

## StarUML 5.0 개발자 가이드

---

Copyright © 2005 Minkyu Lee.  
Copyright © 2005 Hyunsoo Kim.  
Copyright © 2005 Jeongil Kim.  
Copyright © 2005 Jangwoo Lee.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

## 차례

---

### 제 1 장. 소개

StarUML 개요

왜 UML/MDA 플랫폼이 필요한가?

### 제 2 장. StarUML 아키텍처

플랫폼 아키텍처

모듈 구성

Open API 개요

### 제 3 장. HelloWorld 예제

"Hello, world" 예제

스크립트 작성

메뉴 확장 파일 작성

Add-In 배치

Add-In 등록

Add-In 추가 확인 및 실행

### 제 4 장. Open API 사용하기

프로젝트 다루기

모델링 요소 다루기

애플리케이션 자동화 개체 다루기

메타-모델 사용하기

### 제 5 장. 접근법 사용하기

접근법의 기본 개념

새로운 접근법 작성하기

접근법 관련 메소드 사용하기

### 제 6 장. 프레임워크 사용하기

모델 프레임워크의 기본 개념

새로운 모델 프레임워크 작성하기

새로운 모델 프레임워크 등록하기

모델 프레임워크 관련 메소드 사용하기

### 제 7 장. 프로파일 사용하기

UML 프로파일의 기본 개념

UML 프로파일 작성하기

UML 프로파일 등록하기

확장 요소 개체 다루기

### 제 8 장. 메뉴 확장하기

메뉴 확장의 기본 개념

메뉴 확장 파일 작성하기

메뉴 확장 파일 등록하기

### 제 9 장. Add-in COM Object 사용하기

Add-In COM Object 기본 개념

IStarUMLAddIn 인터페이스 메소드

Add-In COM Object 예제

Add-In 설명 파일 작성하기

Add-In 설명 파일 등록하기

옵션 확장

옵션 스키마 작성하기

옵션 스키마 등록하기

- 옵션값 접근하기
- 이벤트 수신기 기본 개념
- 이벤트의 종류
- 이벤트 수신하기

#### 제 10 장. 노테이션 확장하기

- 왜 노테이션 확장이 필요한가?
- 노테이션 확장 언어
- 새로운 종류의 다이어그램 만들기

#### 제 11 장. 템플릿 작성하기

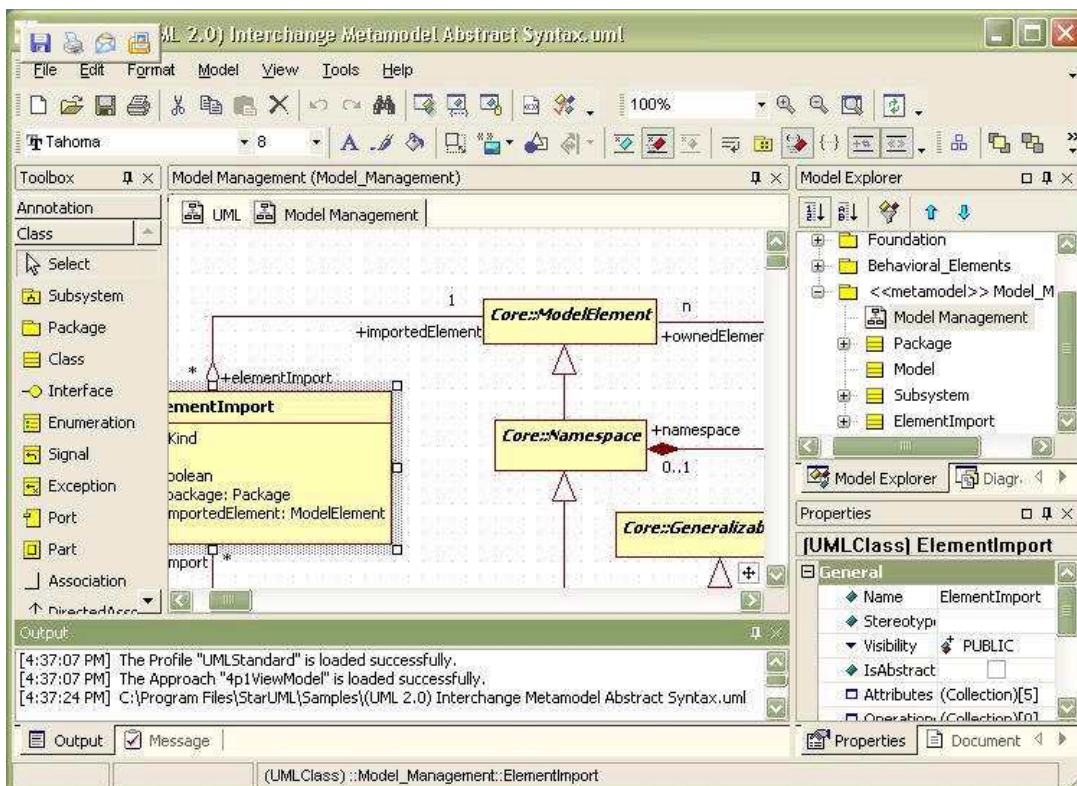
- 템플릿 구성 요소
- 텍스트 기반 템플릿 작성하기
- 워드 템플릿 작성하기
- 엑셀 템플릿 작성하기
- 파워포인트 템플릿 작성하기
- 템플릿 등록하기
- 템플릿 배포하기

## 제 1 장. 소개

**StarUML™ 개발자 가이드**는 UML 기반 소프트웨어 모델링 플랫폼인 **StarUML™**의 확장 메커니즘을 사용하여, 개발자들이 **StarUML™** 모듈을 개발하는데 필요한 정보를 제공한다.

## StarUML 개요

StarUML™은 UML(Unified Modeling Language)을 지원하는 소프트웨어 모델링 플랫폼으로써, UML 버전 1.4에 기반을 두고, UML 2.0 표기법을 지원하고 있으며 총 11가지의 다양한 종류의 다이어그램을 제공한다. 그리고 UML 프로파일 개념을 지원하여 MDA(Model Driven Architecture) 접근방법을 적극적으로 지원한다. StarUML™은 고객의 환경에 대한 맞춤 능력이 우수하고 기능에 대한 확장성이 매우 뛰어난 것이 장점이다.



### 고객에 적응하는 UML 도구

StarUML™은 고객의 환경에 최대한 적응할 수 있도록 설계되어 있습니다. 따라서, 고객의 소프트웨어 개발 방법론, 프로젝트의 플랫폼, 언어 등에 모두 적응할 수 있는 커스터마이징 변수들을 제공한다.

### 진정한 MDA 지원 도구

소프트웨어 아키텍처는 향후 10년 이상 내다보는 매우 중요한 작업입니다. OMG에서는 MDA 기술을 통해서 플랫폼에 독립적인 소프트웨어 모델을

구성하고 그것으로부터 플랫폼에 의존적인 모델이나 코드 등을 자동으로 얻을 수 있도록 하는 것을 지향하고 있습니다. StarUML™은 UML 1.4 표준 메타모델과 2.0 표기법을 최대한 준수하면서 UML Profile 개념을 제공하여 플랫폼에 독립적인 모델을 작성할 수 있도록 지원하며, 간단한 템플릿 문서 작성만으로 고객이 원하는 산출물을 쉽게 얻을 수 있다.

## 놀라운 확장성과 유연성

StarUML™은 놀라운 유연성과 확장성을 제공합니다. 도구의 기능을 확장하기 위한 Add-In 프레임워크를 제공하고, COM Automation을 통한 모델/메타모델 및 도구의 모든 기능에 접근할 수 있으며, 메뉴 및 옵션 항목까지도 확장할 수 있도록 설계되어 있습니다. 또한 고객의 방법론에 맞도록 접근법(Approach) 및 프레임워크(Framework)를 직접 추가 작성할 수 있고 어떠한 외부 도구와도 통합이 가능하다.

## 왜 UML/MDA 플랫폼이 필요한가?

StarUML™은 하나의 소프트웨어 모델링 플랫폼(Software Modeling Platform)이다. 그럼, 왜 단순한 UML 도구가 아닌 모델링 플랫폼이 필요한가?

- 최종 사용자는 커스터마이징이 가능한 도구를 원한다. 따라서, 다양한 커스터마이징 변수들을 제공해서 사용자의 환경에 최적으로 적용할 수 있도록 해야 높은 생산성과 품질을 달성할 수 있다.
- 모든 기능을 제공하는 모델링 도구는 없다. 따라서, 점진적으로 기능을 추가할 수 있어서 기존의 도구를 구입할 때 들었던 투자 비용을 보충할 수 있어야 한다.
- MDA(Model Driven Architecture) 기술은 플랫폼 독립성을 요구할 뿐만 아니라 멀티-플랫폼 기능들을 요구한다. 따라서, 특정 개발 환경에 편입된 모델링 도구는 MDA에 적합하지 않고 그 자체가 모델링 플랫폼이 되어 다양한 플랫폼 기술과 도구들에 대한 기능을 제공해야 한다.
- 도구의 효율을 극대화하기 위해서는 다른 도구들과의 통합은 필수적이다. 따라서, 높은 수준의 확장성을 제공해야 하고 이를 통해 기존의 도구 혹은 사용자의 특수한 도구와의 통합이 이루어질 수 있어야 한다.

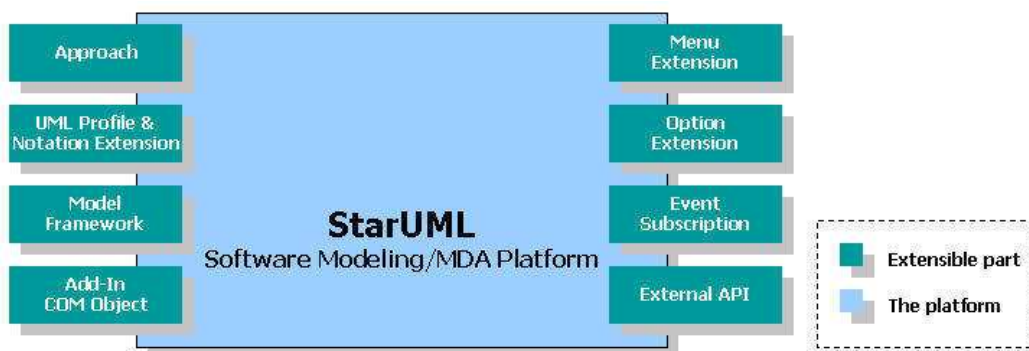
처음으로

## 제 2 장. StarUML 아키텍처

이 장에서는 StarUML™의 기본 아키텍처에 대해서 설명한다. 플랫폼 아키텍처와 Add-In의 구성 그리고 외부 API(External API)가 어떻게 구성되어 있는지에 대해 주로 기술한다.

### 플랫폼 아키텍처

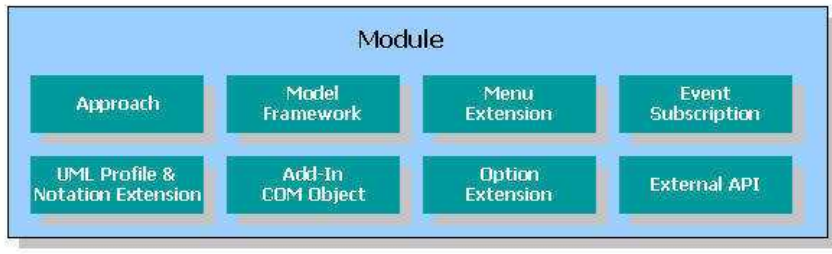
StarUML™은 단순히 정의된 기능들만을 제공하는 것이 아니라 새로운 기능들을 얼마든지 추가할 수 있는 확장 가능한 소프트웨어 모델링 플랫폼이다. 다음 그림은 StarUML™의 아키텍처 구성을 보여준다. 하늘색은 플랫폼(Platform)을 의미하고 초록색은 확장 가능한 부분(Extensible part)을 의미한다. 확장 가능한 부분은 사용자 혹은 제3자에 의해 개발된 후 플랫폼에 추가되어 상호 연동될 수 있다.



- **Approach** : 접근법(Approach)은 프로젝트의 모델과 다이어그램의 기본 구성(organization)을 정의할 수 있도록 한다. 접근법에 대한 자세한 내용은 "**Chapter 5. Writing Approaches**"를 참고하라.
- **UML Profile & Notation Extension** : UML 프로파일(Profile)은 UML의 확장 메커니즘을 통해 소프트웨어 모델에 대한 표현력을 확장할 수 있도록 한다. UML 프로파일에 대한 자세한 내용은 "**Chapter 7. Writing UML Profiles**"를 참고하라.
- **Model Framework** : 모델 프레임워크(Model Framework)는 소프트웨어 모델을 재사용 가능하도록 하여 다른 소프트웨어 모델을 정의할 때 함께 사용될 수 있도록 한다. 모델 프레임워크에 대한 자세한 내용은 "**Chapter 6. Writing Frameworks**"를 참고하라.
- **Add-In COM Object** : Add-In COM 개체는 StarUML™에 새로운 기능(Functionality)을 추가할 수 있도록 한다. Add-In COM 개체에 관한 자세한 내용은 "**Chapter 9. Writing Add-In COM Object**"를 참고하라.
- **Menu Extension** : StarUML™ 애플리케이션의 메뉴(메인 메뉴 및 팝업 메뉴)를 사용자가 원하는 대로 추가할 수 있도록 한다. 메뉴 확장에 관한 자세한 내용은 "**Chapter 8. Extending Menu**"를 참고하라.
- **Option Extension** : StarUML™의 옵션 항목들을 사용자가 원하는 대로 추가할 수 있도록 한다. 옵션 확장에 관한 자세한 내용은 "**Chapter 9. Writing Add-in COM Object**"를 참고하라.
- **Event Subscription** : StarUML™에서 발생하는 다양한 이벤트를 수신할 수 있도록 한다. 이벤트 수신에 관한 자세한 내용은 "**Chapter 9. Writing Add-in COM Object**"를 참고하라.
- **External API** : StarUML™에서 외부에 공개된 API로 다양한 기능과 정보에 대한 접근을 허용한다. 외부 API에 관한 자세한 내용은 이 개발자 가이드 전반에 걸쳐 설명되며 특히 StarUML™ 프로그램 설치 시 예제의 하나로 제공되는 '**StarUML Application Model.uml**'를 참고할 수 있다. 이에 대해서는 "**Appendix A. Plastic Application Model**" 부분을 참고하라.

## 모듈 구성

모듈은 StarUML™을 확장하여 새로운 기능과 특징들을 추가할 수 있도록 해주는 소프트웨어 패키지이다. 모듈은 StarUML™의 여러 가지 확장 메커니즘으로 구성되는데 다음의 그림과 같이 여러 개의 접근법, 여러 개의 모델 프레임워크, 여러 개의 UML 프로파일 그리고 여러 개의 스크립트 (Script)들과 메뉴 확장(Menu Extension), 옵션 확장(Option Extension), 도움말(Help) 및 Add-In COM Object가 사용되어 하나의 패키지로 구성될 수 있다.



## 모듈의 응용

모듈은 다양한 구성 요소들을 포함할 수 있기 때문에 여러 가지의 목적에 맞도록 개발될 수 있다. 특정 프로세스나 언어 그리고 플랫폼을 지원하는 것에서부터 더 도구와의 연동과 기능의 확장 등으로 고려될 수 있다.

- 특정 프로세스 지원 : UML Components, RUP, Catalysis, XP, ...
- 특정 프로그래밍 언어 지원 : C/C++, Python, C#, Visual Basic, Java, Perl, Object Pascal, ...
- 특정 도구와의 연동 : Visual SourceSafe, CVS, MS Word, Eclipse, Visual Studio.NET, ...
- 기타 기능의 확장 : Traceability Manager, Design Patterns Support, Rule Checking, ...
- 자신(혹은 특정 회사)만의 환경 구축 등

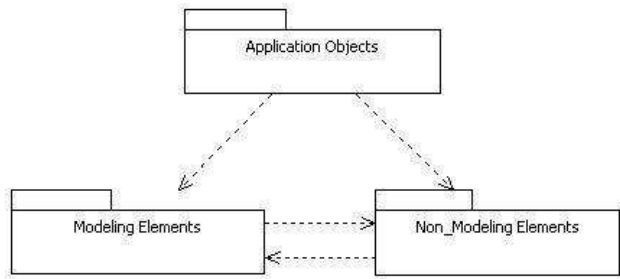
## 모듈의 각 구성 요소

- **Approach** : 접근법은 프로젝트 시작 시에 적용되어 초기 모델 구조를 결정 짓는다. 예를 들면 이것은 특정 프로세스를 위한 모듈을 만드는 경우 프로세스의 각 단계가 산출하게 되는 모델을 관리하기 위한 구조를 미리 설정하는 용도로 사용될 수 있다.
- **Model Framework** : 모델 프레임워크는 특정 언어나 플랫폼에 관계된 모듈을 개발하는 경우에 클래스 라이브러리(Class Library) 혹은 애플리케이션 프레임워크(Application Framework)를 제작할 수 있고, 기타 기본 서비스(e.g. Event, Transaction, Security, Directory, ...)들을 모델로 개발하여 추가할 수도 있을 것이다.
- **UML Profile** : UML 프로파일은 특정 프로세스나 언어, 프레임워크 등 다양한 용도에서 UML의 표기법을 확장해야 하거나 추가적인 프로퍼티를 사용해야 하는 경우에 정의될 수 있다. 이것은 모듈에서 매우 광범위한 역할을 수행할 수 있다.
- **Menu Extension** : Add-In은 대부분 새로운 기능들을 추가하는데 각각의 기능들을 사용자가 선택하여 실행할 수 있도록 StarUML™의 메인 메뉴 혹은 팝업 메뉴를 확장하는데 사용된다. Add-In 개발에서 거의 필수적인 부분이다.
- **Option Extension** : Add-In 자체에 다양한 선택 사항들을 둘 수 있는데, 이것을 활용하면 StarUML™의 옵션 다이얼로그를 그대로 사용하면서 옵션 항목을 설정할 수 있도록 한다.
- **Add-In COM Object** : Visual Basic, Delphi, Visual C++, C# 등과 같은 언어 및 도구들을 활용하여 확장할 기능을 구현할 수 있다. 주로 추가적인 GUI가 필요하거나 복잡한 기능을 구현할 때에 COM 개체로 구현할 수 있고 간단한 것은 Script로 해결할 수 있다. 이것은 주로 외부 API를 다루어 프로그래밍하게 된다.
- **Script** : 간단한 기능의 확장은 Scripting Language(JScript, VBScript, Python, ...)를 사용하여 작성할 수 있다. 이것은 주로 외부 API를 다루어 프로그래밍하게 된다.
- **Help** : Add-In에 관한 도움말을 HTML로 작성하여 원격주소 또는 로컬경로로 등록할 수 있다.

