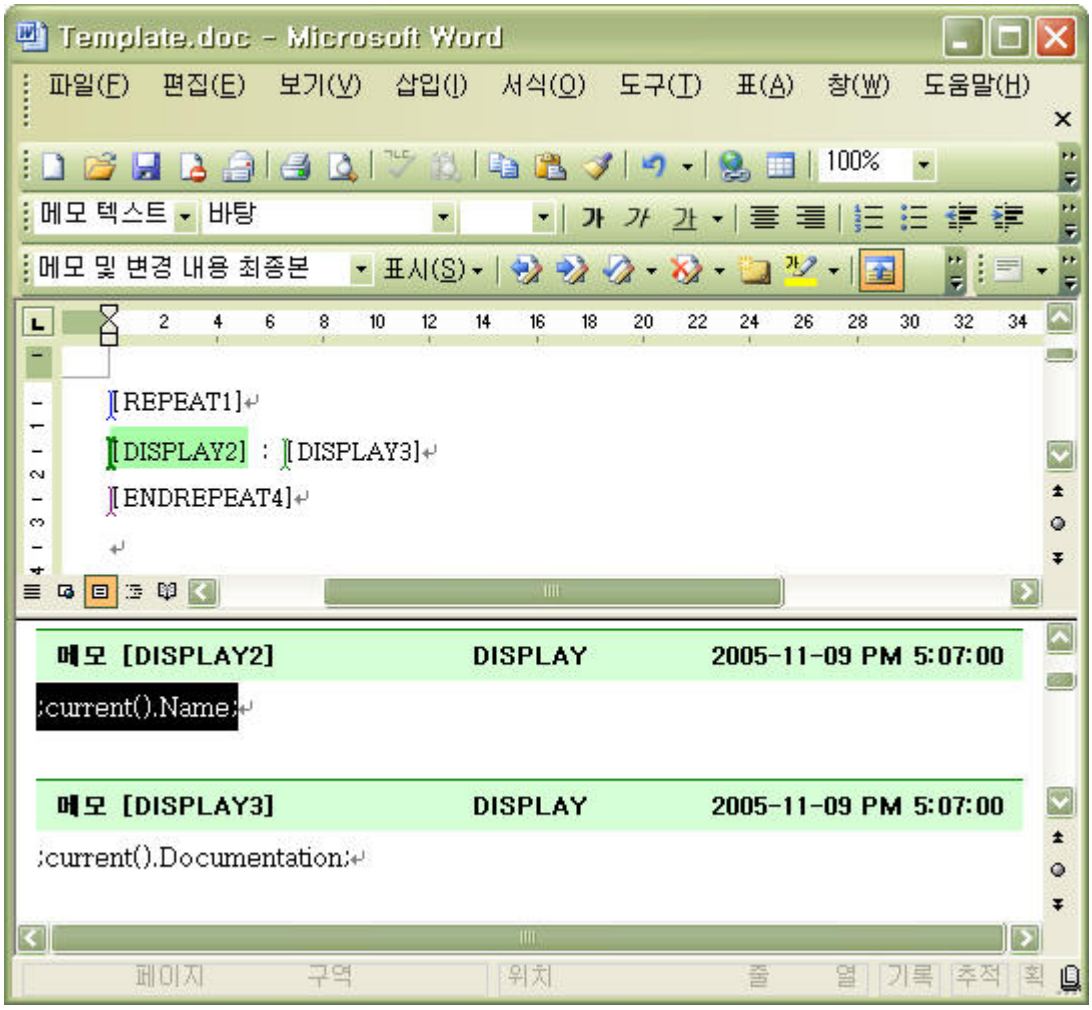
Чтобы проитерировать элементы типа "UMLClass", находящиеся по маршруту "::Design Model", скопируйте и вставьте комментарии [REPEAT] и [ENDREPEAT]. Выберите [REPEAT] и нажмите кнопку инспектора комментария WORD, чтобы установить аргументы команды REPEAT. В открывшемся окне, введите свойства комментария как показано ниже.

Замечание

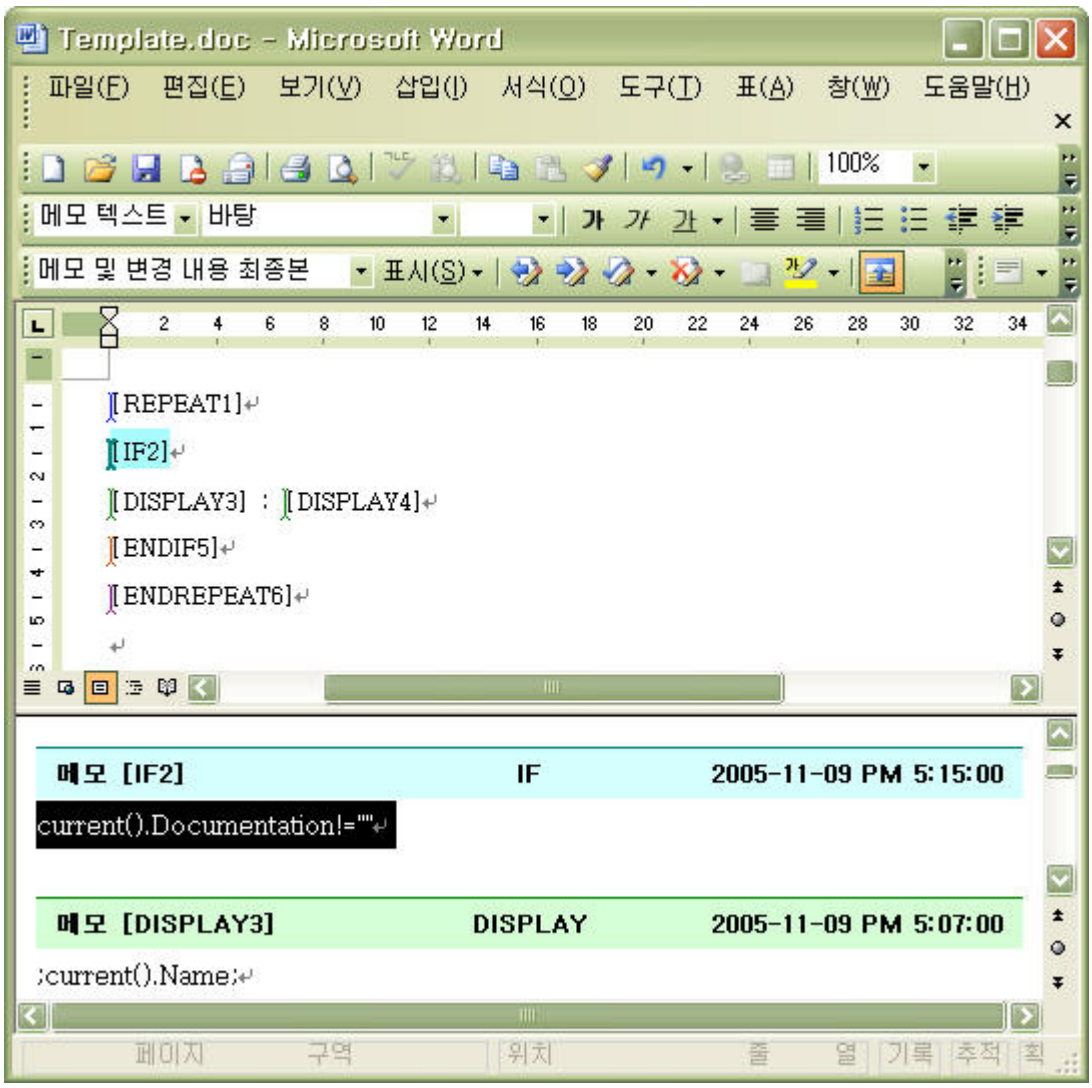
- Свойство автора комментария не устанавливается пользователем. Поэтому копируйте существующие в шаблоне комментарии и вставляйте их в нужную позицию.