## Open API 개요

StarUML™은 방대한 Open API(Application Programming Interface)를 제공한다. StarUML™ 외부 API는 외부에서 프로그램 내부의 기능을 호출하여 사용할 수 있도록 하는 규격화된 프로그래밍 인터페이스이다.

StarUML™ 외부 API는 아래 그림과 같이 **Modeling Elements**, **Non\_Modeling Elements** 그리고 **Application Objects**의 크게 3 분류로 나눌 수 있다. **Modeling Elements** 부분은 모델링 요소에 대한 접근에 관한 인터페이스를 제공하고, **Non\_Modeling Elements**는 모델링 요소 이외의 여러 가지 요소들 및 MOF(Meta-Object Facility)에 관련된 인터페이스를 제공한다. **Application Objects**에는 애플리케이션 자체를 다루는 다양한 인터페이스들을 제공하고 있다.

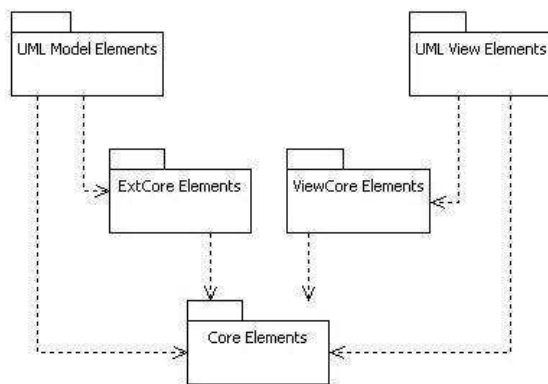


## Application Objects 부분

**Application Objects** 부분에는 애플리케이션 자체를 다루는 인터페이스들을 포함하고 있다. 이 부분에 포함되어 있는 인터페이스로는 기본 인터페이스인 **IPlasticApplication**, 요소들의 선택을 다루는 **ISelectionManager**, 요소들을 생성하는 **IUMLFactory**, 프로젝트를 관리하기 위한 **IProjectManager** 및 이벤트, GUI에 관한 인터페이스 등이 있다.

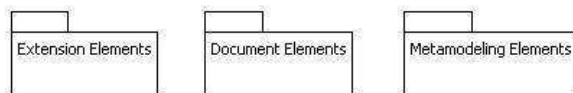
## Modeling Elements 부분

**Modeling Elements** 부분에는 모델링 요소들을 다루기 위한 인터페이스들을 포함하고 있다. 이 부분은 다시 여러 개의 부분으로 나눌 수 있다. **Core Elements** 부분은 모델 및 뷰 그리고 다이어그램 요소들의 최상위 인터페이스가 정의되어 있으며, **ExtCore Elements** 부분은 확장 가능한 모델 요소들에 관련된 인터페이스가 위치하고 있고 **UML Model Elements** 부분은 이것에 기반하여 UML 모델 요소들이 정의되어 있다. 그리고 **ViewCore Elements** 부분은 뷰 요소들의 기본 컴포넌트들에 대한 인터페이스들이 위치하고 있고, **UML View Elements** 부분에는 역시 이것에 기반하여 UML 뷰 요소들이 정의되어 있다.



## Non\_Modeling Elements 부분

**Non\_Modeling Elements** 부분에는 모델링 요소들을 제외한 기타 요소들에 대한 인터페이스들을 포함하고 있다. 이 부분은 다시 여러 개의 부분으로 나뉘는데 UML 확장 메커니즘에 관련된 요소들에 대한 인터페이스를 포함하는 **Extension Elements** 부분과 StarUML™의 저장 파일들을 다루는 **Document Elements** 부분 그리고 메타-수준의 요소들을 다루기 위한 **Metamodeling Elements** 부분으로 구성된다.



[처음으로](#)

## 제 3 장. HelloWorld 예제

이 장에서는 Hello, world 예제를 가지고 Add-In을 개발하는 방법과 과정을 간단하게 설명한다.

### "Hello, world" 예제

Hello, world 예제는 어떤 기술들을 익히기 위해 가장 처음으로 사용되는 예제이자 가장 쉬운 예제이다. 이 예제를 통해서 Add-In에 대한 감을 잡을 수 있도록 한다. Hello, world 예제는 모든 Add-In 구성 요소들을 사용하지 않고 가장 기본적인 것만을 사용하여, 다음과 같이 구성한다.

- 하나의 메뉴 확장(Menu Extension)
- 하나의 스크립트(Script)

Hello, world 예제는 메뉴 **[Hello, world!]**를 하나 추가하고 사용자가 해당 메뉴를 선택하면 프로젝트의 명칭(title)이 "Helloworld"로 바뀌도록



하는 기능을 추가하는 과정을 설명한다.

## 스크립트 작성하기

먼저 프로젝트의 명칭을 "Helloworld"로 바뀌게 하는 스크립트를 JScript 언어로 작성한다. 스크립트 소스를 다음과 같이 텍스트 편집기로 작성하고 **helloworld.js**로 저장한다.

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

스크립트의 첫 번째 라인은 **StarUMLApplication** 개체를 하나 생성한다. 이 개체는 **StarUML™**을 다루기 위한 시발점이 되는 개체 이므로 반드시 생성해야 한다. 두 번째 라인은 프로젝트(Project)에 해당하는 개체를 얻어오는 것이고, 세 번째 라인은 가져온 프로젝트 개체에 명칭(Title)을 "Helloworld"로 지정하라는 명령이다.

## 메뉴 확장 파일 작성하기

StarUML™의 메뉴를 확장하기 위해서는 먼저 메뉴 확장 파일(.mnu)을 작성해야 한다. 여기서는 **[Hello, world!]** 라는 메뉴를 **[Tools]** 메뉴 하위에 추가한다.

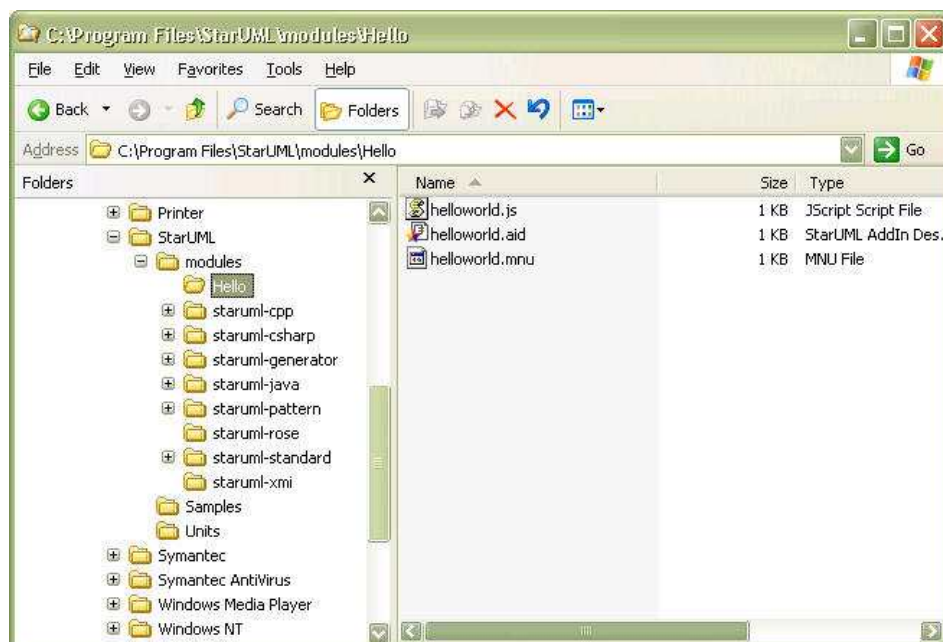
```
<?xml version="1.0"?>
<ADDINMENU addInID="StarUML.HelloWorldAddIn">
  <BODY>
    <MAINMENU>
      <MAINITEM base="TOOLS" caption="Hello, world!"
        availableWhen="PROJECT_OPENED"script="helloworld.js"/>
    </MAINMENU>
  </BODY>
</ADDINMENU>
```

메뉴 확장 파일은 <ADDINMENU> 태그로 시작하여 <HEADER>와 <BODY>로 구성된다. <HEADER> 부분은 생략 가능하고 <BODY> 부분에 메뉴 확장에 관한 내용이 기록되는데, 여기서는 메인 메뉴를 확장할 것이므로 <MAINMENU> 요소 하위에 <MAINITEM> 요소로 메뉴를 추가한다. 이 요소의 'base' 속성은 메뉴가 추가될 위치를, 'caption'은 메뉴의 이름, 'availableWhen'은 메뉴가 활성화되는 시점, 그리고 'script'는 메뉴가 눌러졌을 때 실행될 스크립트를 연결한다.

**노트:** 메뉴 확장에 관한 자세한 내용은 "**Chapter 8. Extending Menu**"를 참고하라.

## Add-In 배치

작성된 소크립트 파일(helloworld.js)과 메뉴 확장 파일(helloworld.mnu)을 특정 디렉토리에 함께 배치해야 한다. **StarUML™**을 설치한 디렉토리에 보면 "modules"라는 디렉토리가 있다. 이 디렉토리 하위에 "HelloworldAddIn"이라는 디렉토리를 만들고 두 파일을 여기에 위치시킨다.



## Add-In 등록

Add-In 파일들을 정상적으로 디렉토리에 배치했다면 **StarUML™**에서 Add-In을 인식할 수 있도록 Add-In Description 파일을 작성해야 한다. Add-In Description 파일은 확장자가 .aid 인 XML문서 파일로 여기에는 Add-In의 이름, COM 객체 이름, 실행 모듈 파일명, 메뉴 파일명, 도움말

URL 등 Add-In에 관한 전반적인 정보를 담고 있다. Add-In Description 파일 작성에 대한 자세한 내용은 "**Chapter 9. Writing Add-in COM Object**"를 참고하라.

다음은 HelloWorld 예제의 Add-In Description 파일이다.

```
<?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>
  <HELPPFILE>http://www.staruml.com</HELPPFILE>
  <ICONFILE>HelloWorld.ico</ICONFILE>
  <ISACTIVE>True</ISACTIVE>
  <MENUFILE>helloWorld.mnu</MENUFILE>
  <VERSION>1.0.1.35</VERSION>
</ADDIN>
```

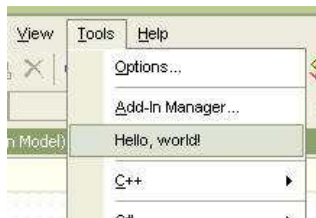
작성된 Add-In Description 파일을 Add-In을 배치한 디렉토리에 저장한다.

## Add-In 추가 확인 및 실행

위 과정을 정상적으로 수행했다면 StarUML™에 Hello, world Add-In이 추가되어 있을 것이다. StarUML™을 구동한 뒤 **[Tools]->[Add-In Manager]**를 선택하여 Add-In이 정상적으로 추가되었는지를 점검한다.



정상적으로 설치되었다면 **[Tools]** 메뉴 하위에 **[Hello, world!]** 메뉴가 추가되어 있는 것을 확인할 수 있다. 이 메뉴를 선택하면 **helloworld.js** 파일이 실행되어 프로젝트 명칭(Title)이 "Helloworld"로 변경될 것이다.



[처음으로](#)

## 제 4 장. Open API 사용하기

StarUML™은 UML메타모델과 애플리케이션 객체 등 프로그램의 대부분에 접근할 수 있도록 COM 객체화하고 API를 외부로 노출시켰다. 이 장에서는 StarUML™의 외부 API를 사용하는 방법을 자세하게 설명한다.

### 프로젝트 다루기

이 섹션에서는 StarUML™에서 프로젝트, 유닛 그리고 모델 조각들을 다루는 방법에 대해 소개한다.

#### 프로젝트 관리 기본 개념

프로젝트를 관리하기 위해서는 먼저 프로젝트에 관련된 개념(프로젝트, 유닛, 모델 조각)들을 잘 이해해야 한다.

##### 프로젝트

프로젝트는 StarUML™에서 다루는 가장 기본이 되는 단위이다. 프로젝트는 하나 혹은 그 이상의 소프트웨어 모델들을 관리할 수 있으며 항상 존재하는 최상위 패키지(Top-level Package)로도 이해될 수 있다. 하나의 프로젝트는 일반적으로 하나의 파일에 저장된다. 프로젝트는 다음과 같은 모델링 요소들만 포함하고 관리할 수 있다.

--	--

프로젝트 하위 요소	설명
모델(Model)	하나의 소프트웨어 모델을 관리하기 위한 요소.
서브시스템(Subsystem)	하나의 서브시스템을 표현한 요소들을 관리하기 위한 요소.
패키지(Package)	요소들을 관리하기 위한 가장 일반적인 요소.

프로젝트 파일은 XML 형태로 저장되며 확장명은 ".UML" 이다. StarUML™에서 작성된 모든 모델, 뷰, 다이어그램들은 하나의 프로젝트 파일에 저장되지만, 다음에 설명할 유닛(Unit)을 사용하면 프로젝트를 여러 파일에 나누어 저장할 수도 있다. 프로젝트 파일에는 다음과 같은 정보들이 저장된다.

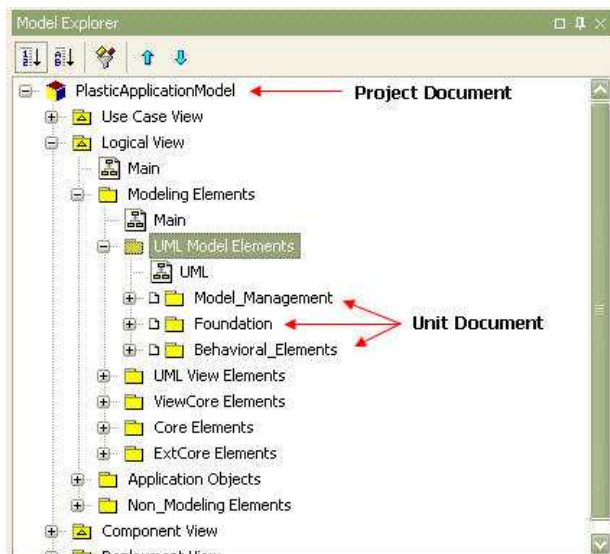
- 프로젝트가 참조하는 UML 프로파일들
- 프로젝트가 참조하는 유닛 파일들
- 프로젝트에 포함된 모든 모델 정보
- 프로젝트에 포함된 모든 다이어그램 및 뷰 정보

## 유닛

프로젝트는 기본적으로 하나의 파일에 저장되지만 프로젝트를 여러 명이 작업하거나 하는 등의 이유로 여러 개의 파일로 나누어서 다루어야 할 경우가 있다. 이러한 경우, 프로젝트를 여러 개의 유닛으로 만들어서 다룰 수 있도록 허용한다. 유닛은 계층적으로 구성될 수 있어서 유닛의 하부에 여러 개의 서브 유닛을 가질 수도 있다. 유닛은 ".UNT" 파일에 저장되며, 프로젝트 파일(.UML) 혹은 다른 유닛 파일(.UNT)로부터 참조된다.

패키지(Package), 서브시스템(Subsystem) 그리고 모델(Model) 요소만이 하나의 유닛이 될 수 있다. 이러한 패키지 류의 요소들의 하위에 포함된 모든 요소들은 해당 유닛 파일(.UNT)내에 저장된다.

프로젝트가 하위에 여러 개의 유닛을 관리할 수 있는 것처럼, 유닛 자체도 하위에 여러 개의 유닛을 관리할 수 있다. 상위 유닛은 하위 유닛들에 대한 참조를 가지게 되고 이러한 관계에 의하여 유닛은 계층구조(Hierarchical Structure)를 이루게 된다.



## 모델 조각

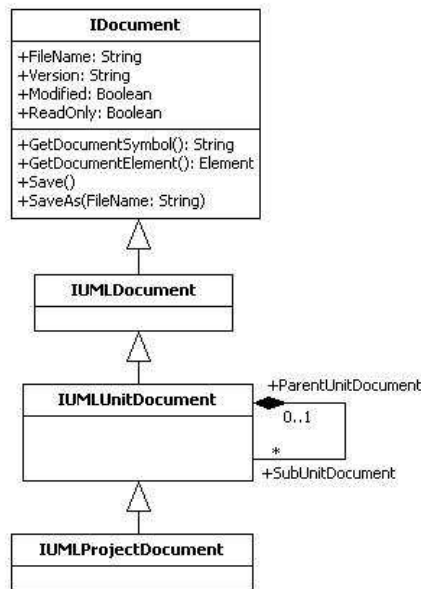
모델 조각은 프로젝트의 일부분을 별도의 파일로 저장한 것을 말한다. 모델 조각의 대상은 모델(Model), 서브시스템(Subsystem), 패키지(Package) 요소에만 해당되며 이것은 ".MFG"라는 확장명의 파일로 저장된다. 이렇게 저장된 모델 조각 파일은 언제든지 어떤 프로젝트에서 쉽게 포함시킬 수 있으며 일단 포함된 모델 조각은 프로젝트의 일부로 완전히 병합되므로 유닛과는 개념이 다르다.

## 도큐먼트 개체 다루기

### 도큐먼트의 개념

도큐먼트(Document)는 StarUML™에서 파일로 저장되는 단위를 추상화한 개체이다. 즉, 프로젝트(.UML)나 유닛(.UNT)과 같은 단위를 하나의 개체로 접근할 수 있도록 여러 가지 프로퍼티와 메소드를 제공한다. 모델 조각(.PMF)도 하나의 파일이지만 내보내기/가져오기와 같이 사용되어 실제 StarUML™ 애플리케이션 내부에서는 관리되지 않으므로 모델 조각에 대한 도큐먼트 개체는 존재하지 않는다. 다음은 도큐먼트 인터페이스들의 계층구조를 보여준다.





- **IDocument** : 도큐먼트에 대한 최상위 인터페이스이다.
- **IUMLDocument** : UML 모델에 관계된 도큐먼트에 대한 상위 인터페이스이다.
- **IUMLUnitDocument** : StarUML™에서 유닛(.UNT)으로 다루어지는 도큐먼트에 대한 인터페이스이다.
- **IUMLProjectDocument** : StarUML™에서 프로젝트(.UML)로 다루어지는 도큐먼트에 대한 인터페이스이다. 프로젝트 도큐먼트도 사실 하나의 유닛 도큐먼트로 간주되므로 유닛 도큐먼트 인터페이스에서 상속 받는다.

## 도큐먼트 개체에 접근하기

프로젝트 도큐먼트 혹은 유닛 도큐먼트 개체에 접근하기 위해서는 먼저 **IProjectManager** 개체의 참조를 얻어와야 한다. 그러면 직접 프로젝트 및 유닛의 도큐먼트 개체에 접근할 수 있다.

```

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);
}
  
```

**IProjectManager**를 통해서 도큐먼트에 접근할 수도 있지만, 특정 모델링 요소로부터 그것을 포함하는 도큐먼트 개체를 얻어올 수도 있다. 다음은 특정 요소로부터 프로젝트 도큐먼트 개체의 참조를 얻어와서 저장하는 예제이다.

```

var elem = ... // Assign specific element(i.e. Class, Package, etc)
var elem_doc = elem.GetContainingDocument();
elem_doc.Save();
  
```

## 도큐먼트 프로퍼티 및 메소드

**IDocument** 인터페이스는 다음과 같은 여러 가지 프로퍼티와 메소드를 제공한다.

프로퍼티	설명
FileName: String	도큐먼트의 파일명을 가져온다. 파일명은 전체 경로 및 확장명을 포함한다.
Version: String	도큐먼트의 버전을 가져온다.
Modified: Boolean	도큐먼트가 사용자에 의해 변경되었는지를 판단한다.
ReadOnly: Boolean	도큐먼트 파일이 읽기전용 상태인지를 판단한다.
메소드	설명
GetDocumentSymbol(): String	도큐먼트 심볼을 얻어온다. 프로젝트 도큐먼트인 경우는 'PROJECT' 유닛 도큐먼트인 경우는 'UNIT' 문자열을 반환한다.
GetDocumentElement(): IElement	도큐먼트에서 최상위의 요소를 반환한다.

Save()	도큐먼트를 현재의 파일명으로 저장한다.
SaveAs(FileName: String)	도큐먼트를 다른 파일명으로 저장하고 현재 파일명을 변경한다.

## 프로젝트 개체 다루기

### 프로젝트 개체에 접근하기

프로젝트를 직접 다루기 위해서는 먼저 프로젝트 개체에 대한 참조를 가져와야 한다. 다음은 그것을 하기 위한 **JScript** 코드이다.

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

애플리케이션 개체(**app**)에서 직접 얻을 수도 있지만 다음과 같은 방법으로 프로젝트 개체에 접근할 수도 있다.

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

### 프로젝트 명칭 및 프로퍼티 변경하기

프로젝트 개체에 대한 참조를 가져왔다면 프로젝트의 명칭 및 프로퍼티 그리고 다양한 메소드들을 호출할 수 있게 된다. 먼저 프로젝트의 명칭을 변경하기 위해서는 **"Title"**이라는 프로퍼티를 변경해야 한다. 그 외에 **'Copyright'**, **'Author'**, **'Company'** 등과 같은 프로퍼티도 같은 방법으로 변경할 수 있다.

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

**주의:** 일반적인 모델링 요소들의 이름은 **"Name"**이라는 프로퍼티를 사용하는데, 프로젝트 개체에는 **"Name"** 프로퍼티를 사용하지 말아야 한다. 프로젝트는 최상위 패키지로써 이름을 가지지 않는다. 이름을 가지지 않는 이유는 요소들 간의 참조를 위해서 경로명(**Pathname**)을 흔히 사용하게 되는데, 프로젝트의 명칭을 변경하면 모든 경로명이 깨어지기 때문이다.

### 프로젝트 하위에 패키지 추가하기

프로젝트 하위에는 모델(**Model**), 서브시스템(**Subsystem**), 패키지(**Package**) 요소만이 추가될 수 있다. 요소를 새로 생성하여 추가하기 위해서는 **IUMLFactory** 개체를 사용해야 한다. 프로젝트 하위에 패키지를 추가하는 방법은 다음 예제를 참고하라.

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

### 새 프로젝트 만들기

새 프로젝트를 만들 때에는 **IProjectManager** 개체의 참조를 가져와서 **NewProject** 메소드를 호출한다.

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

새 프로젝트를 만들 때 빈(**empty**) 프로젝트를 만들지 않고 특정 접근법(**Approach**)을 적용하여 생성하고자 하는 경우에는 **NewProjectByApproach** 메소드를 사용한다. 다음 예는 **"UMLComponents"** 접근법을 사용하여 새 프로젝트를 만든 것을 보여준다.

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

### 프로젝트 열기

파일로 저장되어 있는 프로젝트 파일(.UML)을 열고자 할 때에는 **IProjectManager** 개체의 참조를 가져와서 **OpenProject** 메소드를 사용한다.

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

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

## 프로젝트 저장하기

현재 StarUML™에서 열려있는 프로젝트를 저장하기 위해서는 **IProjectManager** 개체의 참조를 가져온 다음 **SaveProject** 메소드를 사용한다. 만약, 다른 이름으로 저장하고자 할 때에는 **SaveProjectAs** 메소드를, 그리고 프로젝트의 모든 하위 유닛들을 저장하고자 할 때에는 **SaveAllUnits** 메소드를 사용한다.

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

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

## 프로젝트 닫기

프로젝트를 더 이상 사용하지 않게 위해서 닫을 때에는 **IProjectManager** 개체의 참조를 가져와서 **CloseProject** 메소드를 사용한다.

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

var prjmgr = app.ProjectManager;
prjmgr.CloseProject();
```

## 유닛 다루기

### 새로운 유닛 분할하기

특정 패키지, 모델 혹은 서브시스템을 별도의 파일로 관리기 위해 새로운 유닛으로 분할하고자 한다면 **IProjectManager** 개체의 참조를 얻어와서 **SeparateUnit** 메소드를 사용한다.

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

var prjmgr = app.ProjectManager;

var pkg = ... // Assign reference for the package to separate as a new unit.
var new_unit = prjmgr.SeparateUnit(pkg, "NewUnit.unt");
```

### 유닛 병합하기

이미 유닛으로 분리되어 있는 패키지, 모델 혹은 서브시스템을 더 이상 별도의 파일로 관리하지 않기 위해서 병합하고자 한다면 **IProjectManager** 개체의 참조를 얻어와서 **MergeUnit** 메소드를 사용한다.

```
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);
```

### 하위 유닛에 접근하기

유닛은 계층적으로 구성될 수 있다. 프로젝트 하위에 여러 개의 유닛이 있을 수 있으며 그 유닛 하위에 또 다시 여러 개의 하위 유닛이 존재할 수 있다. 특정 유닛 하위에 존재하는 유닛들에 접근하기 위해서는 다음의 예를 참고하라.

```
var unit = ... // Assigns reference for the unit that contains sub-units to access.

for (var i = 0; i < unit.GetSubUnitDocumentCount(); i++) {
    var sub_unit = unit.GetSubUnitDocumentAt(i);
    ...
}
```

## 모델 조각 다루기

### 패키지를 모델 조각으로 만들기

패키지, 모델 혹은 서브시스템을 별도의 모델 조각 파일로 저장할 수 있다. 다음과 같이 **IProjectManager** 개체의 참조를 가져와서 **ExportModelFragment** 메소드를 사용한다.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var prjmgr = app.ProjectManager;
var pkg = ... // Assigns package to make as a model.
prjmgr.ExportModelFragment(pkg, "MyFragment.mfg");
```

## 모델 조각 가져오기

파일로 존재하는 모델 조각을 특정 패키지, 모델, 서브시스템 혹은 프로젝트에 포함시킬 수 있다. 다음과 같이 **IProjectManager** 개체의 참조를 가져와서 **ImportModelFragment** 메소드를 사용한다.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var prjmgr = app.ProjectManager;
var pkg = ... // Assigns package to add a model fragment.
prjmgr.ImportModelFragment(pkg, "MyFragment.mfg");
```

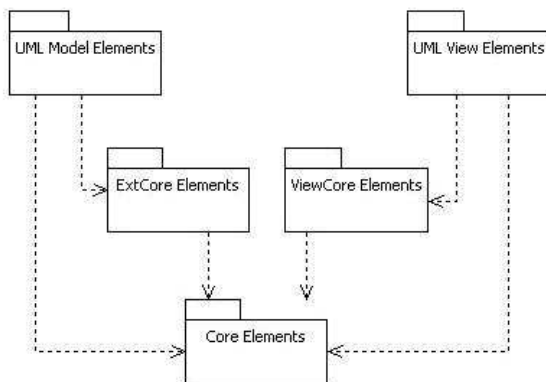
## 모델링 요소 다루기

이 섹션에서는 **StarUML™** 외부 API 중에서 모델링 요소에 해당하는 인터페이스 타입들을 소개하고, 이들을 어떻게 사용하는지를 소개할 것이다. 모델링 요소는 **UML**을 사용하여 소프트웨어 모델링을 할 때 사용하게 되는 **UML 모델(Model)**, **뷰(View)**, 그리고 **다이어그램(Diagram)** 요소들을 의미한다. 예를 들어 패키지, 클래스, 액터 등과 같은 모델 요소들과 각 모델 요소에 대응하는 뷰 요소들, 그리고 클래스 다이어그램, 유스 케이스 다이어그램 등과 같은 다이어그램 요소들이 여기에 해당한다. 모델링 요소에 대한 외부 API를 사용하여 모델, 뷰, 다이어그램 요소들을 생성, 삭제, 또는 수정할 수 있다.

**참고:** UML 모델링 요소에 대한 완전한 목록은 "**Appendix B. UML 모델링 요소 목록**"에서 확인할 수 있다.

## 모델링 요소의 구성

모델링 요소는 다음과 같은 논리적인 그룹으로 구분되어 있다.



- **Core Elements:** Core Elements 그룹은 모델, 뷰, 다이어그램 요소들의 최상위 인터페이스를 정의하고 있다.
- **ExtCore Elements:** ExtCore Elements 그룹은 확장 가능한 모델 요소들의 공통 상위 인터페이스를 정의하고 있다.
- **ViewCore Elements:** ViewCore Elements 그룹은 뷰 요소들의 기반 타입들을 정의하고 있다.
- **UML Model Elements:** UML 모델 요소들을 정의하고 있다. UML 표준 모델링 요소들에 해당한다.
- **UML View Elements:** UML View Elements 그룹은 UML 뷰 요소들을 정의하고 있다.

모델링 요소는 크게 **모델**, **뷰**, 그리고 **다이어그램** 타입으로 구분할 수 있다. 그런데 이 중에서 다이어그램 타입은 실제로는 모델 및 뷰 타입의 일부이기 때문에 **모델**과 **뷰** 타입으로 구분하는 것이 더 정확하다. 모델은 소프트웨어 모델에 대한 실제 정보를 가지고 있는 요소이며, 뷰는 특정 모델이 담고 있는 정보를 시각적으로 표현하는 수단이다. 하나의 모델은 여러 개의 뷰를 가질 수 있으며, 뷰는 일반적으로 하나의 모델을 참조하게 된다.

## 모델링 요소를 사용하는 간단한 예제

모델링 요소들에 대한 외부 API 인터페이스들을 소개하기 전에, 모델링 요소 사용에 대한 힌트를 제공하기 위한 간단한 예제를 소개한다. **StarUML™** 애플리케이션의 최상위 프로젝트(**Project**) 요소에서부터 패키지, 클래스, 인터페이스 등과 같은 네임스페이스(**Namespace**) 타입 요소들과 각 네임스페이스 타입 요소의 하위 요소들을 탐색(**Trace**)하려 한다고 가정하자. 이런 경우에 모델링 요소들의 구조를 이용해야 하는데, 먼저 이 기능을 구현한 아래의 JScript 코드를 보자.

```
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++){
```

```

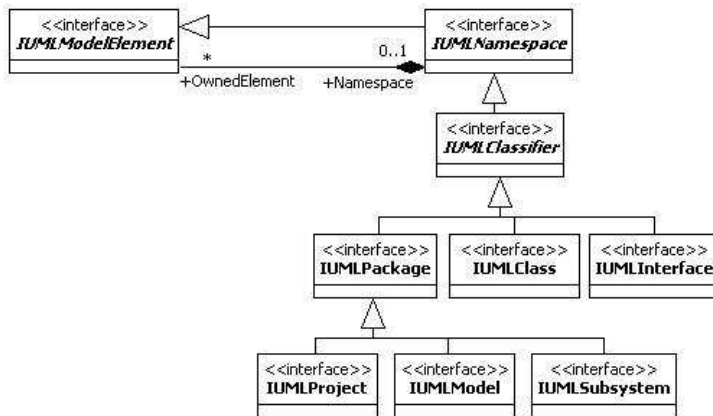
    elem = owner.GetOwnedElementAt(i);
    ...

    if (elem.IsKindOf("UMLNamespace")) VisitOwnedElement(elem);
}
}

```

이 예제는 위에서 가정한 기능을 구현하기 위해, 최상위 프로젝트 요소에서부터 "OwnedElement" 관계에 있는 하위 요소들을 재귀적(Recursive)으로 얻어오고 있다. 이 코드의 핵심은 **VisitOwnedElement** 라는 사용자 정의 함수라고 할 수 있는데, 이 함수는 모델링 요소의 하나인 **IUMLNamespace** 타입의 요소를 인자로 받아 **IUMLNamespace** 인터페이스의 메소드인 **GetOwnedElementCount**, **GetOwnedElementAt**을 사용하고 있다.

**VisitOwnedElement** 함수를 구현하는데 대한 힌트는 모델링 요소들의 관계에서 얻을 수 있다. 아래의 그림은 StarUML™ 외부 API 중에서 **IUMLNamespace** 인터페이스를 중심으로 위 예제와 관련 있는 인터페이스 타입들과의 관계를 요약해서 그린 것이다.



**IUMLNamespace** 인터페이스는 **IUMLModelElement**에서 상속되는데, 이것은 **IUMLPackage**, **IUMLClass**, **IUMLInterface** 타입 등에 대한 공통 상위 타입이다. 그리고 **IUMLNamespace**는 **IUMLModelElement**와 **Namespace-OwnedElement**라는 연관(Association) 관계를 맺고 있다. 이것을 보면 **IUMLPackage**, **IUMLClass** 등과 같은 **IUMLNamespace** 타입 모델링 요소들은 하위에 **IUMLModelElement** 타입 요소들을 **OwnedElement** 관계로 포함할 수 있음을 알 수 있을 것이다. 이처럼 모델링 요소들의 관계에 따라 외부 API의 모델링 요소 부분 인터페이스 구조가 정의된다.

**노트:** 표준 UML 요소에 해당하는 모델링 요소의 이름은 표준 UML 요소의 이름 앞에 접두어 "UML"을 붙인 형태이다. 예를 들어 UML의 한 요소인 **Actor**의 경우 **UMLActor**가 된다. 그리고 외부 API에서는 코딩 관례에 따라 접두어 'I'를 붙혀 **IUMLActor**와 같이 사용된다. UML 모델링 요소의 목록과 각 요소의 이름에 대한 내용은 "**Appendix B. UML 모델링 요소 목록**"을 참고할 수 있다.

## 외부 API에서 연관관계를 표현하는 규칙

위의 그림에서 **IUMLModelElement** 인터페이스와 **IUMLNamespace** 인터페이스 타입은 **OwnedElement-Namespace** 연관관계를 가지고 있다. StarUML™ 외부 API의 인터페이스 정의에서 이와 같은 연관관계는 참조로 표현되는데, 예를 들어 **IUMLModelElement** 인터페이스에는 **Namespace** 연관관계가 다음과 같이 표현된다.