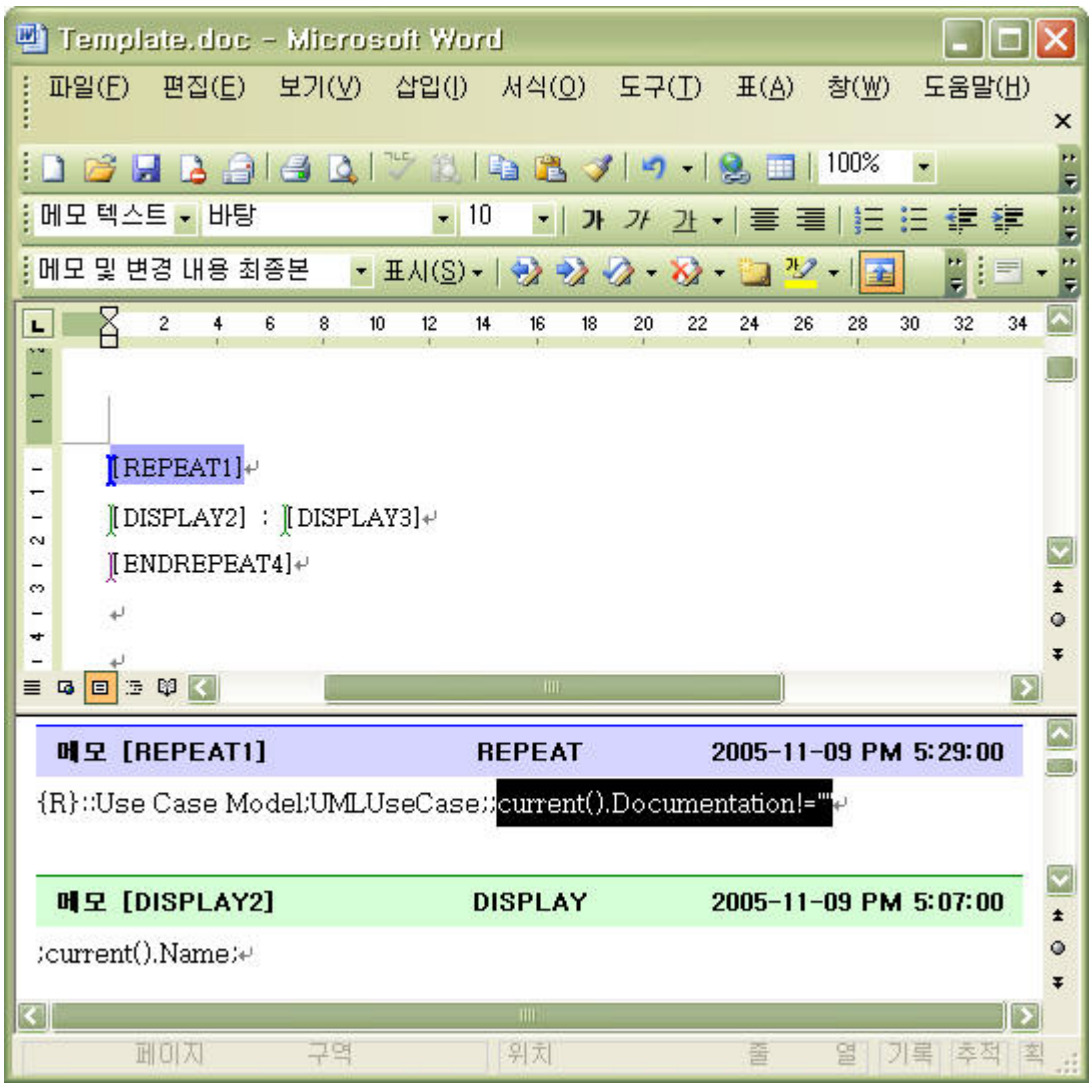
Скопируйте комментарий [DISPLAY] и вставьте его между [REPEAT] и [ENDREPEAT], заполните значения аргументов в тексте комментария, как показано ниже. Эта команда переберёт все прецеденты по маршруту "::Use case Model" и для каждого выведет название и документацию.



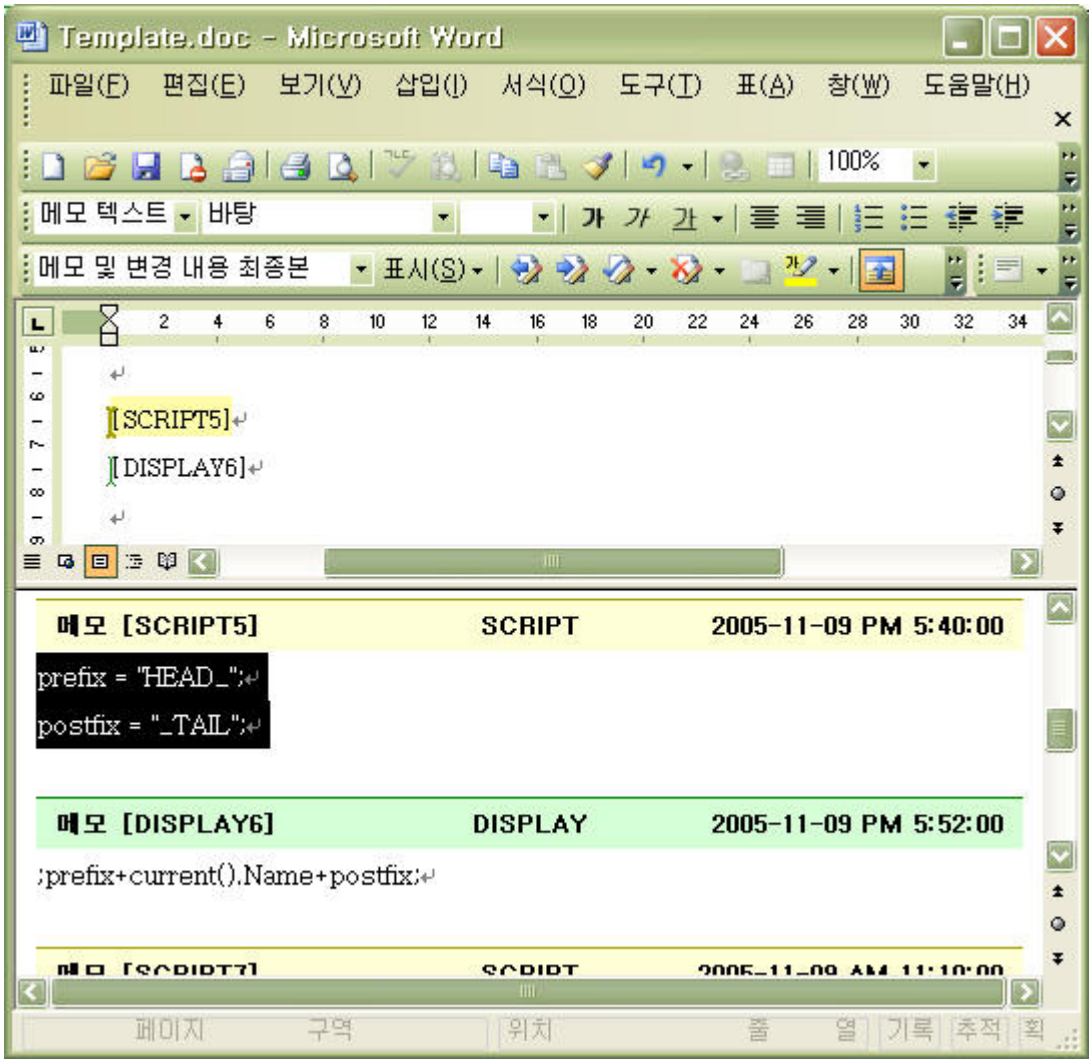
Чтобы сделать что-нибудь один раз при удовлетворении специального условия, используйте комментарии [IF] и [ENDIF], как показано ниже.