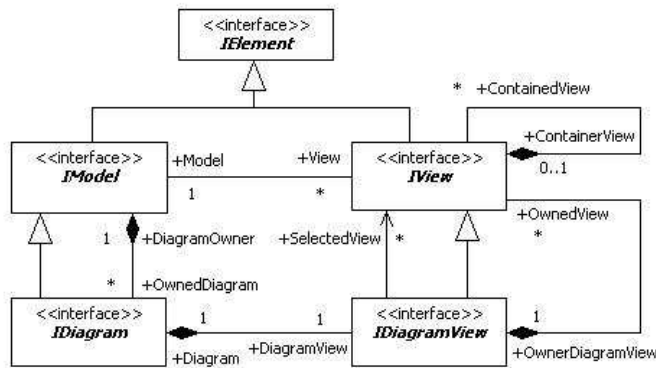
<b>IUMLModelElement</b>
Namespace: IUMLNamespace

그리고 **IUMLNamespace** 인터페이스에는 **OwnedElement** 연관관계가 다음과 같이 표현되는데, 이것은 다중성(Multiplicity)이 복수 개(\*) 이기 때문에 프로그램 내부 구현에서 집합 또는 리스트 구조를 사용하기 때문이다. 외부 API의 인터페이스 정의에서 연관관계는 모두 이와 같은 규칙에 따라 표현되므로 **IUMLModelElement-IUMLNamespace** 외의 다른 인터페이스들에도 똑같이 적용된다.

<b>IUMLNamespace</b>
function GetOwnedElementCount(): Integer;
function GetOwnedElementAt(Index: Integer): IUMLModelElement;

## Core 요소들

Core Elements 그룹의 Core 요소들은 모델링 요소들에 대한 최상위 부모 인터페이스들이다. 여기에는 **IElement**, **IModel**, **IView**, **IDiagram**, **IDiagramView** 인터페이스가 있으며 아래의 그림과 같은 구조를 가진다. Core 그룹의 인터페이스 타입들은 일반적으로 많이 사용되며 중요한 역할을 하므로 아래의 구조를 주의 깊게 살펴볼 필요가 있다. 특히 인터페이스들의 연관관계를 잘 살펴보기 바란다.



인터페이스 타입	설명
IElement	모든 모델링 요소들의 공통 최상위 요소를 정의하는 인터페이스 타입.
IModel	모델 요소들의 공통 부모 요소를 정의하는 인터페이스 타입.
IView	뷰 요소들의 공통 부모 요소를 정의하는 인터페이스 타입.
IDiagram	다이아그램 모델 요소들의 공통 부모 요소를 정의하는 인터페이스 타입.
IDiagramView	다이아그램 뷰 요소들의 공통 부모 요소를 정의하는 인터페이스 타입.

## IElement

**IElement** 인터페이스는 모든 모델링 요소들의 공통 최상위 타입을 정의하는 인터페이스이며, 다음과 같은 주요한 메소드들을 제공한다.

주요 메소드	설명
GetGUID(): String	모델링 요소의 GUID (Global Unique Identifier)를 반환하는 함수. GUID는 Base64 형식으로 인코딩되어 있다.
GetClassName(): String	모델링 요소의 클래스 이름을 반환하는 함수. 반환 값 예: "UMLClass"
IsKindOf(ClassName: String): Boolean	모델링 요소가 인자로 받은 것과 같은 타입의 요소인가를 검사하는 함수. 인자 값 예: "UMLClass"
IsReadOnly(): Boolean	모델링 요소가 읽기전용 상태인지 검사하는 함수. 읽기 전용인 상태의 모델링 요소는 속성값을 수정할 수 없다.
MOF_GetAttribute(Name: String): String	모델링 요소의 기본 타입 속성들 중에서 인자의 이름이 지정하는 속성의 값을 문자열로 반환한다.
MOF_GetReference(Name: String): IElement	모델링 요소의 참조 타입 속성들 중에서 인자의 이름이 지정하는 속성의 값(개체 참조)을 반환한다.
MOF_GetCollectionCount(Name: String): Integer	모델링 요소의 참조 컬렉션 타입 속성들 중에서 인자의 이름이 지정하는 참조 컬렉션의 아이템 개수를 반환한다.
MOF_GetCollectionItem(Name: String; Index: Integer): IElement	모델링 요소의 참조 컬렉션 속성들 중에서 인자의 이름이 지정하는 참조 컬렉션의 Index 번째 아이템에 해당하는 값(개체 참조)을 반환한다.

**IElement** 인터페이스의 메소드 중에서 **MOF\_XXX** 메소드들은 문자열(String) 이름으로 각 모델링 요소의 속성값에 접근할 수 있는 일관성 있는 방법을 제공한다. 먼저 예를 하나 들면, **IElement**의 서브 타입인 **IUMLModelElement**는 "Visibility"라는 속성을 가진다. 일반적으로 이 속성의 값을 알기 위해서는 **IUMLModelElement.Visibility** 와 같은 형식을 사용하게 되는데, 아래와 같이 **IElement.MOF\_GetAttribute** 메소드를 사용하면 "Visibility" 라는 문자열 이름으로 해당 속성의 값을 얻어올 수 있다. **MOF\_XXX** 메소드는 이처럼 각 모델링 요소의 기본 타입/참조 타입/참조 컬렉션 타입의 속성들에 대해 문자열 이름으로 접근할 수 있도록 하며, 이것은 때때로 매우 유용하게 사용될 수 있다.

**노트:** **MOF\_XXX** 메소드들의 인자인 속성의 문자열 이름은 해당 속성의 이름과 동일하다.

다음의 예제는 **IUMLModelElement** 타입 요소에서 **IElement.MOF\_GetAttribute** 메소드를 사용하여 요소의 "Visibility" 속성의 값을 읽고 있다. 주의할 것은, **MOF\_GetAttribute** 메소드는 반환 값으로 문자열을 사용한다는 것이다. 이 예제에서 반환 값은 "vkPrivate", "vkPublic" 등이 될 수 있다.

```

...
var elem = ... // Get reference to IUMLModelElement type element object.
var val = elem.MOF_GetAttribute("Visibility");
...

```

모델링 요소의 참조(Reference) 타입 속성의 값을 읽을 때는 **IElement.MOF\_GetReference** 메소드를 사용한다. **MOF\_GetReference** 메소드는 **IElement** 타입 개체의 참조를 반환한다. 다음 예제는 **IUMLModelElement** 타입 요소의 "Namespace" 참조 속성의 값을 읽고 있다.

```

...

```



```
var elem = ... // Get reference to IUMLModelElement type element object.
var refElem = elem.MOF_GetReference("Namespace");
...
```

모델링 요소의 참조(Reference) 컬렉션 타입 속성의 값을 읽을 때는 **IElement.MOF\_GetCollectionItem** 메소드를 사용한다.

**MOF\_GetCollectionItem** 메소드는 참조 컬렉션 타입 속성의 이름과 아이템 인덱스(Index)를 인자로 받는데, 컬렉션 아이템의 개수는 **MOF\_GetCollectionCount** 메소드를 통해 알 수 있다. 그리고 **MOF\_GetCollectionItem** 메소드는 **MOF\_GetReference** 메소드와 마찬가지로 **IElement** 타입 개체의 참조를 반환한다. 다음 예제는 **IUMLClassifier** 타입 요소의 "Attributes" 참조 컬렉션 속성의 값을 읽고 있다.

```
...
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);
    ...
}
```

**주의:** **MOF\_XXX** 메소드들의 인자 값으로 존재하지 않는 속성 이름이 지정되면 에러가 발생한다.

## IModel

**IModel** 인터페이스는 모델 요소들의 공통 부모 타입을 정의하는 인터페이스 타입으로, 다음과 같은 주요한 프로퍼티들과 메소드들을 제공한다.

주요 프로퍼티	설명
Name: String	이름 속성.
Documentation: String	문서화 속성.
Pathname: String	모델 요소의 경로이름. 경로이름은 최상위 프로젝트 요소를 제외한 모든 상위요소들의 이름이 '::'로 구분된 형태이다. 경로이름 예: "::Application Model::Modeling Elements::UML Model Elements". ※ 읽기전용.
주요 메소드	설명
AddAttachment(Attach: String);	첨부파일 속성에 값(파일경로, URL)을 추가한다.
FindByName(AName: String): IModel	하위 모델요소들 중에서 인자로 받은 이름과 동일한 이름의 요소가 있으면 반환한다.
FindByRelativePathname(RelPath: String): IModel	중첩된 하위 모델요소들 중에서 인자 값이 지정하는 상대경로이름과 일치하는 모델요소가 있으면 반환한다. 인자 값에는 해당 모델의 이름은 포함시키지 않는다. 인자 값 예: "Model_Management::UMLPackage"
ContainsName(AName: String): Boolean	하위 모델요소들 중에 인자로 받은 이름과 동일한 이름의 요소가 이미 있는지 검사한다.
CanDelete(): Boolean	해당 모델요소가 읽기전용 상태인지 검사한다.
GetViewCount: Integer	해당 모델의 뷰 요소의 개수를 반환한다.
GetViewAt(Index: Integer): IView	해당 모델의 뷰 요소들 중에서 Index 번 째 요소를 반환한다.
GetOwnedDiagramCount: Integer	해당 모델이 포함하고 있는 다이어그램 요소의 개수를 반환한다.
GetOwnedDiagramAt(Index: Integer): IDiagram	해당 모델이 포함하고 있는 다이어그램 요소 중에서 Index 번 째 요소를 반환한다.

아래의 예제는 모델 요소의 기본적인 속성 값들을 읽고 다시 값을 설정하는 것을 보여주는 것이다.

```
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");
        }
    }
}
```

모델 요소의 하위 요소를 찾을 경우 **FindByName** 메소드와 **FindByRelativePathname** 메소드를 사용할 수 있다. **FindByName** 메소드는 하위 요소들 중에서 인자로 받은 스트링 값과 이름이 동일한 첫 번째 하위 요소를 반환한다. **FindByName** 메소드는 모델 요소의 바로 밑 하위 요소에 대해서만 검색을 수행하는데, 하위 요소들이 중첩된 구조로 되어있는 경우 모든 하위 요소들에 대해 검색을 수행하려면

**FindByRelativePathname** 메소드를 사용해야 한다. 다음 예제는 **FindByName**과 **FindByRelativePathname** 메소드를 사용하는 방법을 보이고 있다.

```
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");
```

위의 그림에서 보인 바와 같이 **IModel** 인터페이스와 **IView** 인터페이스는 **Model-View** 연관관계를 맺고 있다. **IModel** 타입 요소는 여러 개의 **IView** 타입 요소를 가질 수 있으며 **IView** 타입 요소는 하나의 **IModel** 타입 요소를 가져야 한다. 아래의 예제는 각 **IUMLClass** 타입 요소에 대한 모든 **IView** 타입 요소 참조를 얻는 방법을 보여주고 있다.

```
var elem = ... // Get reference to IModel type element.

if (elem.GetClassName() == "UMLClass"){
    for (var i = 0; i < elem.GetViewCount(); i++){
        var view = elem.GetViewAt(i);
        ...
    }
}
```

위의 그림에서 보인 바와 같이 **IModel** 인터페이스와 **IDiagram** 인터페이스는 **DiagramOwner-OwnedDiagram** 연관관계를 맺고 있다. **IDiagram** 인터페이스는 모든 다이어그램 모델 타입들의 부모 타입이기 때문에 아래의 예제와 같은 방식으로 모델 요소가 포함하고 있는 다이어그램 요소들의 참조를 얻을 수 있다.

```
var elem = ... // IModel type element

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

## IView

**IView** 인터페이스는 뷰 요소들의 공통 부모 타입을 정의하는 인터페이스로 다음과 같은 주요한 프로퍼티들을 제공한다.

주요 프로퍼티	설명
LineColor: String	선의 색상을 지정한다. BGR 형식을 사용한다. 예: "0xff0000" (blue); "0x00ff00" (green); "0x0000ff" (red); "0x000000" (black); "0xffffffff" (white)
FillColor: String	채움 색상을 지정한다. BGR 형식을 사용한다.
FontFace: String	폰트를 지정한다. 예: "Times New Roman"
FontColor: String	폰트의 색상을 지정한다. BGR 형식을 사용한다.
FontSize: String	폰트의 크기를 지정한다.
FontStyle: Integer	폰트의 스타일을 지정한다. 정수 1 (bold), 2 (italic), 3 (underline), 4 (strikeout)를 사용하거나 조합하여 사용할 수 있다. 예: 1 + 2 (bold & italic) ※ 기본적인 스타일이 정해져 있는 뷰 요소에는 적용되지 않는다.
Selected: Boolean	해당 뷰 요소가 현재 선택된 상태인지를 지정한다. ※ 읽기전용.
Model: IModel	해당 뷰 요소에 대응하는 모델 요소에 대한 참조를 지정한다. ※ 읽기전용.
OwnerDiagramView: IDiagramView	해당 뷰 요소가 포함되어 있는 다이어그램 뷰 요소를 지정한다. ※ 읽기전용.

아래의 예제는 **IView** 타입 요소의 기본적인 속성 값들을 설정하는 것을 보여주는 것이다.