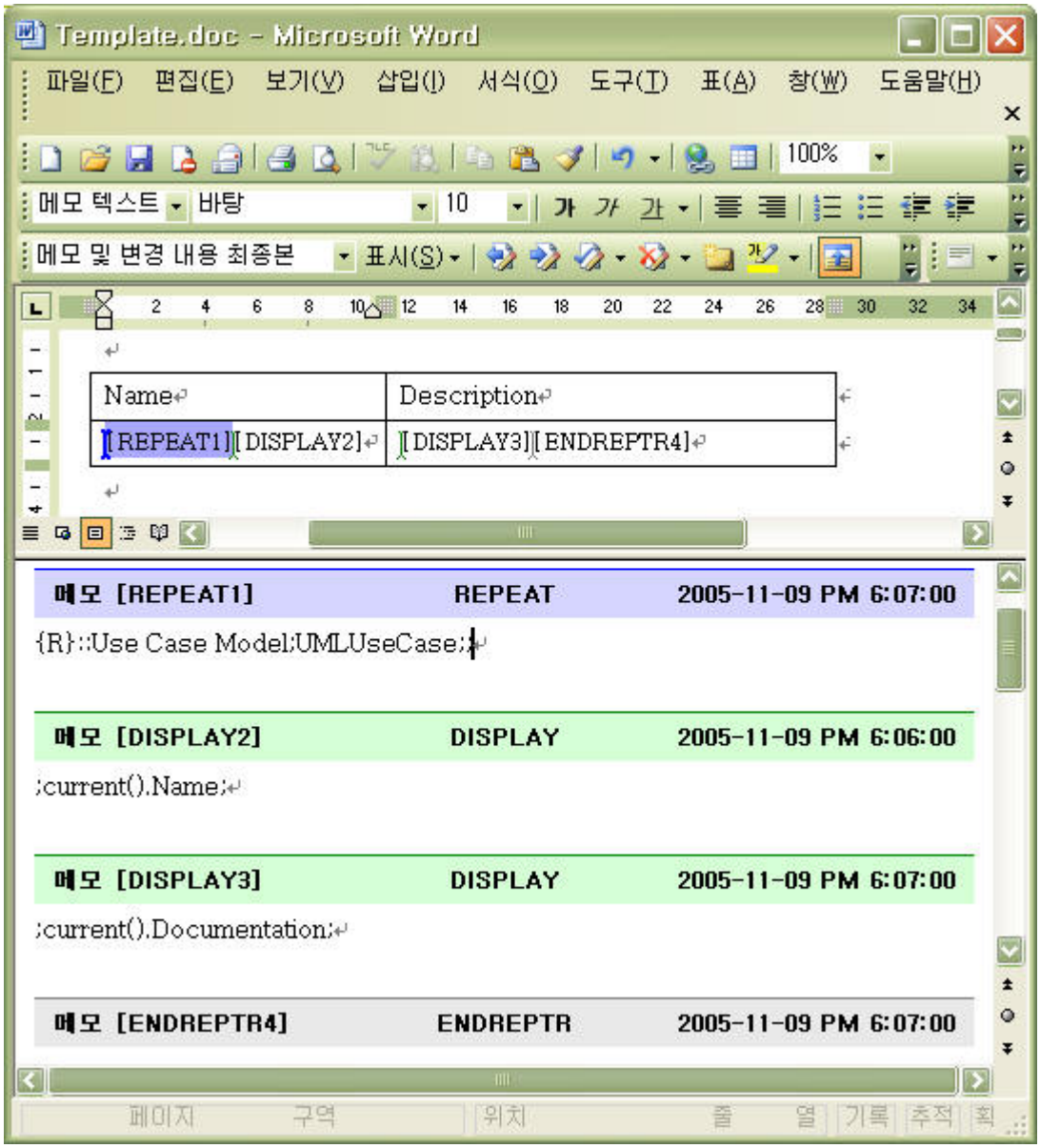
Комбинация [REPEAT] и [IF] может быть заменена одним комментарием [REPEAT]. Переместите аргумент условия команды [IF] в команду [REPEAT] и удалите команды [IF] и [ENDIF]. Получите эквивалентное действие.



Подобно другим шаблонам, шаблон WORD может выполнять инструкции JScript с помощью команды SCRIPT. Если Вы хотите напечатать значение, вычисленное с помощью JScript, заполните инструкцию JScript, которая содержит оператор присваивания переменной в тексте комментария и поместите переменную в качестве аргумента в команду [DISPLAY].



В шаблоне WORD Вы можете итерировать строки таблицы. Чтобы сделать это, используйте команды [REPEAT] И [ENDREPTR]. Аргументы те же самые, что и в случае [REPEAT] и [ENDREPEAT]. Только комментарий [REPEAT] должен быть помещен в первую ячейку строки, а комментарий [ENDREPTR] - в последнюю ячейку строки. ниже приведён пример, который генерирует таблицу с именами и документацией прецедентов.



Закончив редактирование шаблона для WORD, сохраните документ шаблона. После этого Вы сможете генерировать WORD документы на его основе . См. раздел "Генерация по шаблону" для получения подробной информации об этой операции.

Написание шаблона для Excel

Перед написанием шаблона под EXCEL, должны быть выполнены следующие шаги.

1. Загрузите типовой шаблон (template-excel.zip) генерации документа для EXCEL из раздела downloads/templates вэб-страницы StarUML.
2. Создайте новую папку с именем "template-excel" и разорхивируйте в ней загруженный файл.
3. Выполните StarUML.
4. Выберите меню [Tools] -> [StarUML Generator...].
5. Выберите шаблон "Default Excel Template" на странице [Select templates for generator].
6. Щёлкните кнопку [Clone Template], укажите имя шаблона и маршрут для его хранения, нажмите[OK].
7. Выберите вновь созданный шаблон в [List of templates], нажмите кнопку [Open Template], новый шаблон для EXCEL будет открыт в окне редактора.
8. Вводите команды, соответствующие шаблону MS Excel.

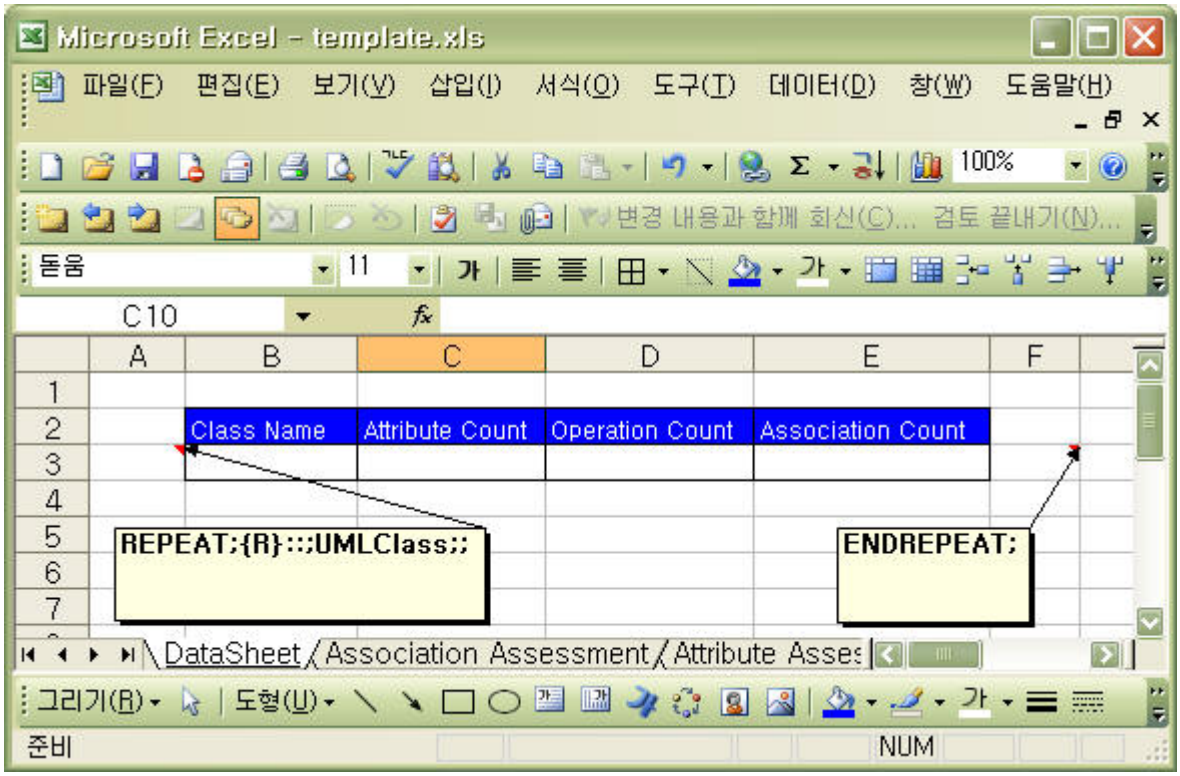
В шаблоне EXCEL, область команды представляется в виде комментария EXCEL. Название команды и её аргументы указываются в тексте комментария. Текст комментария состоит из названия команды и последовательности аргументов. Название и аргументы в тексте комментария разделяются символом ";". Всё, что находится вне комментария расценивается как область стиля и помещается в генерируемый документ как есть.

Шаблон EXCEL может анализировать и оценивать модельную информацию, используя возможности EXCEL (статистика, диаграммы). Этот параграф показывает, как извлекать числовые значения, связанные с модельными элементами и строить графики на их основе.

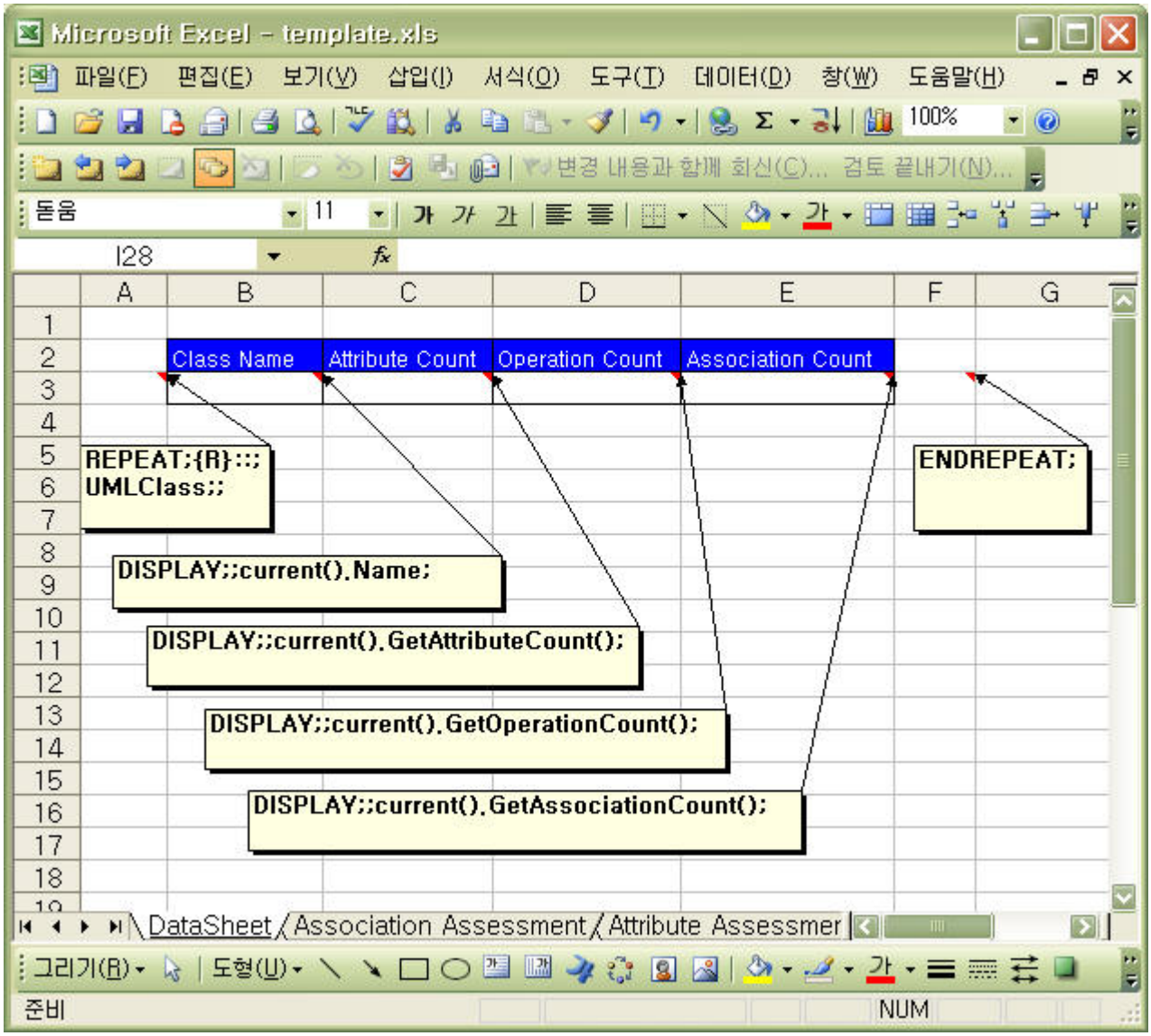
Чтобы создать данные для статистики, Вы должны проитерировать все классы в модели, используя команду REPEAT. Помещайте команды REPEAT и ENDREPEAT в начальной и конечной ячейках целевой строки.