```
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;
```

**IUMLNoteView**, **IUMLNoteLinkView**, **IUMLTextView** 타입 뷰 요소를 제외한 뷰 요소들은 모델 요소에 대한 참조를 가지고 있다. **IView** 타입 요소가 참조하고 있는 **IModel** 타입 요소의 정보를 얻으려면 다음과 같이 할 수 있다.

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

**IView** 타입 요소가 속해 있는 다이어그램 정보를 얻을 때는 아래의 코드와 같이 할 수 있다.

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

## IDiagram

**IDiagram** 인터페이스는 **IModel** 인터페이스에서 상속되며 모든 다이어그램 타입 모델 요소들의 공통 부모 타입이다. **IDiagram** 인터페이스는 다음과 같은 주요 프로퍼티들을 가진다.

주요 프로퍼티	설명
DefaultDiagram: Boolean	해당 다이어그램이 <b>Default Diagram</b> 인지를 지정한다. <b>Default Diagram</b> 은 프로젝트가 열릴 때 자동으로 열리는 다이어그램을 의미하며, 클래스/유스 케이스/컴포넌트/디플로이먼트 다이어그램에만 지정할 수 있다.
DiagramOwner: IModel	해당 다이어그램이 포함되어있는 상위 모델 요소를 지정한다. ※ 읽기전용
DiagramView: IDiagramView	해당 다이어그램 모델에 대응하는 다이어그램 뷰 요소를 지정한다. ※ 읽기전용

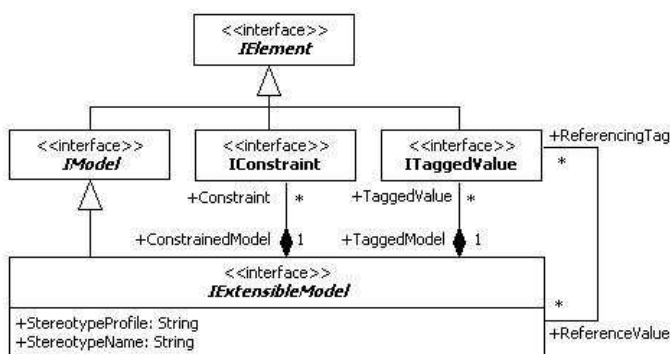
## IDiagramView

**IDiagramView** 인터페이스는 **IView** 인터페이스에서 상속되며 모든 다이어그램 뷰 요소들의 공통 부모 타입이다.

주요 프로퍼티	설명
Diagram: IDiagram	해당 다이어그램 뷰 요소에 대응하는 다이어그램 모델 요소를 지정한다. ※ 읽기전용
주요 메소드	설명
GetSelectedViewCount: Integer	다이어그램에서 현재 선택된 상태에 있는 뷰 요소들의 개수를 반환한다.
GetSelectedViewAt(Index: Integer): IView	다이어그램에서 현재 선택된 상태에 있는 뷰 요소들 중에서 <b>Index</b> 번 째 요소를 반환한다.
GetOwnedViewCount: Integer	다이어그램에 포함되어 있는 뷰 요소들의 개수를 반환한다.
GetOwnedViewAt(Index: Integer): IView	다이어그램에 포함되어 있는 뷰 요소들 중에서 <b>Index</b> 번 째 요소를 반환한다.
LayoutDiagram()	다이어그램을 자동배치시킨다.
ExportDiagramAsBitmap(FileName: String)	다이어그램을 <b>Bitmap</b> 이미지로 변환하여 해당 경로와 파일명으로 저장한다.
ExportDiagramAsMetafile(FileName: String)	다이어그램을 <b>Windows Metafile</b> 로 변환하여 해당 경로와 파일명으로 저장한다.
ExportDiagramAsJPEG(FileName: String)	다이어그램을 <b>JPEG</b> 이미지로 변환하여 해당 경로와 파일명으로 저장한다.

## ExtCore 요소들

ExtCore 요소들은 UML 확장 기능을 적용할 수 있는 모델 요소들에 대한 기반 구조를 제공한다. UML 확장 기능이 적용되는 모든 모델 요소들은 **IExtensibleModel** 인터페이스에서 상속된다. **IExtensibleModel** 인터페이스는 다음 그림과 같이 여러 개의 제약사항(**Constraint**)과 태그값(**Tagged Value**)을 가질 수 있다.



인터페이스 타입	설명
IExtensibleModel	UML 확장 기능을 적용할 수 있는 모델 요소들의 공통 상위 타입.
IConstraint	제약사항 요소.
ITaggedValue	태그값 요소.

## IExtensibleModel

**IExtensibleModel** 인터페이스는 다음과 같은 주요 프로퍼티 및 메소드를 정의하고 있다.

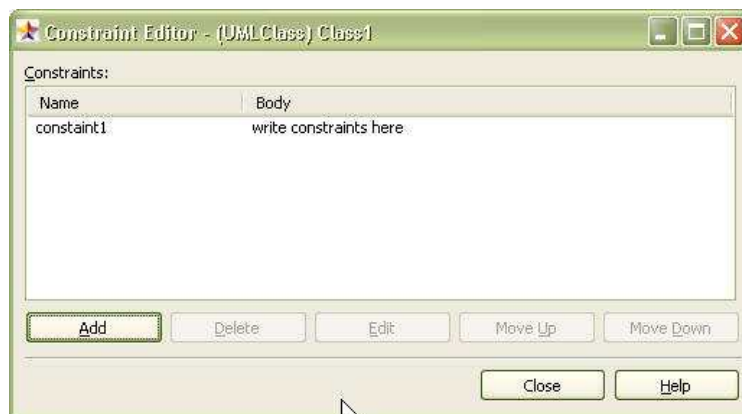
주요 프로퍼티	설명
StereotypeProfile: String	해당 모델 요소에 적용된 스테레오타입이 정의되어 있는 UML 프로파일 이름을 지정한다. ※ 읽기전용
StereotypeName: String	해당 모델 요소에 적용된 스테레오타입 이름을 지정한다. ※ 읽기전용
주요 메소드	설명
GetConstraintCount: Integer	해당 모델이 포함하고 있는 제약사항 요소의 개수를 반환한다.
GetConstraintAt(Index: Integer): IConstraint	해당 모델이 포함하고 있는 제약사항 요소 중에서 Index 번 째 요소를 반환한다.
AddConstraint(Name: String; Body: String): IConstraint	제약사항 요소를 인자가 정하는 이름과 값으로 생성한다.
IndexOfConstraint(AConstraint: IConstraint): Integer	인자가 지정하는 제약사항 요소의 인덱스를 반환한다.
DeleteConstraint(Index: Integer)	해당 모델이 포함하고 있는 제약사항 요소 중에서 Index 번 째 요소를 삭제한다.
GetTaggedValueCount: Integer	해당 모델이 포함하고 있는 태그값 요소의 개수를 반환한다.
GetTaggedValueAt(Index: Integer): ITaggedValue	해당 모델이 포함하고 있는 태그값 요소 중에서 Index 번 째 요소를 반환한다.
GetStereotype: IStereotype	해당 모델에 적용된 스테레오타입 요소를 반환한다.
SetStereotype(const Name: WideString)	<b>IStereotype</b> 요소가 아니라 단지 문자열로 스테레오타입 값을 지정한다.
SetStereotype2(Profile: String; Name: String)	스테레오타입이 정의된 UML 프로파일과 스테레오타입 값을 같이 지정한다.

스테레오타입이나 태그값의 경우 원칙적으로 UML 프로파일을 통해 정의되어야 한다. 그렇지만 StarUML™에서는 스테레오타입의 경우 UML 프로파일에 익숙하지 않은 사용자들을 위해 문자열 값으로 설정할 수 있도록 허용하고 있다. 아래의 예제는 어떤 **IExtensibleModel** 타입 요소에서 스테레오타입 값을 읽고 다시 설정하는 것을 보여주고 있다.

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

스테레오타입의 경우와 달리 태그값의 경우에는 반드시 UML 프로파일을 통해 정의된 것만 사용할 수 있다. UML 프로파일, 스테레오타입, 태그값에 대한 자세한 설명은 "**Chapter 7. Writing UML Profiles**"를 참고하라.

## IConstraint



제약사항은 StarUML™ 애플리케이션에서 위의 그림과 같이 제약사항 편집기를 통해서 추가하거나 편집할 수 있는데, 외부 API에서는 **IConstraint** 인터페이스를 사용하여 제약사항을 추가하거나 편집할 수 있다. **IConstraint** 인터페이스는 다음과 같은 프로퍼티를 제공한다.

주요 프로퍼티	설명
Name: String	제약사항의 이름.
Body: String	제약사항의 내용.
ConstrainedModel: IExtensibleModel	이 제약사항이 적용된 IExtensibleModel 타입 요소.

제약사항 요소는 **IExtensibleModel** 타입 요소가 제공하는 메소드를 통해 생성할 수 있는데, 아래의 예제는 어떤 **IExtensibleModel** 타입 요소에 제약사항을 추가하고 편집하고 삭제하는 것을 보여준다.

```
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);
```

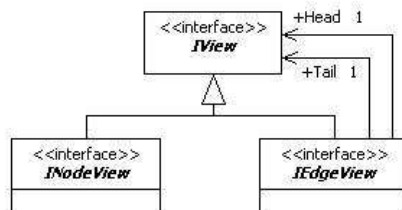
## ITaggedValue

**ITaggedValue** 인터페이스는 태그값 요소를 정의하는 인터페이스로 다음과 같은 프로퍼티와 메소드들을 제공한다. 태그값 요소에 대한 자세한 설명은 "**Chapter 7. Writing UML Profiles**"를 참고하기 바란다.

주요 프로퍼티	설명
ProfileName: String	해당 태그값이 정의된 UML 프로파일 이름을 지정한다. ※ 읽기전용
TagDefinitionSetName: String	해당 태그값이 포함되어 있는 태그 정의 집합 (Tag Definition Set)을 지정한다. ※ 읽기전용
Name: String	UML 프로파일에서 정의된 해당 태그값의 이름을 지정한다. ※ 읽기전용
DataValue: String	태그값의 값을 지정한다. ※ 읽기전용
TaggedModel: IExtensibleModel	해당 태그값이 적용된 IExtensibleModel 타입 요소에 대한 참조를 지정한다. ※ 읽기전용
주요 메소드	설명
GetTagDefinition: ITagDefinition	해당 태그값에 대한 태그정의 요소를 반환한다.
GetTagDefinitionSet: ITagDefinitionSet	해당 태그값에 대한 태그정의집합 요소를 반환한다.
GetProfile: IProfile	해당 태그값이 정의된 UML 프로파일 요소를 반환한다.

## ViewCore 요소들

ViewCore 그룹의 인터페이스 타입들은 **IView** 인터페이스에서 상속 받으며 모든 뷰 타입 요소들에 대한 기반 구조를 제공한다. ViewCore 그룹에는 많은 인터페이스 타입들이 포함되는데 여기서는 가장 중요한 **INodeView**, **IEdgeView** 인터페이스를 중심으로 설명한다.



주요 타입	설명
INodeView	Node 타입 뷰들의 최상위 인터페이스 타입.
IEdgeView	Edge 타입 뷰들의 최상위 인터페이스 타입.

## INodeView

**INodeView** 인터페이스는 Node 타입 뷰 요소들에 대한 기반 타입이다. Node 타입 뷰는 **Class View**와 같이 면적을 가지는 형태의 뷰 요소를 의미한다. **INodeView** 인터페이스는 다음과 같은 주요한 프로퍼티를 제공한다.

주요 프로퍼티	설명
Left: Integer	뷰의 위치 정보 (Left).
Top: Integer	뷰의 위치 정보 (Top).
Width: Integer	뷰의 크기 정보 (Width).
Height: Integer	뷰의 크기 정보 (Height).
MinWidth: Integer	해당 뷰 요소의 최소 크기를 지정한다 (Width). ※ 읽기전용
MinHeight: Integer	해당 뷰 요소의 최소 크기를 지정한다 (Height). ※ 읽기전용

<b>AutoSize: Boolean</b>	해당 뷰 요소의 자동크기 속성을 지정한다.
--------------------------	-------------------------

다음 예제는 **INodeView** 타입 뷰의 위치와 크기를 변경하는 예를 보이고 있다.

```
var nodeView = ... // Get reference to INodeView type element.
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

**IEdgeView** 인터페이스는 **Edge** 타입 뷰 요소들에 대한 기반 타입이다. **Edge** 타입 뷰는 **Dependency View**와 같은 선 형태의 뷰 요소를 의미한다. **IEdgeView** 인터페이스는 다음과 같은 주요한 프로퍼티를 제공한다.

주요 프로퍼티	설명
<b>LineStyle: LineStyleKind</b>	선 스타일을 지정한다.
<b>Points: IPoints</b>	선의 좌표정보를 지정한다.
<b>Tail: IView</b>	선의 시작 지점이 되는 뷰 요소를 지정한다.
<b>Head: IView</b>	선의 끝 지점이 되는 뷰 요소를 지정한다.

Edge 타입 뷰의 선 스타일은 **LineStyleKind** 열거체(Enumeration)에서 정의된 다음 값들을 사용할 수 있다.

값	설명
<b>IsRectilinear</b>	직교선 형태의 선 스타일.
<b>IsOblique</b>	꺾은선 형태의 선 스타일.

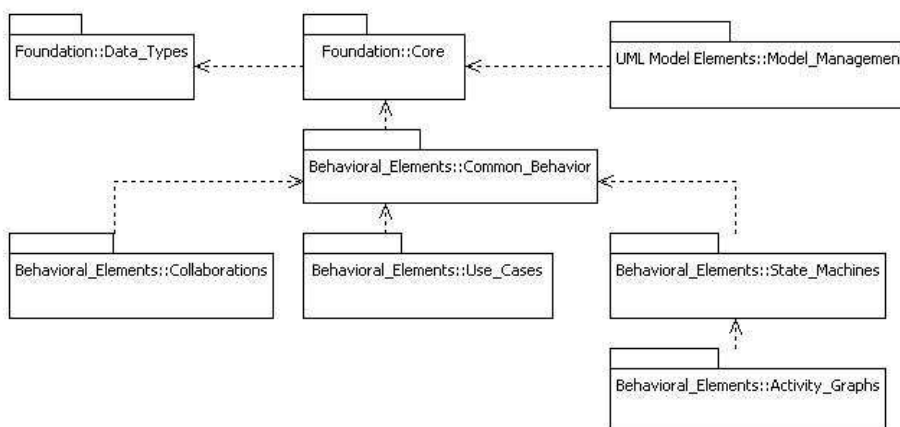
다음 예제는 **Edge** 타입 뷰의 선 스타일을 변경하는 예를 보이고 있다.

```
IsRectilinear = 0;
IsOblique = 1;

var view = ... // Get reference to view element.
if (view.IsKindOf("EdgeView")){
    view.LineStyle = IsRectilinear;
}
```

## UML 모델 요소 다루기

UML Model Elements 그룹은 다시 다음과 같은 패키지들로 그룹화되어 있다. 참고로 UML Model Elements 그룹에 정의된 UML 모델 요소들은 UML 표준 명세에서 정의하고 있는 표준 UML 요소들에 대한 StarUML™의 구현으로, 표준 UML 요소들과 거의 동일하다. 여기서는 UML Model Elements 그룹에 있는 구체적인 UML 모델 요소들에 대한 설명은 하지 않는다.



## UML 모델 요소 생성하기

UML 모델 요소를 생성할 때는 **IUMLFactory** 인터페이스를 사용해야 한다. **IUMLFactory** 인터페이스는 UML 모델 요소 뿐만 아니라 UML 다이



어그럼 요소 및 UML 뷰 요소들과 같은 모든 UML 모델링 요소들에 대한 생성 메소드들을 제공한다. **IUMLFactory** 타입 개체는 다음과 같이 **IStarUMLApplication** 타입 개체를 통해 얻을 수 있다.

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

**IUMLFactory**는 다음과 같은 UML 모델 요소 생성 메소드들을 제공한다.

UML 모델 요소	생성 메소드
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
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: 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

다음 예제는 **IUMLFactory**를 사용하여 UML 모델 요소들을 생성하는 예를 보이고 있다.

```
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;
...
```

## UML 모델 요소 삭제하기

UML 모델 요소를 삭제할 때는 **IStarUMLApplication** 인터페이스의 **DeleteModel** 메소드를 사용하면 된다. 모델 요소를 삭제하기 전에 먼저 **IModel** 인터페이스의 **CanDelete** 메소드를 사용하여 해당 모델 요소가 삭제 가능한 상태인지 검사할 수 있다. 해당 모델 요소가 읽기전용 상태인 경우 **CanDelete** 메소드는 "false"를 반환할 것이다. 어떤 모델 요소를 삭제했을 때 해당 모델 요소 하위의 모든 모델 요소들이 같이 삭제되며, 해당 모델 요소와 연관된 모든 뷰 요소들도 자동으로 삭제된다는 점을 기억해야 한다. 다음 예제는 위의 모델 요소 생성 예제에 연결하여 클래스 요소 하나를 삭제하는 것을 보여준다.

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

## UML 다이어그램 다루기

### UML 다이어그램 요소 생성하기

UML 모델 요소 생성과 같이 UML 다이어그램 요소를 생성할 때도 **IUMLFactory**를 사용하는데, **IUMLFactory**의 다이어그램 관련 생성 메소드는 다음과 같은 것이 있다.

UML 다이어그램 요소	생성 메소드
UMLClassDiagram	CreateClassDiagram(AOwner: Model): IUMLClassDiagram
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

UML 다이어그램 요소의 생성 방법은 위에서 보인 UML 모델 요소 생성과 거의 동일한데, UML 다이어그램 요소는 모델 타입 요소가 생성될 때 관련된 뷰 타입 요소도 자동으로 생성된다는 차이점이 있다. 아래의 예는 UML 다이어그램 요소를 생성하고 자동으로 생성된 UML 다이어그램 뷰 요소에 접근하는 것을 보이고 있다.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var pkgElem = ... // Upper level model element to contain UML diagram element.

var dgmElem = facto.CreateClassDiagram(pkgElem); // Create UMLClassDiagram.
var dgmViewElem = dgmElem.DiagramView; // Automatically created diagram view element.
app.OpenDiagram(dgmElem);
...
```

### UML 다이어그램 요소 삭제하기

UML 다이어그램 요소도 UML 모델 요소들 중의 하나이므로 UML 모델 요소의 삭제와 마찬가지로 **IStarUMLApplication** 인터페이스의 **DeleteModel** 메소드를 사용하여 삭제할 수 있다. 그리고 삭제 가능 여부를 검사하려면 **IModel** 인터페이스의 **CanDelete** 메소드를 사용하면 된다.

## UML 뷰 요소 다루기

### UML 뷰 요소 생성하기

뷰 요소를 생성할 때도 **IUMLFactory**를 사용한다. 뷰 요소 생성과 관련된 **IUMLFactory** 메소드 목록은 다음과 같다.

UML 뷰 요소	생성 메소드
UMLNoteView	CreateNoteView(ADiagramView: DiagramView): IUMLNoteView
UMLNoteLinkView	CreateNoteLinkView(ADiagramView: DiagramView; ANote: UMLNoteView; LinkTo: View): IUMLNoteLinkView
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
	CreateComponentInstanceView(ADiagramView: DiagramView; AModel:

UMLComponentInstanceView	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: 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

다음 예제는 클래스 다이어그램 뷰에 두 개의 **IUMLClassView** 타입 요소를 생성하고, 이 두 요소를 연결하는 **IUMLDependencyView**와 **IUMLAssociationView**를 생성하는 것을 보이고 있다. 뷰 요소를 생성하기 위해서는 해당 뷰 요소에 대한 모델 요소가 존재해야 하므로 먼저 각 모델 요소를 먼저 생성시키고 있다.

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

// Get reference to existing model elements.
var rootElem = app.FindByPathname("::Logical View");
if (rootElem != null) {
    app.BeginUpdate();
    try{
        // Create model elements.
        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;

        // Create view elements.
        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);

        // Adjust view element attributes.
        classView1.Left = 100;
        classView1.Top = 100;
        classView2.Left = 300;
        classView2.Top = 100;
        app.OpenDiagram(diagram);
    }
    finally{
        app.EndUpdate();
    }
}
```

## UML 뷰 요소 삭제하기

UML 뷰 요소를 삭제할 때는 **IStarUMLApplication** 인터페이스의 **DeleteView** 메소드를 사용하면 된다. 모델 요소를 삭제했을 때는 해당 모델 요소와 연관된 모든 뷰 요소들도 자동으로 삭제되지만, 뷰 요소를 삭제하는 경우 연관된 모델 요소는 삭제되지 않는다는 점에 주의할 필요가 있다.

다음 예제는 위의 예제에서 생성했던 뷰 요소들을 삭제하는 것을 보여주고 있다.

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

## 애플리케이션 자동화 개체 다루기

### 애플리케이션 개체 다루기

#### StarUMLApplication Object

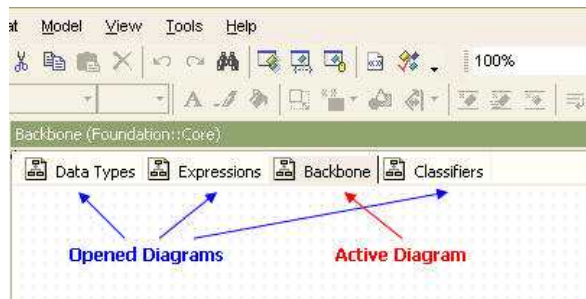
StarUML™의 외부 API를 사용하기 위해서 가장 먼저 얻어와야 하는 것이 바로 **StarUMLApplication** 개체의 참조이다. 모든 다른 개체들은 이것을 통해서 접근될 수가 있다. **IStarUMLApplication** 인터페이스에는 StarUML™ 애플리케이션 자체에 대한 추상화인데 다음과 같은 메소들을 직접 포함하고 있다.