Примечание

- В шаблоне EXCEL команда REPEAT повторяется только для строк, но не для столбцов.



Вставьте команды DISPLAY, которые выводят имя класса, количество атрибутов, количество операций, количество ассоциаций, между командами REPEAT и ENDREPEAT, как показано ниже.



Чтобы создать граф информации о классах, вставьте диаграмму EXCEL и выберите значения количества атрибутов, операций и ассоциаций как исходные данные.

Закончив редактирование шаблона EXCEL, сохраните документ шаблона. После этого Вы сможете генерировать документы EXCEL на основе вашего собственного шаблона. См. раздел "Генерация по шаблону" для получения подробной информаци об этой операции.

Написание шаблонов для PowerPoint

Перед написанием шаблона под POWERPOINT должны быть выполнены следующие шаги.

1. Загрузите образец шаблона (template-powerpoint.zip) для генерации документа POWERPOINT из раздела downloads/templates домашней страницы StarUML. Создайте новую папку с именем "template-powerpoint" и разорхивируйте в ней загруженный файл.
2. Запустите StarUML.
3. Выберите меню [Tools] -> [StarUML Generator...].
4. Выберите шаблон "Default Powerpoint Template" на странице [Select templates for generator].
5. Щёлкните кнопку [Clone Template], укажите имя шаблона и маршрут для хранения, нажмите [OK].
6. Выберите вновь созданный шаблон в [List of templates], нажмите кнопку [Open Template], новый шаблон будет открыт в окне редактора.

7. Вводите коанды, соответствующие шаблону для MS Powerpoint.

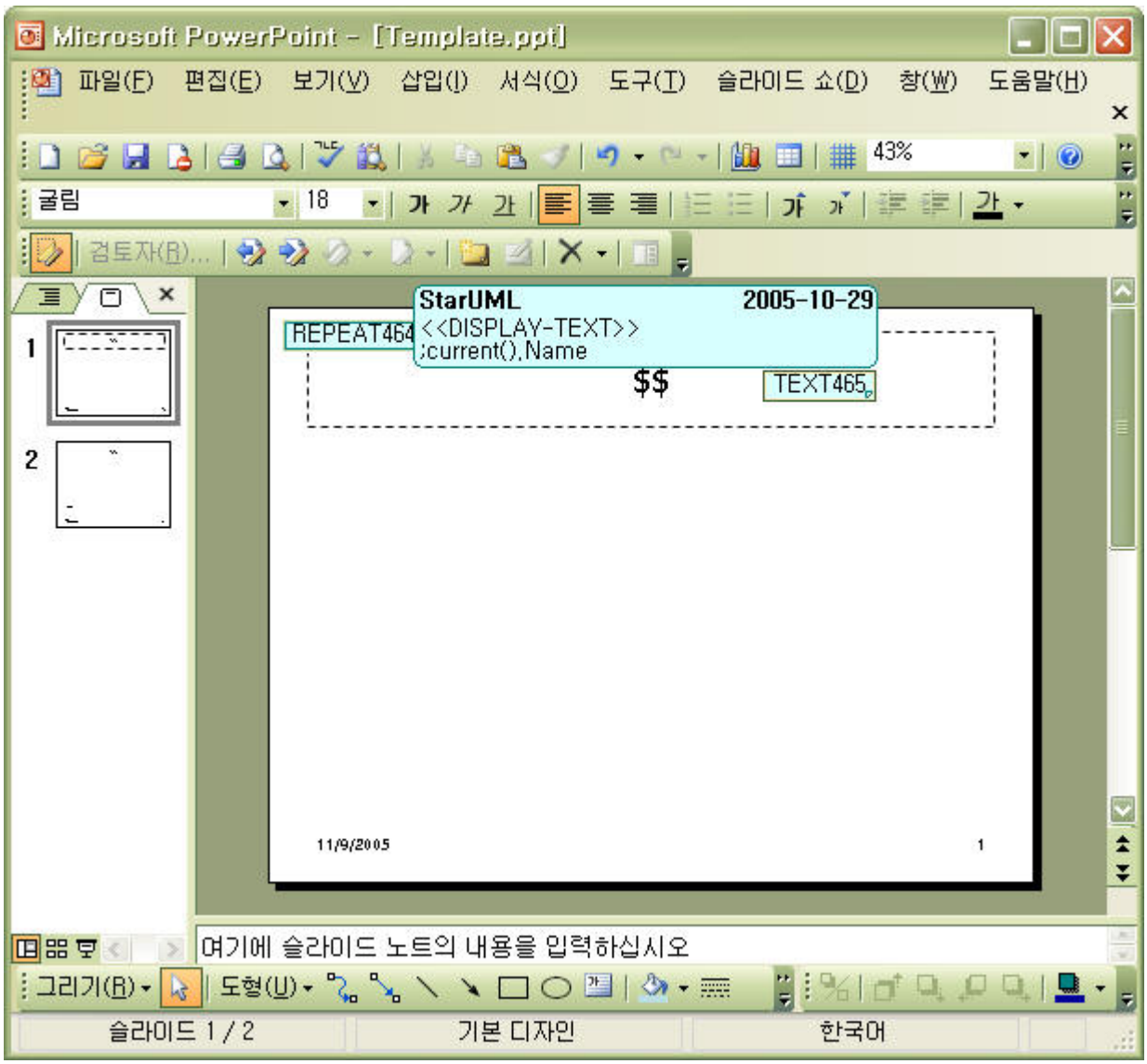
В шаблоне для POWERPOINT, область команды размещается в комментарии POWERPOINT. Название команды заключается в символы "<<" и ">>" в первой строке текста комментария, а аргументы указываются во второй строке текста комментария. Разделитель аргументов - символ ";". Всё, что находится вне области комментария, расценивается как область стиля и выводится в генерируемый документ как есть.

Например, давайте написшем шаблон POWERPOINT, который генерирует слайды, состоящие из диаграмм и документации диаграмм. Прежде всего, чтобы разместить диаграмму в слайде, вставьте комментарий в левом-верхнем углу слайда и установите текст комментария как показано ниже. В данном случае Вы не должны вставлять комментарий ENDREPEAT. Причина будет объяснена позже.

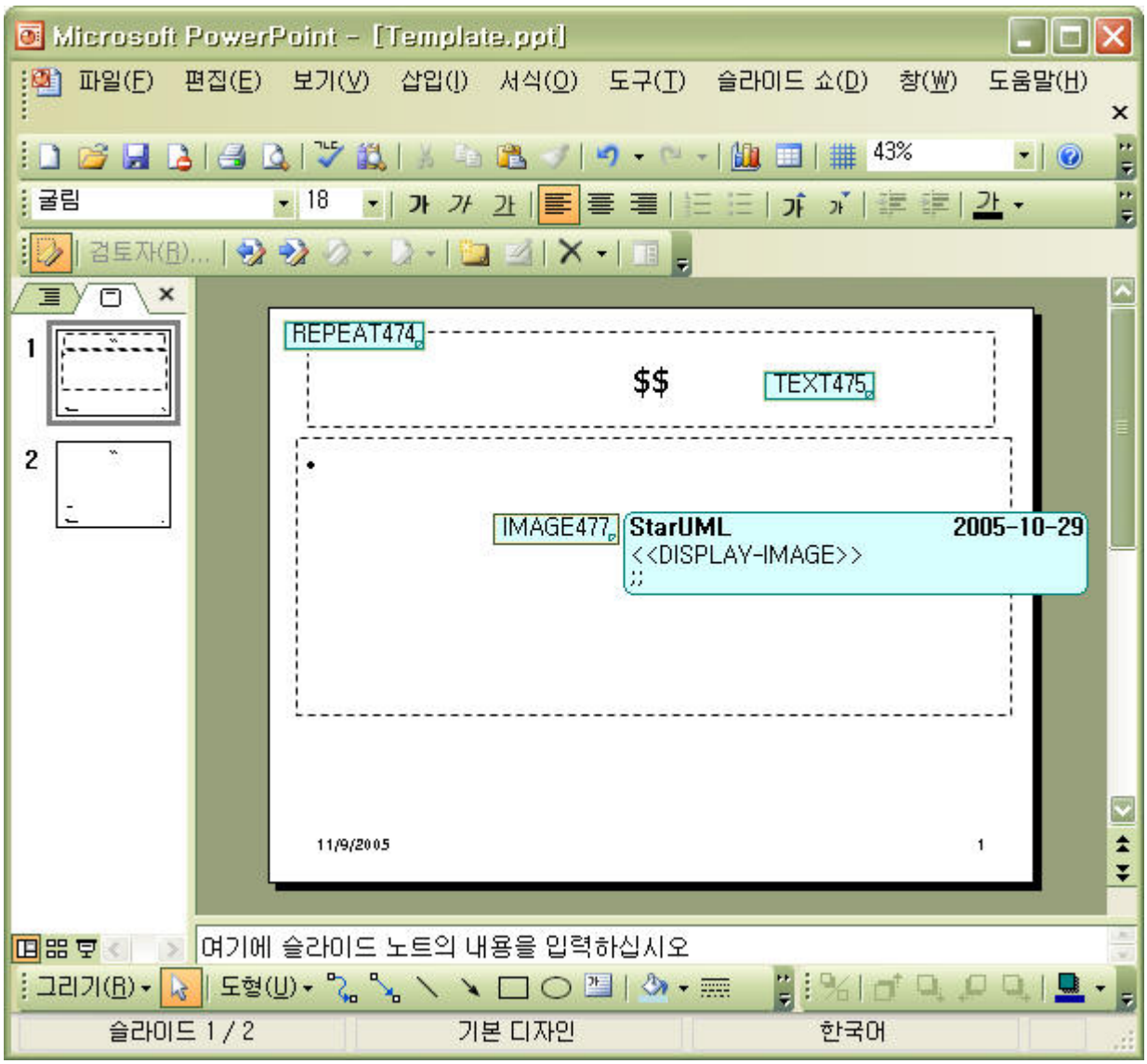
Примечание

- Перед написанием шаблона POWERPOINT учтите, что команда REPEAT повторяет слайд, и ничего кроме слайда.

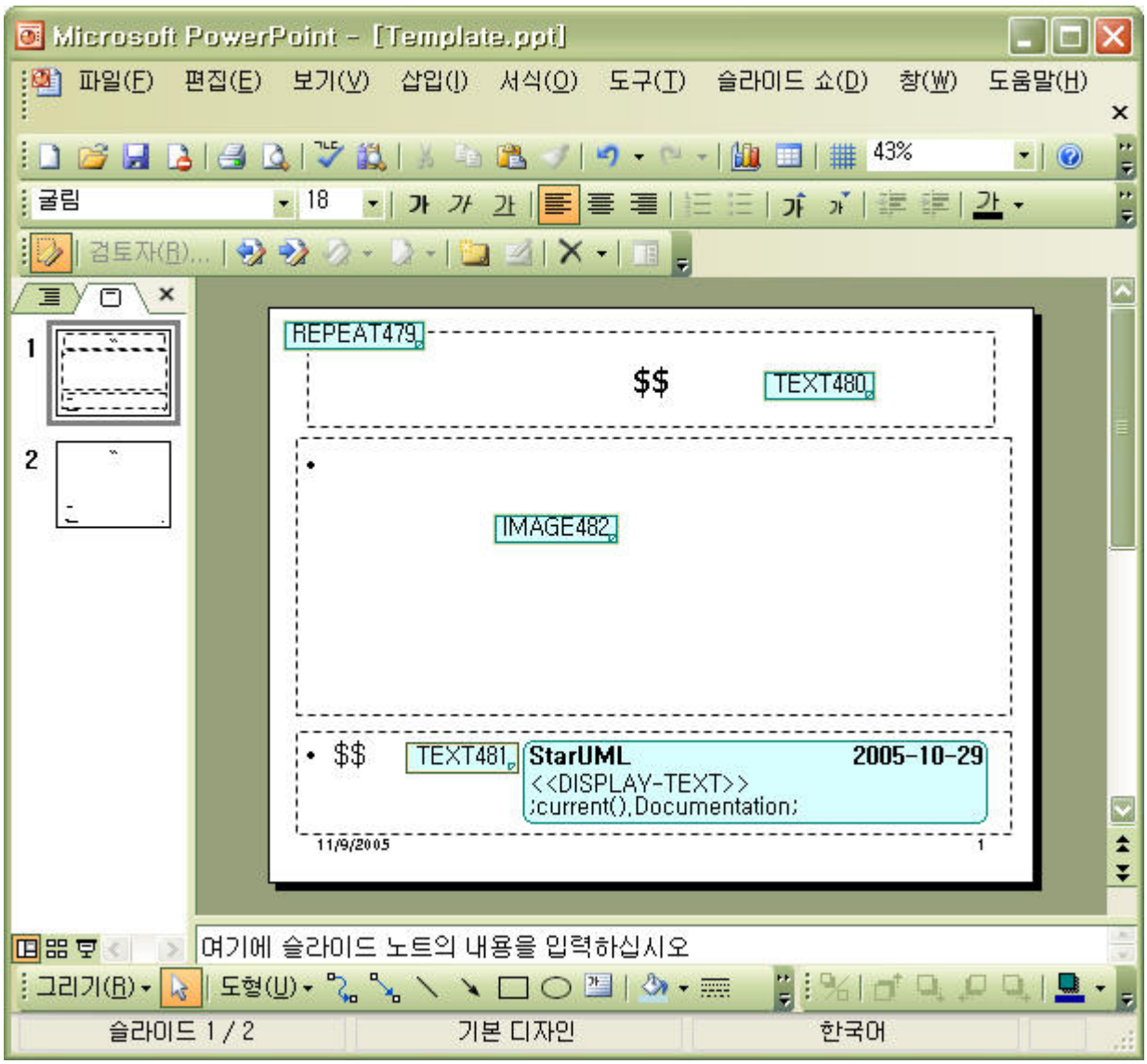
Затем, чтобы вывести название диаграммы в качестве заголовка слайда, вставьте текстбокс и комментарий DISPLAY-TEXT, далее введите указанный ниже текст. Вставьте строку "$$" в текстбокс, чтобы указать команде DISPLAY-TEXT, где печатать текст. Команды DISPLAY - ... печатают только в тех случаях, когда текст или изображение содержит точную границу команды. Поэтому Вы должны поместить команду DISPLAY внутрь границы бокса изображения или текста.