- 사용자 액션 관련 (Undo, Redo, ClearHistory, BeginUpdate, EndUpdate, BeginGroupAction, EndGroupAction, ...)
- 요소 편집 관련 (Copy, Cut, Paste, ...)
- 모델, 뷰 및 다이어그램 삭제 관련 (DeleteModel, DeleteView, ...)

- 옵션 항목의 값 읽기 (GetOptionValue)
- 로그, 메시지 및 웹 브라우징 관련 (Log, AddMessageItem, NavigateWeb, ...)
- 열린 다이어그램 관리 (OpenDiagram, CloseDiagram, ...)
- 기타 (FindByPathname, SelectInModelExplorer, ...)

## 열린 다이어그램 관리하기

StarUML™의 다이어그램 영역에서는 열려져 있는 다이어그램(**Opened Diagram**)들이 다음 그림과 같이 탭(tab)으로 관리된다. 그 중에 현재 활성화 되어 있는 다이어그램을 **Active Diagram**이라고 한다.



특정 다이어그램을 열기 위해서는 다음 예제와 같이 수행한다. 열려져 있지 않은 다이어그램이라면 다이어그램이 열리면서 자동으로 활성화 다이어그램(**Active Diagram**)이 된다. 만약 이미 열려져 있는 다이어그램이라면 활성화 다이어그램으로 설정된다.

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

열려져 있는 다이어그램들에 대한 참조들을 가져오고 싶다면 **GetOpenedDiagramCount** 와 **GetOpenedDiagramAt** 메소드를 사용하여야 한다.

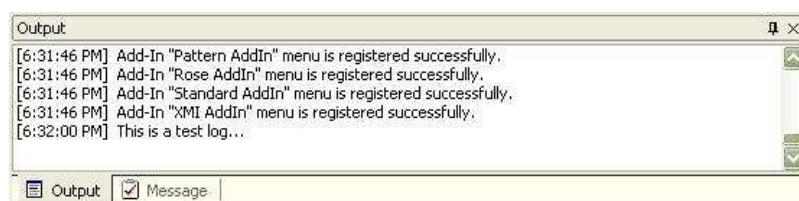
```
var app = new ActiveXObject("StarUML.StarUMLApplication");
...
for (i=0; i<app.GetOpenedDiagramCount(); i++) {
    var dgm = app.GetOpenedDiagramAt(i);
    ...
}
```

열려져 있는 다이어그램을 닫을 수도 있다. 이 때에는 **CloseDiagram** 메소드를 사용하면 되는데, 모든 다이어그램을 닫아야 하거나 (**CloseAllDiagram**), 활성화 다이어그램만 닫고 싶은 경우(**CloseActiveDiagram**)에는 그에 맞는 메소드를 사용하여야 한다.

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

## 로그 남기기

StarUML™ 정보 영역의 **[출력]** 탭에는 애플리케이션의 수행 로그를 기록하여 사용자에게 보여준다. 이 부분에 로그를 남길 수 있는 인터페이스를 제공한다.



**[출력]** 부분에 로그를 남기기 위해서는 **Log** 메소드를 다음 예제와 같이 사용하면 된다.

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

## 메시지 항목 다루기

StarUML™에서 특정 메시지를 사용자에게 알려주고 싶은 경우에는 메시지 항목을 사용한다. 메시지 항목은 요소 검색에서 발견되거나 모델 검증을 통과하지 못한 내용과 요소 등을 알려주는데 사용된다. 메시지 항목은 다음의 표와 같이 총 3 종류가 있는데, 일반 항목, 요소 검색 결과 항목, 그리고 모델 검증 결과 항목이다.

--	--	--



값	리터럴	설명
0	mkGeneral	일반적인 메시지 항목.
1	mkFindResult	요소 검색 결과를 위한 메시지 항목.
2	mkVerificationResult	모델 검증 결과를 위한 메시지 항목.

메시지 항목을 하나 추가할 때에는 메시지 항목의 종류, 메시지 내용 그리고 연관된 요소의 참조를 인자로 넘겨주어야 한다. 다음은 3가지 종류의 메시지를 서로 다른 메시지 내용으로 프로젝트 요소를 참조하도록 추가하는 예제이다. 그 결과는 다음의 그림과 같다.

```
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());
```



메시지를 더블 클릭(double click)하게 되면 연관된 요소가 자동으로 모델 탐색기에 선택되고 다이어그램에 표현되었다면 해당 다이어그램이 활성화된다.

### 경로명으로 요소 찾기

경로명(pathname)으로 특정 요소를 찾을 수 있다. 예를 들어 **Package1** 아래 **Package2** 아래에 있는 **Class1** 요소의 경로명은 **"::Package1::Package2::Class1"**이다. 경로명은 이름(Name)을 "::" 구분자(delimiter)로 연결한 것인데 최상위 프로젝트에서 부터 찾기 시작하고 최상위 프로젝트의 이름은 항상 빈 문자열(null string)이므로 항상 "::"으로 시작하게 된다. 그러나, 항상 처음에 붙는 "::"는 생략해도 무방하다. 즉, **"Package1::Package2::Class2"**로 써도 동일한 경로명이 된다. 다음은 경로명으로 모델 요소의 참조를 얻어오는 예제이다.

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

### 애플리케이션 업데이트 조정하기

사용자가 특정 작성을 수행하거나 API를 통해 특정 명령을 수행하게 되면 그 즉시 **StarUML™**은 변경 내용을 업데이트하여 사용자에게 보여주게 된다. 그러나, API를 통해 복잡한 작업을 수행하기 위해서는 명령을 여러 번 한꺼번에 수행해야 하고 이런 경우 각각의 작업이 즉시 즉시 반영되어 보기에 좋지 않을 뿐 아니라 속도도 매우 저하 된다. 이런 경우에는 변경 내용의 업데이트를 잠시 중단하고 복잡한 작업들을 처리한 다음에 변경 내용을 한꺼번에 반영하는 것이 자연스럽다. **StarUMLApplication** 개체는 이러한 기능을 제공하기 위해 **BeginUpdate**와 **EndUpdate** 메소드를 제공한다.

복잡하고 방대한 처리 작업을 수행하기 전에 **BeginUpdate** 메소드를 호출한 다음 수행히 끝난 직후에 **EndUpdate** 메소드를 호출하면 이 때 모든 변경된 내용들이 반영되게 된다. **BeginUpdate**를 호출한 이후 작업을 처리하다 오류 혹은 문제가 발생하게 되어 **EndUpdate**가 호출되지 않는다면 그 후 어떠한 변경 작업도 반영되지 않으므로 주의해야 한다. 이러한 것을 방지하기 위해서 예외 처리 기법(특히 try ... finally)을 다음 예제와 유사하게 잘 활용하기 바란다.

```
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.
}
...
```

변경의 마지막임을 알리고 변경 내용을 반영시키는 메소드는 **EndUpdate** 외에 **EndUpdate2**가 하나 더 존재한다. 이 두 가지 중 어떠한 것을 사용해도 무방한데, **EndUpdate2**는 더 세부적인 조절이 가능한 메소드이다. 이 메소드는 다음 두 가지의 인자를 통해 더 세부적인 조절을 가능하게 한다.

인자	타입	설명
CompletelyRebuild	Boolean	모델 탐색기에 나타나는 모든 트리 구조를 처음부터 새로 생성한다. 처리되는 내용이 매우 방대한 모델 요소의 생성이나 변경이 가해진다면 이 인자 값을 'True'로 하는 것이 훨씬 빠른 변경의 반영이 될 수도 있다. <b>EndUpdate()</b> 메소드에서는 이 값이 'False' 인 것과 효과가 동일하다.
		모델 탐색기에 반영되는 트리 항목들의 삽입/삭제/변경 등을 수행할 때 역시 한꺼번에 반영하도록 한다. 즉, GUI 상에서 트리 항목의 변경이 시각적으로 보이지 않고 한번에 처리가 된다.

UseUpdateLock	Boolean	매우 방대한 모델이 구축되어 있는 상황에서 이 값을 'True'로 사용하게 되면 비록 적은 수의 모델 요소가 반영 되더라도 비교적 오랜 시간이 소요될 수 있다. 이 값을 'True'로 한 경우 수행에 걸리는 시간은 변경된 모델 요소의 개수보다는 전체 모델 요소 개수에 비례한다. <b>EndUpdate()</b> 메소드에서는 이 값이 'True' 인 것과 효과가 동일하다.
---------------	---------	---

## 그룹 액션 사용하기

사용자가 수행한 액션은 되돌리기(Undo)와 다시 실행(Redo)이 가능하다. API를 통해 실행한 명령도 동일한데, 만약 명령을 2번 실행했다면 실행 전으로 돌아가기 위해서는 Undo를 두 번 수행해야 될 것이다. 그러나, 여러 가지 상황에서 여러 명령들의 조합이 하나의 액션으로 처리되기를 바라는 경우가 많다. 예를 들면, 특정 속성(Attribute)에 대해 자동으로 Get 함수와 Set 함수를 추가하는 코드를 작성하는 경우에는 사용자가 코드를 실행한 이후에 Undo를 수행하면 Get 함수와 Set 함수가 추가되지 전으로 돌아가야 한다. 그러나 Get/Set 함수의 추가를 위해서는 여러 개의 명령을 조합해서 실행해야만 한다. 이런 경우에 여러 개의 명령들을 하나의 그룹으로 다루어 마치 하나의 액션인 것처럼 처리하도록 할 수 있다.

**StarUMLApplication** 개체는 **BeginGroupAction**과 **EndGroupAction** 메소드를 사용하여 여러 개의 명령들이 하나의 액션처럼 수행될 수 있도록 허용한다. **BeginGroupAction** 메소드를 호출하면 새로운 가상의 그룹 액션(Group Action)이 생성된다. 그리고 그 이후에 처리되는 작업들은 해당 그룹 액션에 포함되고 **EndGroupAction** 메소드를 호출하면 액션의 그룹화를 마감하게 된다. **BeginGroupAction**이 실행된 이후에는 그룹에 포함될 처리 작업에서 오류가 발생하더라도 반드시 **EndGroupAction**이 호출되어야 하므로 예외 처리(특히 try ... finally)를 확실하게 다루어 주어야 한다. 이렇게 처리된 그룹 액션은 한번의 Undo와 Redo를 통해서도 마치 하나의 액션처럼 동작하게 될 것이다.

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

**BeginGroupAction** 메소드를 호출하면 동시에 **BeginUpdate**를 호출한 것과 같은 상태가 된다. 마찬가지로 **EndGroupAction** 메소드를 호출하면 **EndUpdate**를 호출한 것과 같은 상태가 된다. 즉, 그룹이 완전히 결정되기 전까지는 변경 내용이 반영되지 않는다. 따라서, **BeginGroupAction**과 **EndGroupAction** 사이에는 **BeginUpdate**와 **EndUpdate** 메소드를 사용해서는 안 된다.

## 요소 선택 다루기

StarUML™에서는 사용자가 선택한 모델 혹은 뷰 요소에 대한 정보를 가져오거나 강제로 특정 요소를 선택하도록 하는 방법들을 제공한다. 이러한 것에 관련된 기능들은 **ISelectionManager** 인터페이스에 모두 정의되어 있다.

### 선택된 요소 얻어오기

현재 선택되어 있는 모델 혹은 뷰 요소들에 대한 목록을 가져오기 위해서는 먼저 **SelectionManager** 개체에 대한 참조를 얻어와야 한다. 그런 다음 아래 예제와 같이 선택된 모델 요소들이나 뷰 요소들에 대한 참조를 가져올 수 있다.

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

// List selected model elements.
for (i=0; i<selmgr.GetSelectedModelCount(); i++) {
    var m = selmgr.GetSelectedModelAt(i);
    ...
}

// List selected view elements.
for (i=0; i<selmgr.GetSelectedViewCount(); i++) {
    var v = selmgr.GetSelectedViewAt(i);
    ...
}
```

### 현재 활성 다이어그램 얻어오기

현재 활성 다이어그램(StarUML™에서 화면에 보여지고 있는 다이어그램)의 참조를 가져올 수 있다. 다이어그램은 항상 **Diagram** 개체와 **DiagramView** 개체의 두 가지로 나누어져 관리되는데 현재 활성 다이어그램의 **Diagram** 개체 및 **DiagramView** 개체 두 가지 모두를 바로 얻을 수 있다.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var selmgr = app.SelectionManager;
var dgm = selmgr.ActiveDiagram // Diagram object of the currently active diagram
var dgmview = selmgr.ActiveDiagramView // DiagramView object of the currently active diagram
```

## 모델 요소 선택하기

특정 모델 요소(e.g. Class, Interface, Component, ...)를 선택하려면 **SelectModel** 메소드를 사용한다. 이 메소드를 호출하면 기존에 선택되어

있는 모든 요소들의 선택을 해제하고 해당 모델 요소 하나만을 선택하게 된다. 만약, 기존의 선택 요소들을 유지하면서 추가적으로 모델 요소를 선택하고자 한다면 **SelectAdditionalModel** 메소드를 사용해야 한다.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var selmgr = app.SelectionManager;
var m = ... // Assign reference to model element to select.
...
selmgr.SelectModel(m); // Select only the model element 'm'.
...
selmgr.SelectAdditionalModel(m); // Add model element 'm' to selection.
...
```

모델 요소의 선택을 해제하고 싶을 때에는 다음 예와 같이 **DeselectModel** 메소드를 사용한다.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var selmgr = app.SelectionManager;
var m = ... // Assign reference to model element to deselect.
...
selmgr.DeselectModel(m); // Deselect model element 'm'.
...
selmgr.DeselectAllModels(); // Deselect all model elements.
...
```

## 뷰 요소 선택하기

다이아그램에 그려져 있는 뷰 요소들을 선택하려면 **SelectView** 메소드를 사용한다. 이 메소드를 호출하면 기존에 선택되어 있는 모든 뷰 요소들의 선택이 해제되고 해당 뷰 요소 하나만을 선택하게 된다. 만약, 기존의 뷰 요소들의 선택을 유지하면서 추가적인 뷰 요소를 선택하고 싶다면 **SelectAdditionalView** 메소드를 사용해야 한다.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var selmgr = app.SelectionManager;
var v = ... // Assign reference to model element to select.
...
selmgr.SelectView(v); // Select only the model element 'm'.
...
selmgr.SelectAdditionalView(v); // Add model element 'm' to selection.
...
```

뷰 요소의 선택을 해제하고 싶을 때에는 다음 예와 같이 **DeselectView** 메소드를 사용한다.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var selmgr = app.SelectionManager;
var v = ... // Assign reference to view element to add.
...
selmgr.DeselectView(v); // Select only the view element 'v'.
...
selmgr.DeselectAllViews(); // Add view element 'v' to the selection.
...
```

## 다이아그램의 영역 선택하기

현재 활성 다이어그램의 특정 영역의 좌표를 입력하여 영역에 포함되는 뷰 요소들을 선택할 수 있는 방법을 제공한다. 이것은 **SelectArea** 메소드를 사용하거나 추가적으로 영역을 선택하고자 하는 경우에는 **SelectAdditionalArea** 메소드를 사용하면 된다. 다음 예제는 현재 활성 다이어그램에서 영역 (100, 100, 500, 300)에 포함되는 모든 뷰 요소들을 선택한다.

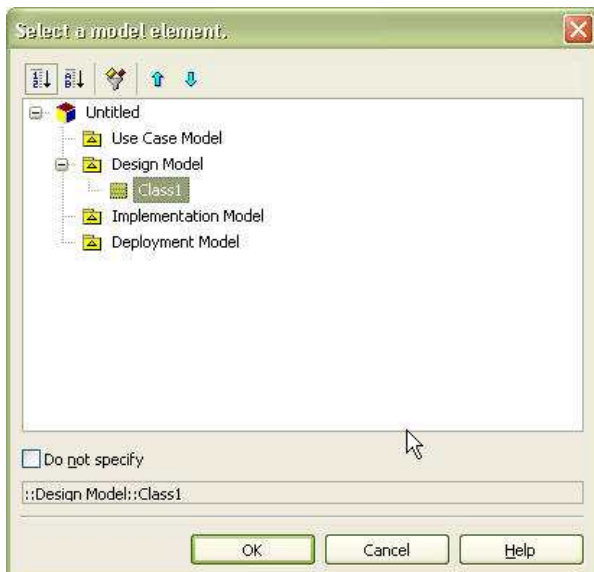
```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var selmgr = app.SelectionManager;
selmgr.SelectArea(100, 100, 500, 300);
```

## 요소 선택 다이얼로그 다루기

StarUML™에서는 특정 요소를 선택하도록 도와주는 다이얼로그에 2가지 종류를 제공하고 있다. 트리 뷰(Tree View) 형태의 것 (**ElementSelector**)과 리스트 뷰(List View) 형태의 것(**ElementListSelector**)이 있는데 일반적으로 모델 탐색기와 동일한 형태로 트리 구조에서 요소를 선택하도록 하는 **ElementSelector**가 주로 사용되고 동일한 종류의 요소들을 열거하여 선택할 수 있도록 하는 것에는 **ElementListSelector**가 사용된다.

### ElementSelector 개체 다루기

**ElementSelector**는 다음의 그림과 같이 모델 탐색기와 동일한 형태의 트리 구조를 나타내어 사용자에게 하나의 요소를 선택할 수 있도록 하는 다이얼로그이다. 요소를 선택할 수도 있고 아무것도 지정하지 않도록(Null 값을 지정)할 수도 있다.



**ElementSelector** 다이얼로그 개체에 대한 참조는 다음과 같이 **StarUMLApplication** 개체를 통해 얻을 수 있다.

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

**ElementSelector** 다이얼로그는 다음과 같은 프로퍼티 및 메소드들을 제공한다.

주요 프로퍼티	설명
AllowNull: Boolean	아무것도 지정하지 않는 것(Null 값을 지정)을 허용할 지를 지정.
주요 메소드	설명
Filter(Filtering: ElementFilteringKind)	어떤 타입의 모델링 요소들을 보여줄 것인가를 지정한다. 다음 값들 중의 하나를 지정할 수 있다. fkAll (0): 모든 모델링 요소를 보인다. fkPackages (1): UMLPackage 타입 요소(UMLPackage, UMLModel, UMLSubsystem) 요소만을 보인다. fkClassifiers (2): UMLClassifier 타입 요소만을 보인다.
ClearSelectableModels	선택 가능한 요소 타입 목록을 초기화 시킨다.
AddSelectableModel(className: String)	선택 가능한 요소 타입 목록에 지정된 타입을 추가한다. 인자 값 예: "UMLClass"
RemoveSelectableModel(className: String)	선택 가능한 요소 타입 목록에서 지정된 타입을 삭제한다. 인자 값 예: "UMLClass"
Execute(title: String): Boolean	다이얼로그를 실행시킨다. 인자로 다이얼로그의 타이틀 문자열을 지정한다.
GetSelectedModel: IModel	사용자에 의해 선택된 요소에 대한 참조를 반환한다.

다음 예는 **ElementSelector** 다이얼로그를 실행시키고 사용자에게 의해 선택된 요소를 얻는 전 과정을 보여준다.

```
fkClassifiers = 2;

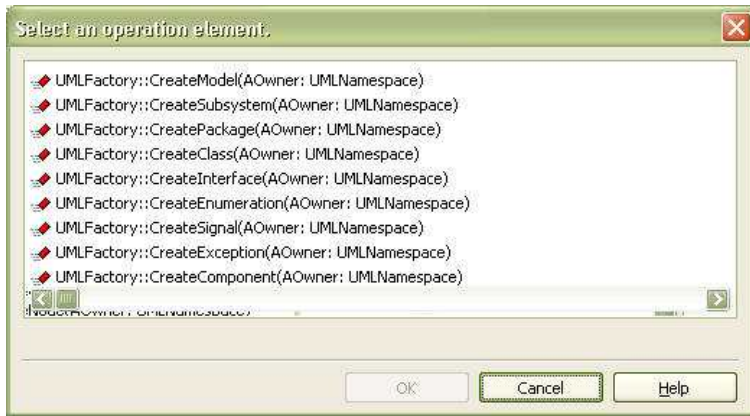
var app = new ActiveXObject("StarUML.StarUMLApplication");
var sel_dlg = app.ElementSelector;

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, ...
}
```

## ElementListSelector 개체 다루기

**ElementListSelector**는 다음의 그림과 같이 선택 가능한 요소들의 목록을 리스트 뷰(List View) 형태로 나타내어 사용자가 하나의 요소를 선택할 수 있도록 하는 다이얼로그이다.



**ElementListSelector** 다이얼로그 개체에 대한 참조는 다음과 같이 **StarUMLApplication** 개체를 통해 얻을 수 있다.

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

**ElementListSelector** 다이얼로그는 다음과 같은 프로퍼티 및 메소드들을 제공한다.

주요 프로퍼티	설명
AllowNull: Boolean	아무것도 지정하지 않는 것(Null 값을 지정)을 허용할 지를 지정.
주요 메소드	설명
ClearListElements	목록을 초기화 시킨다.
AddListElement(AModel: IModel)	인자가 지정하는 모델 요소를 목록에 추가한다.
AddListElementsByCollection(AModel: IModel; CollectionName: String; ShowInherited: Boolean)	인자가 지정하는 모델 요소의 지정된 컬렉션 항목들의 목록을 추가한다. 'ShowInherited' 인자는 지정된 모델 요소의 상속구조를 따라 가면서 상위 요소들에 대해서도 지정된 컬렉션 항목들을 목록에 추가할 것인지를 지정한다.
AddListElementsByClass(MetaClassName: String; IncludeChildInstances: Boolean)	인자가 지정하는 타입의 요소들을 목록에 추가한다. 'IncludeChildInstances' 인자가 'true' 이면 지정하는 타입에서 파생되는 요소들도 목록에 추가된다.
Execute(Title: String): Boolean	다이얼로그를 실행시킨다. 인자로 다이얼로그의 타이틀 문자열을 지정한다.
GetSelectedModel: IModel	사용자에 의해 선택된 요소에 대한 참조를 반환한다.

다음 예는 **ElementListSelector** 다이얼로그를 실행시키고, 사용자가 지정된 어떤 클래스(Class) 요소의 오퍼레이션(Operation) 컬렉션 중에서 하나를 선택하도록 하고 있다. "ShowInherited" 인자가 "true" 이기 때문에 지정된 클래스 요소의 부모 클래스들이 있다면 이 클래스 요소들의 오퍼레이션 컬렉션들도 선택할 수 있게 된다.

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

sel_dlg.AllowNull = false;
sel_dlg.ClearListElements();

var class = ... // Get reference to class element.
sel_dlg.AddListElementsByCollection(class, "Operations", true);

if (sel_dlg.Execute("Select an operation element.")){
    var selElem = sel_dlg.GetSelectedModel;
    ...
}
else{
    // If canceled, ...
}
```

위 예제에서는 **AddListElementsByCollection** 메소드를 사용하는 것을 보였다. 아래의 예제는 **AddListElementsByClass** 메소드를 사용하는 것을 보여준다. "IncludeChildInstances" 인자가 "true" 이기 때문에 지정된 타입의 요소들과 지정된 타입에서 파생되는 모든 서브 타입의 요소들이 목록에 추가된다.

```
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 ...
    }
}

```

## 메타-모델 다루기

이 장에서는 StarUML™ 메타-모델 요소들의 개념과 사용법에 대해 설명한다. StarUML™ 메타-모델 요소들은 **"Chapter 2. StarUML Architecture"**에서 소개했던 바와 같이 **Non\_Modeling Elements::MetaModeling Elements** 패키지에 속하는 요소들이다.

### 메타-모델 기본 개념

StarUML™ 메타-모델 요소들은 위에서 설명했던 StarUML™ 모델링 요소들에 대한 메타-수준의 접근 방법을 제공한다. 메타-모델은 한마디로 이 모델링 요소들을 정의하기 위한 요소들이라고 할 수 있다. 메타-모델 요소를 사용하면 현재 실행중인 프로젝트 내에서 각 모델링 요소들에 해당하는 요소들의 목록을 얻거나, 모델링 요소 그 자체의 정보에 접근할 수 있다. 메타-모델의 개념은 처음 접하는 사용자들에게 조금 어렵게 생각될 수도 있지만, 매우 명료하게 정의되어있으며 또 유용하게 사용될 수 있으므로 아래의 내용을 잘 읽어보기 바란다.

#### 메타-모델을 사용하는 간단한 예제

메타-모델의 개념을 설명하기 전에 간단한 예제를 통해 StarUML™ 메타-모델 요소들을 사용하는 방법에 대한 이해를 돕고자 한다. 먼저, 외부 API 를 통해 현재 실행중인 StarUML™ 애플리케이션의 모든 **Class** 요소들의 목록을 얻어야 한다고 가정해 보자. 이를 위해서 최상위 프로젝트 요소에서부터 하위 요소들을 모두 검색하는 방법을 사용할 수도 있지만, 메타-모델 요소를 사용하면 더욱 간단하게 표현될 수 있다. 아래의 코드를 보자.

```

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

var metaClass = meta.FindMetaClass("UMLClass");
for (var i = 0; i < metaClass.GetInstanceCount(); i++){
    var ACClassElem = metaClass.GetInstanceAt(i);
    ...
}

```

이 예제는 메타-모델 요소를 사용하여 모든 **Class** 요소들에 대한 참조를 얻고 있다. **Class** 요소들에 접근하기 위하여 **IMetaModel.FindMetaClass** 메소드의 인자로 **"UMLClass"**라고 모델링 요소의 이름을 주고 있는데, 만약 모든 **Attribute** 요소들의 목록을 얻어야 한다면 인자를 **"UMLAttribute"**로 바꾸면 된다. 즉 모든 모델링 요소에 동일하게 적용되는 것이다.

**노트:** 요소 이름에 대한 규칙은 **"Appendix B. UML 모델링 요소 목록"**을 참고하라.

두 번째 예제는 모델링 요소 자체의 정보에 접근하는 것을 보여 줄 것이다. 프로그램 구현 코드에서 UML 모델링 요소의 하나인 **Class** 요소가 어떤 속성들을 가지는지 알려면 어떻게 해야 할까? 이 질문은 사용자가 생성한 임의의 한 **Class** 요소에 어떤 **Attribute**들이 정의되어 있는가가 아니라, UML 모델링 요소인 **Class** 요소 그 자체에 어떤 **Attribute**들이 정의되어있는가 하는 것이다. 예를 들어 **Class** 모델링 요소는 **"Name"**, **"Visibility"**, **"IsAbstract"** 등의 속성(Attribute) 항목들을 가진다.

```

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;
    ...
}

```

이 예제는 **Class** 모델링 요소가 가지는 모든 속성(Attribute)들의 이름을 얻어 오고 있다. 첫 번째 예제와 마찬가지로 다른 모델링 요소에 대해 실행하려면 **IMetaModel.FindMetaClass** 메소드의 인자만 적절하게 바꾸면 된다.

### UML 메타모델링 아키텍처

이 섹션에서는 UML 메타모델링 아키텍처에 대해서 짧게 소개한다. 이것은 StarUML™ 메타-모델을 이해하는데 도움이 될 것이다.

OMG(Object Management Group)에서는 UML 요소들의 명세를 정의하기 위하여 메타모델링 아키텍처라는 방법을 사용하고 있다. 이 메타모델링 아키텍처는 다음과 같은 계층(Layer)들로 구성된다.

- Meta-metamodel
- Metamodel
- Model
- User Objects

UML 명세(Specification)에서 기술하고 있는 UML 모델링 요소들의 정의(Definition)는 이 계층들 중에서 **메타모델**에 해당한다. 즉 UML을 사용할 때 흔히 접하게 되는 **Package**, **Class**, **Use Case**, **Actor** 등등이 **메타모델** 요소들이나 것이다. 그리고 우리가 소프트웨어 모델링을 할 때 생성하게 되는 UML 요소들, 예를 들어 이름이 **"Class1"** 또는 **"Class2"**인 **Class** 요소들은 **메타모델**의 하나의 인스턴스(Instance), 즉 **모델**에 해당한다. 좀 더



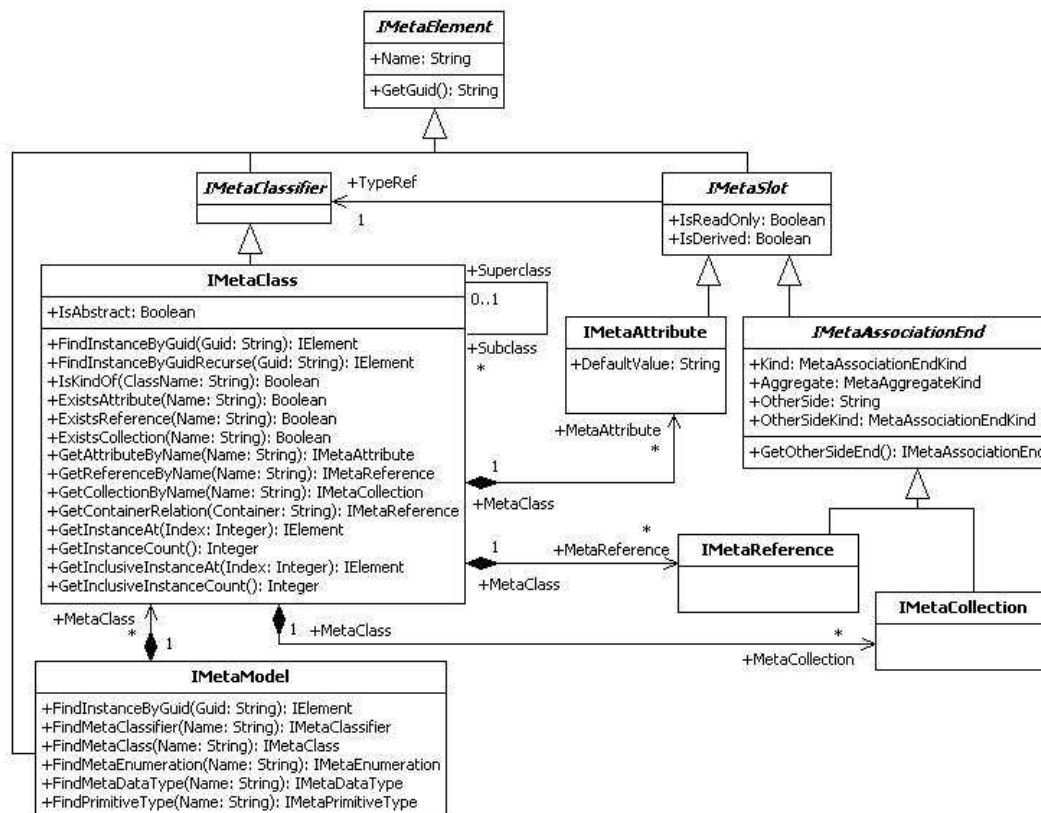
정확하게 말하면 이 "Class1", "Class2"는 **Class** (StarUML™에서는 **UMLClass**)라는 **메타모델** 요소의 인스턴스들이다.

그리고 Package, Class, Use Case, Actor 등과 같은 **UML 메타모델**을 정의하기 위한 기반 계층이 **메타-메타모델**인데, StarUML™ 메타-모델은 바로 이 **메타-메타모델** 계층에 해당하는 것이다. 즉, 모든 모델링 요소들이 아래에서 설명할 **MetaClass** 타입의 인스턴스에 해당한다고 볼 수 있는 것이다. 다만 StarUML™ 메타-모델은 모델링 요소들을 정의하기 위한 목적보다는 모델링 요소들을 메타-수준으로 일관성 있게 접근할 수 있도록 하는 역할을 하고 있다.

## 메타-모델 구조

### Meta-Model Organization

다음은 StarUML™ 메타-모델 요소들의 구성과 구조를 표현한 클래스 다이어그램이다. 화면 구성을 위하여 일부 요소들을 생략하였으므로, 완전한 다이어그램은 **Plastic Application Model**의 **::Application Model::Non\_Modeling Elements::Metamodeling Elements** 패키지를 참고하기 바란다.



StarUML™ 메타-모델은 이 그림과 같이 비교적 적은 수의 메타-모델 요소들로 구성되어 있다. 메타-모델 요소들의 최상위 요소는 **IMetaElement**로 **Name**과 **GUID** 속성을 가진다. 모델링 요소들은 메타-모델 요소(구체적으로 **IMetaClass**)의 인스턴스들이므로, **IMetaElement**의 **Name** 속성의 값은 이전 섹션들에서 설명했던 모델링 요소들의 이름 중의 하나가 될 것이다. 예를 들면 "Model", "View", "UMLClass", "UMLAttribute" 등이 있다.

최상위 **IMetaElement**에서 **IMetaClassifier**, **IMetaSlot** 등의 메타-모델 요소들이 나오는데, **IMetaClassifier**는 모델링 요소들 그 자체를 정의하기 위한 메타 요소이고 **IMetaSlot**은 모델링 요소들의 속성과 참조속성들을 정의하기 위한 것이다. 그리고 **IMetaClassifier**, **IMetaSlot**에서 **IMetaClass**, **IMetaAttribute**, **IMetaReference**, **IMetaCollection**, **IMetaModel** 등의 구체(Concrete) 요소들이 파생되는데, StarUML™ 메타-모델 구조에서 가장 중요한 역할을 하는 것이 이 요소들이다.

## 메타-모델 요소 다루기

### IMetaModel

**IMetaModel** 요소는 메타-모델 요소들을 컬렉션으로 유지하며 관리하는 요소로, 다른 메타-모델 요소들에 대한 접근성을 제공한다. **IMetaModel**은 한 StarUML™ 애플리케이션에서 하나만 존재하며, **IStarUMLApplication** 인터페이스를 통해 개체의 참조를 얻을 수 있다.

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

위에서 **IMetaModel** 요소가 다른 메타-모델 요소들에 대한 접근성을 제공한다고 했는데, 아래의 예제는 **IMetaModel**을 통해 **IMetaClass** 메타-요소들에 대한 참조를 얻는 것을 보여주고 있다. 참고로, 이 장의 **IMetaClass** 요소 항목에서 설명하겠지만 **IMetaClass** 타입의 참조는 모델링 요소들의 수만큼 존재한다 (아래의 예제를 통해 확인해보기 바란다).