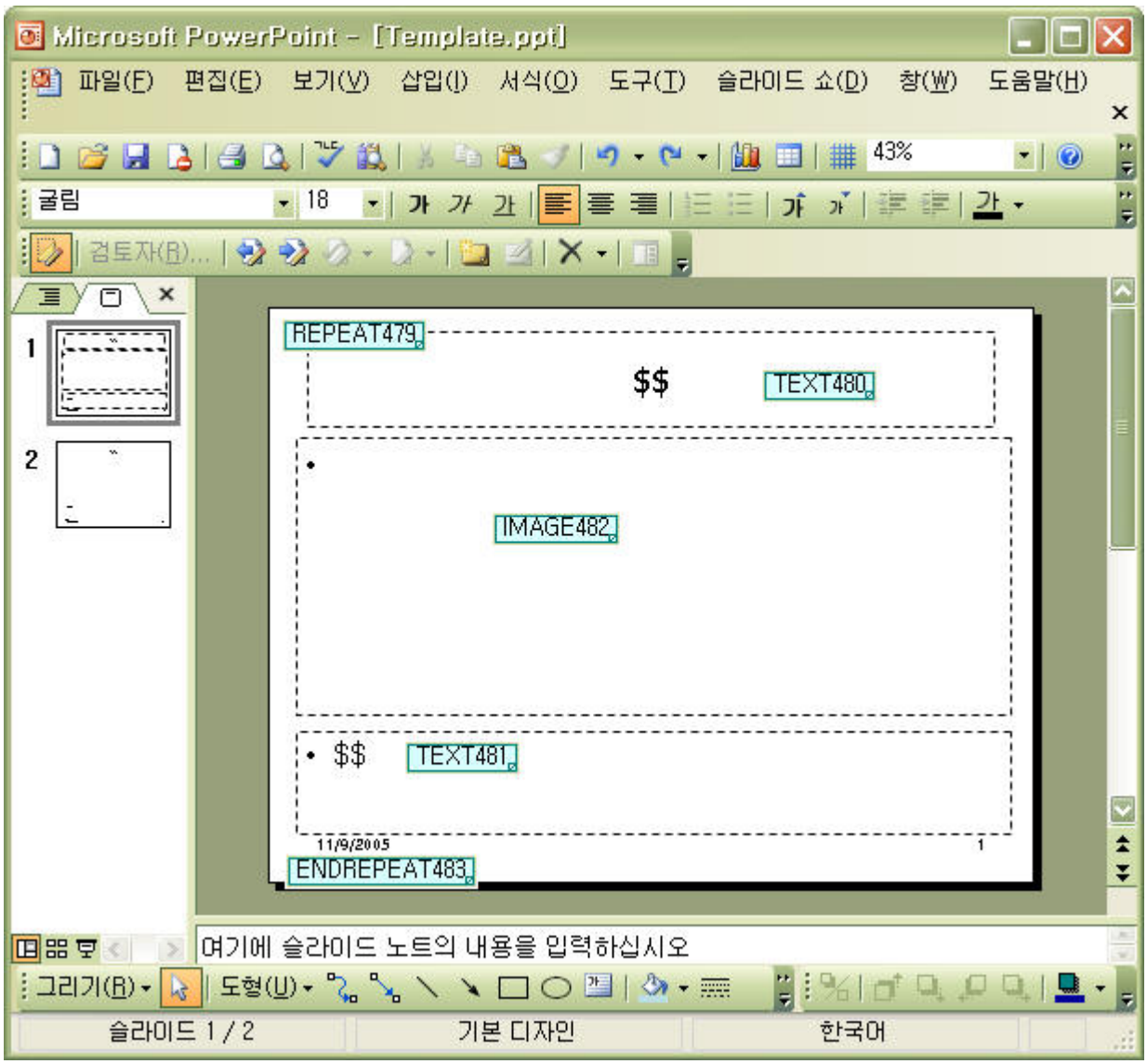
Чтобы разместить диаграмму в середине слайда, вставьте текстбокс и измените его размеры. Также вставьте команду DISPLAY-IMAGE, разместите её в текстбоксе, и введите текст как показано ниже.



Чтобы печатать документацию диаграммы в нижней части слайда, вставьте команду DISPLAY-TEXT и текстбокс, установите текст комментария как показано ниже.



Наконец, чтобы указать границу повторяемой области, вставьте команду ENDREPEAT в низу слайда. Причина того, что ENDREPAT была вставлена в самую последнюю очередь, заключается в следующем. Порядок интерпретации команд шаблона POWERPOINT определяется не позицией комментариев, а очерёдностью их создания. Одна команда выполняется раньше другой не потому что она расположена выше, а потому что она была создана раньше. Если Вы вставляете REPEAT, ENDREPEAT, DISPLAY-TEXT в указанном порядке, генератор считает, что не существует никаких команд между REPEAT и ENDREPEAT. Чтобы повторить команды, следующие за командой REPEAT, Вы должны создать команду REPEAT, затем повторяющиеся команды, и только после этого ENDREPEAT.



Если редактирование шаблона POWERPOINT закончено, сохраните документ шаблона. После этого Вы сможете генерировать презентации на основе этого шаблона. См. раздел "Генерация по шаблону" для получения подробной информации об этой операции.

Регистрация шаблонов

Пользователь может регистрировать собственные шаблоны в генераторе

1. Нажмите кнопку [Register Template] на странице [Select templates for generation].
2. В диалоге [Register Template], щёлкните кнопку  и укажите маршрут.
3. Введите информацию о шаблоне в окне [Properties:] и нажмите кнопку [OK]. Регистрация закончена.

Основная информация

Укажите информацию о шаблоне: название шаблона, группу, категорию и описание.

Элемент	Описание
Template Name	Указывает имя целевого шаблона.

Элемент	Описание
Group	Указывает группу, содержащую целевой шаблон.
Category	Указывает категорию шаблона внутри группы.
Описание	Указывает описание шаблона.

Подробная информация

Укажите подробную информацию о шаблоне.

Элемент	Описание
Document Type	Указывает тип документа. Выберите одно из DOCUMENT, REPORT, CODE.
Format	Указывает формат документа.
Version	Указывает версию шаблона.
Related Profile	Указывает профиль, относящийся к шаблону.
Related Approach	Указывает подход, относящийся к шаблону.
Translator Type	Указывает тип генератора. Допустимо одно из следующих значений.

Значение	Смысл
WORD	генератор под Word
EXCEL	генератор под Excel
POWERPOINT	генератор под Powerpoint
TEXT	генератор текстового файла
COM	генератор на базе COM-объекта, определённый пользователем
SCRIPT	генератор на базе скрипта, определённый пользователем
EXE	генератор, на базе exe-файла, сделанный пользователем

Translator	Указывает имя файла генератора. Допустимо для генератора, созданного пользователем.
Example	Указывает простое имя модели, к которой применяется шаблон.
Parameters	указывает требуемые параметры.
Related files	Указывает файл генерации.

Параметры

1. Нажмите кнопку свойства параметров  .
2. В диалоге [Parameters] щёлкните кнопку  , чтобы добавить параметр, щёлкните кнопку , чтобы удалить параметр.
3. В диалоге [New Parameter] укажите имя, тип, значение по умолчанию и нажмите [OK].

Устанавливаемые парметры под кждый тип транслятора показаны ниже.

Элемент	Тип	Тип транслятора	Описание
TemplateFile	FILENAME or STRING	WORD,EXCEL, POWERPOINT	Указывает имя файла шаблона.
OutputFile	FILENAME or STRING	WORD,EXCEL, POWERPOINT, TEXT	Указывает имя файла результирующего документа.
Keep Comment	BOOLEAN	WORD,EXCEL, POWERPOINT	Указывает, что результирующий файл содержит командную информацию
ShowGenerationProcess	BOOLEAN	WORD,EXCEL, POWERPOINT	Указывает, нужно ли показывать прогрессор MS Office. Если установлен в true, замедляет

Элемент	Тип	Тип транслятора	Описание
			процесс генерации.
Normal Generation	BOOLEAN	WORD	Указывает начальный целевой путь генерации. Если установлен в false, начальным элементом является текущий элемент.
Generate Index	BOOLEAN	WORD	Указывает, нужно ли генерировать индекс
intermediate	STRING	TEXT	Указывает, что промежуточные файлы для генерации сгенерированы.
target	STRING	TEXT	Указывает путь к папке, которая содержит генерируемые файлы.

Ссылка

- При установке параметров Вы можете использовать константы, поддерживаемые генераторм StarUML, показанные ниже.

Имя	Описание
$PATH$	Путь к папке, содержащей файл шаблона и файл описания шаблона.
$GROUP$	Значение свойства группы шаблона.
$CATEGORY$	Значение свойства категории шаблона.
$NAME$	Имя шаблона
$TARGET$	Путь к папке, которую пользователь выбрал в диалоге **[Generator]**.

Об управлении зарегистрированными шаблонами, см. раздел "Генерация по шаблонам" в "Главе 7. Генерация кодов и шаблонов" Руководства пользователя.

Создание дистрибутивного пакета шаблона

Шаблоны устанавливаются в папку "staruml-generator". Все шаблоны и пакетные задачи располагаются в папке "templates", вложенной в папку "staruml-generator". Вообще все файлы ресурсов, связанные с одним шаблоном, располагаются в одной папке. Папка шаблона должна быть вложена в папку "templates". Шаблон состоит из файла описания шаблона (*.tdf) и собственно шаблона (*.doc, *.ppt, *.xls, *.cot, и т.д.). Файл описания шаблона содержит параметры, описанные в руководстве пользователя "Глава 7. Генерация кодов и документов. Регистрация шаблона". Пакетная задача описывается в пакетном файле задачи. Пакетный файл задачи имеет расширение ".btf" и располагается в папке "batches", вложенной в папку "staruml-generator".

Расширение файла	description
BTF	Содержит список пакетных задач и параметры каждой задачи.
TDF	Содержит информацию о шаблоне (имя, тип, имя файла, параметры и т.п.)
DOC, DOT	содержит команды и стиль шаблона для Word
XLS, XLT	содержит команды и стиль шаблона для Excel
PPT, POT	содержит команды и стиль шаблона для Powerpoint
COT	содержит команды и стиль шаблона для текстового документа

Структура папки генератора

Общая структура папки генератора следующая.

```
staruml-generator\
    templates\
        template1\
            template1.tdf
            template1.doc
        template2\
            ...
    batches\
        batch1.btf
        ...
```

Установка и удаление шаблона

Установить шаблон очень просто. Сделайте копию папки, которая содержит шаблон, и вставте (перенесите) её в папку "staruml-generator\templates" целевого компьютера. Инсталляция закончена.

Удалить шаблон также очень просто. Удалите папку, которая содержит ненужный шаблон.

Упаковка шаблонов

Структура каталогов внутри папки "staruml-generator\templates" может быть произвольной.

Поэтому Вы можете упорядочивать шаблоны, не изменяя список пакетов и информацию шаблонов. Это облегчает управление шаблонами и их дистрибуцию. Например, Вы можете вложить несколько папок шаблонов в одну папку, упаковать их в архив типа ZIP, и поставить их на другой компьютер. Получатель, чтобы установить эти шаблоны, должен только извлечь их в папку "staruml-generator\templates".

Инсталляция и удаление пакетной задачи

Установить пакетную задачу очень просто. Перед инсталляцией пакета, установите шаблоны, используемые в пакетной задаче. Затем, сделайте копию файла пакетной задачи (*.btf). Перенесите её в каталог "staruml-generator\batches" целевого компьютера. Инсталляция закончена.

Удалить пакетную задачу также очень просто. Просто удалите пакетный файл задачи (*.btf).