```

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

        for (var i = 0; i < meta.GetMetaClassCount(); i++){
            var metaClass = meta.GetMetaClassAt(i);
            ...
        }

```

앞에서 보인 클래스 다이어그램에서는 생략되었지만, **IMetaClass**와 유사한 관계로 **IMetaEnumeration**, **IMetaDataType**, **IMetaPrimitiveType** 메타-모델 요소들이 있는데 **IMetaModel** 인터페이스를 통해 이 요소들에 대한 참조도 얻을 수 있다. 참고로 **IMetaEnumeration** 요소는 모델링 요소와 관련된 열거체(Enumeration) 타입을 정의하기 위한 메타-요소이다. **IMetaEnumeration** 요소의 인스턴스에는, 예를 들어 **UMLVisibilityKind**, **UMLAggregationKind** 등이 해당된다. **IMetaDataType**은 열거체(Enumeration), 기본타입(Primitive Type)이 아닌 데이터타입을 정의하기 위한 메타-요소이다. 이것의 인스턴스로는 **Points** 타입 하나가 존재한다. 그리고 **IMetaPrimitiveType** 요소는 기본타입(Primitive Type)들을 정의하기 위한 메타-요소로, 기본타입에는 **Integer**, **Real**, **Boolean**, **String** 네 가지가 있다.

**IMetaModel** 인터페이스는 메타-요소들에 대한 찾기 메소드를 제공한다. 아래의 예제는 이 장의 첫 번째 예제의 일부분인데 **IMetaModel.FindMetaClass** 메소드를 사용하여 **UMLClass** 모델링 요소에 대한 **IMetaClass** 요소의 참조를 얻고 있다. (**IMetaClass** 타입의 참조는 모델링 요소의 수만큼 존재한다)

```

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

```

**IMetaModel** 인터페이스는 **IMetaClass**와 마찬가지로 다른 메타-요소들에 대한 찾기 메소드들도 제공하는데, **FindMetaClassifier**, **FindMetaEnumeration**, **FindMetaDataType**, **FindPrimitiveType** 등이 있다.

**IMetaModel** 인터페이스는 모델링 요소의 GUID로 해당 모델링 요소에 대한 참조를 얻을 수 있는 **FindInstanceByGuid** 메소드를 제공한다. **FindInstanceByGuid** 메소드는 **IElement** 타입 참조를 반환하는데, 위 예제에 이어 아래와 같이 사용할 수 있다.

```

...
var guid = ... // 모델링 요소의 GUID 값을 얻음.
var elem = meta.FindInstanceByGuid(guid);
...

```

## IMetaClass

**IMetaClass** 요소는 각 모델링 요소들에 대한 정의를 제공하는 메타-요소이며, 또한 각 모델링 요소들의 인스턴스를 컬렉션으로 유지하고 관리한다. StarUML™애플리케이션에서 **IMetaClass**는 모델링 요소의 수 만큼 존재한다. 아래의 코드는 **IMetaModel.FindMetaClass** 메소드를 사용하여 모델링 요소 별로 **IMetaClass** 타입 참조를 얻는 것을 보여준다.

```

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");
...

```

**IMetaClass** 타입 요소의 참조를 얻는 다른 방법으로 모델링 요소들의 최상위 타입인 **IElement** 인터페이스의 **GetMetaClass** 메소드를 사용할 수 있다.

```

elem = ... // Get reference to modeling elements.
var metaClass = elem.GetMetaClass();

```

**IMetaClass** 인터페이스는 각 모델링 요소의 상속구조 상의 상위 요소(Superclass)와 하위요소(Subclass)들을 얻을 수 있는 메소드들을 제공한다. 참고로 최상위 모델링 요소인 **IElement** 타입 요소의 Superclass는 null이다.

```

var metaClass = ... // Get IMetaClass type reference.
var superClass = metaClass.Superclass;
...
for (var i = 0; i < metaClass.GetSubclassCount(); i++){
    var subCls = metaClass.GetSubclassAt(i);
    ...
}

```

**IMetaClass** 인터페이스는 **IMetaModel.FindInstanceByGuid**와 같이 모델링 요소의 GUID로 해당 모델링 요소에 대한 참조를 얻을 수 있는 **FindInstanceByGuid** 메소드를 제공한다. **IMetaClass**의 메소드는 특정 타입의 모델링 요소들에 대해서만 검색하기 때문에 **IMetaModel** 보다 효율적이다. 그리고 해당 타입에서 일치되는 값을 찾지 못한 경우, 파생된 모델링 요소들까지 검색하는 **FindInstanceByGuidRecurse**도 제공한다.

이 장의 첫 번째 예제에서는 **IMetaClass** 인터페이스의 **GetInstanceCount**, **GetInstanceAt** 메소드를 사용하여 특정 모델링 요소의 인스턴스들을 찾는 것을 보였다. 모델링 요소의 인스턴스란 사용자가 생성한 요소들을 말한다.

```
var metaClass = ... // Get IMetaClass type reference.
for (var i = 0; i < metaClass.GetInstanceCount(); i++){
    var AElem = metaClass.GetInstanceAt(i);
    ...
}
```

## IMetaAttribute

**IMetaAttribute** 인터페이스를 사용하여 각 모델링 요소의 기본속성(Attribute)들에 대한 명세를 읽을 수 있다. **IMetaAttribute**는 다음과 같이 **IMetaClass** 인터페이스를 통해 참조를 얻을 수 있다. 참고로 **IMetaClass** 인터페이스는 특정 이름의 기본속성(Attribute)이 존재하는지 검사하는 **ExistsAttribute** 메소드와, 특정 이름의 **IMetaAttribute** 타입 요소를 반환하는 **GetAttributeByName** 메소드도 제공한다.

```
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);
    ...
}
```

다음 예제는 모델링 요소의 기본 속성들의 명세를 읽는 것을 보여준다.

```
var metaAttr = ... // Get IMetaAttribute type reference.
var metaType = metaAttr.TypeRef;

var attrName = metaAttr.Name;
var attrType = metaType.Name;
...
```

그리고 **IMetaAttribute**의 상위 인터페이스인 **IMetaSlot**은 **IsReadOnly**, **IsDerived** 프로퍼티를 제공하는데, **IsReadOnly**는 해당 기본속성(Attribute)이 읽기전용인가를 나타내고 **IsDerived**는 해당 속성이 실제로 존재하는 것이 아니라 다른 속성들을 통해 유추되는 것인가를 나타내는 것이다.

## IMetaReference 와 IMetaCollection

**IMetaReference**, **IMetaCollection** 요소는 각 모델링 요소가 다른 모델링 요소를 참조하는 참조(Reference) 속성들을 정의한다. 이런 참조는 모두 연관관계(Association)를 나타내는 것이다. **IMetaReference**가 다중성(Multiplicity)이 '1' 이하인 참조를 나타내는데 반해 **IMetaCollection**은 컬렉션 형태로 구현되어야 하는 참조를 나타낸다. 이런 차이를 제외하면 **IMetaReference**과 **IMetaCollection**는 매우 유사하다 (**IMetaReference**, **IMetaCollection**은 모두 **IMetaAssociationEnd** 인터페이스에서 파생된다).

먼저 **IMetaReference**와 **IMetaCollection** 개체의 참조를 얻는 예제를 보자. 이것은 **IMetaAttribute**와 마찬가지로 **IMetaClass** 인터페이스를 사용한다.

```
var metaClass = ... // Get IMetaClass type reference for a specific modeling element.

// Get references to IMetaReference type objects.
for (var i = 0; i < metaClass.GetMetaReferenceCount(); i++){
    var metaAttr = metaClass.GetMetaReferenceAt(i);
    ...
}

// Get references to IMetaCollection type objects.
for (var i = 0; i < metaClass.GetMetaCollectionCount(); i++){
    var metaAttr = metaClass.GetMetaCollectionAt(i);
    ...
}
```

**IMetaReference**, **IMetaCollection**의 공통 상위 타입인 **IMetaAssociationEnd** 인터페이스는 해당 모델링 요소가 가지는 참조속성, 즉 연관관계(Association)의 명세를 정의하는 프로퍼티와 메소드들을 제공한다.

먼저 **Kind** 프로퍼티는 단순히 해당 연관관계가 단일 참조형인지 컬렉션 형태의 참조형인지를 구분한다. **IMetaReference** 타입의 경우에는 단일 참조형이고 **IMetaCollection** 타입은 컬렉션 형태의 참조형이다. 그리고 **Aggregate** 프로퍼티는 해당 연관관계의 **AggregationKind** 속성을 알려준다. 이 값은 열거체 타입으로 다음 값들 중의 하나이다.

- makNone (0): 없음
- makAggregate (1): 집합(Aggregation) 연관
- makComposite (2): 합성(Composition) 연관

**OtherSide** 프로퍼티는 연관관계의 반대쪽 연관-끝(**AssociationEnd**)의 이름을 알려주며, **OtherSideKind** 프로퍼티는 반대쪽 연관-끝(**AssociationEnd**)이 단일 참조형인지 컬렉션 형태의 참조형인지를 알려준다.

그리고 **GetOtherSideEnd** 메소드는 연관관계의 반대쪽 연관-끝(**AssociationEnd**)에 해당하는 **IMetaAssociationEnd** 타입 참조를 반환한다. 다음 예제는 **IMetaReference**, **IMetaCollection**의 공통 상위 타입인 **IMetaAssociationEnd** 인터페이스가 제공하는 프로퍼티와 메소드를 사용하는 것을 보여준다.

```
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();
...
```

**IMetaReference**, **IMetaCollection** 타입 개체가 어떤 **IMetaClass** 요소의 것인지 **IMetaSlot** 인터페이스의 **TypeRef** 참조속성을 통해 알 수 있다. 다음 예제는 임의의 모델링 요소에서 특정 연관관계(**Association**)의 반대편 요소가 무엇인지를 읽는 방법을 보여주고 있다.

```
var metaSlot = ... // Get IMetaReference or IMetaCollection type reference.
var otherSideEnd = metaSlot.GetOtherSideEnd();
var otherSideMetaClass = otherSideEnd.TypeRef;
...
```

[처음으로](#)

## 제 5 장. 접근법 사용하기

### 접근법의 기본 개념

소프트웨어를 개발하기 위한 방법론은 수도 없이 많으며, 각 회사나 조직마다 독자적인 방법론을 가지고 있거나 이미 존재하는 것을 자신의 개발조직이나 프로젝트에 맞게 조금씩 변형해서 사용하고 있다. 또한 개발할 소프트웨어에 대한 애플리케이션 영역(**Application Domain**)과 사용될 프로그래밍 언어, 플랫폼도 모두 다르다. 이러한 특성 때문에 소프트웨어를 모델링 할 때 초기에 설정해 주어야 할 사항이 많다. 접근법(**Approach**)은 소프트웨어 개발 방법론이나 플랫폼 등의 특성에 따라 프로젝트의 초기 환경을 설정해 주는 역할을 한다. 사용자들은 프로젝트를 생성할 때 적절한 접근법을 지정함으로써 자신이 의도하는 프로젝트의 품을 구성할 수 있게 된다.

접근법은 프로젝트를 생성할 때 다음과 같은 작업을 수행한다.

- 프로젝트에서 사용될 프로파일들을 설정한다. 접근법에서 정의된 프로파일들은 프로젝트 생성시에 자동으로 프로젝트에 포함된다.
- 프로젝트의 패키지 구조를 결정한다. 이런 패키지의 구조는 대체로 소프트웨어 개발 프로세스 모델에 의존적이다. 예를 들면 **4+1 View Model**의 접근법을 사용한다면 "Logical View", "Physical View", "Process View", "Development View" 와 "UseCase View" 등 다섯 가지의 패키지가 기본적으로 설정될 것이다.
- 프로젝트에서 참조할 프레임워크를 설정한다. 프로젝트가 특정 프로그래밍 언어나 플랫폼을 기반으로 한다면 거기에 해당하는 프레임워크를 접근법에 지정하여 프로젝트 생성시에 로딩하게 할 수 있다. 예를 들어 현재 프로젝트가 **Java**로 개발된다면 **JFC(Java Foundation Classes)** 프레임 워크를 접근법에서 지정하면 프로젝트 생성시에 패키지로 포함되어 직접 참조할 수 있다.
- 프로젝트에 기본적으로 포함할 모델 조각을 읽어 들인다.

새로운 접근법을 지원하려면 다음과 같은 절차가 필요하다.

- 새로운 접근법을 정의하기 위해 접근법 문서 파일(.apr)을 작성한다.
- 접근법 문서 파일을 모듈 디렉토리 하부 계층에 복사한다.

### 새로운 접근법 작성하기

#### 접근법 문서 파일의 기본 구조

접근법 문서 파일은 XML 문서의 규칙을 따르며, 확장자는 .apr(Approach File)이다. 접근법에 대한 내용은 **APPROACH** 요소 내에 기술되며, 구문이나 내용에 오류가 없도록 해야 한다.

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

- encoding** 속성 : XML 문서의 인코딩 속성 값을 지정한다 (e.g. UTF-8, EUC-KR). 이 속성의 값에 대해서는 XML 관련 자료를 참조하기 바란다.
- version** 속성 (APPROACH 요소) : 접근법 문서 포맷의 버전이다. (e.g 1.0)
- HEADER** 요소 : **Header Contents** 섹션 참조
- BODY** 요소 : **Body Contents** 섹션 참조

## Header Contents

접근법 문서의 HEADER 부분에는 프로파일의 이름과 상세설명 등 프로파일에 대한 개괄적인 정보를 기술한다.

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

- NAME 요소: 접근법의 명칭을 기록한다. 이것은 각각의 접근법을 구분하기 위해 사용되는 유일한 이름이다.
- DISPLAYNAME 요소: New Project 대화 상자에서 사용자에게 보여지는 접근법의 이름이다.
- DESCRIPTION 요소: 접근법에 대한 상세한 설명을 기술한다.

## Body Contents

접근법 문서의 BODY 부분은 크게 **IMPORTPROFILES** 요소와 **MODELSTRUCTURE** 요소로 나뉜다. **IMPORTPROFILES** 요소에는 프로젝트 생성시에 로딩할 프로파일 이름을 지정하고, **MODELSTRUCTURE**에서는 프로젝트의 초기 모델구조와 읽어 들일 프레임워크를 기술한다.

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

- IMPORTPROFILES 요소: 프로젝트에 포함시킬 프로파일들을 여러 개의 PROFILE요소로 나열한다.
- PROFILE 요소: 프로젝트에 포함시킬 프로파일의 이름을 기술한다.
- MODELSTRUCTURE 요소: **Model Structure** 섹션을 참조.

## Model Structure

MODELSTRUCTURE 요소에는 프로젝트의 초기 패키지 구조를 표현한다. 이를 위해서 모델(Model), 서브시스템(SubSystem), 패키지(Package)와 프레임워크(Framework)등을 계층적으로 구성한다. 예를 들어 SUBSYSTEM 요소 밑에 다시 모델, 서브시스템, 패키지, 프레임워크 등의 요소를 정의할 수 있다. 프레임워크는 그 자체가 하나의 패키지 요소에 해당되지만 하부에 다른 패키지 요소들을 포함할 수 없다.

다음은 MODELSTRUCTURE 요소에 대한 구문 구조를 정의한 것이다.

```
<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="..." />.
```

- name 속성 (MODEL, PACKAGE, SUBSYSTEM 요소): 각 UML 모델 요소의 이름이다.
- stereotypeProfile 속성 (MODEL, PACKAGE, SUBSYSTEM 요소): 각 UML 모델 요소에 적용할 스테레오타입이 정의되어 있는 프로파일 이름이다.
- stereotypeName 속성 (MODEL, PACKAGE, SUBSYSTEM 요소): 각 UML 모델 요소에 적용할 스테레오타입의 이름이다.
- name 속성 (IMPORTFRAMEWORK 요소): 모듈 디렉토리 밑에 존재하는 삼입할 프레임워크의 이름이다.
- fileName 속성 (IMPORTMODELFRAGMENT 요소): 상위 모델 요소에 읽어들일 모델 조각 파일(.mfg) 이름이다.

## 접근법 문서 예제

다음은 4+1 View Model을 위한 접근법을 예제로 보인 것이다.

```
<?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>

```

## 새로운 접근법 등록하기

작성된 접근법을 프로그램에서 인식하려면 접근법을 **StarUML™** 모듈 디렉토리(<install-dir>\modules) 하부 디렉토리로 옮겨야 한다. **StarUML™**은 프로그램 초기화 시에 모듈 디렉토리 하부를 검색하여 모든 접근법 파일들을 읽어들이고 후 프로그램에 자동 등록한다. 만약 접근법 파일이 문법에 맞지 않게 잘 못 작성되어 있거나, 확장자가 **.apr**이 아닌 경우에는 접근법을 정상적으로 읽어 들이지 않고 무시하게 될 것이다. 접근법이 모듈 디렉토리 하부에 존재하면 프로그램에 등록되는 데에는 문제가 없다. 그러나, 가급적이면 다른 모듈들과의 혼동을 피하기 위하여 별도의 서브디렉토리를 만들어 사용하는 것을 권고한다.

**노트:** 접근법의 아이콘을 등록하려면, 접근법과 동일한 이름의 아이콘 파일(.ico)을 만들어 접근법이 있는 디렉토리에 두면 된다. 접근법 아이콘은 새 프로젝트 대화상자에서 접근법의 이름과 함께 접근법 목록 리스트에 표시된다. 만약 접근법의 이름과 동일한 이름의 아이콘 파일이 없으면 기본 아이콘으로 접근법을 등록할 것이다.

**노트:** 접근법을 더 이상 사용하지 않으려면 **StarUML™** 모듈 디렉토리(<install-dir>\modules) 하부에서 삭제하면 된다.

## 접근법 관련 메소드 사용하기

### 시스템에 설치된 접근법 정보 읽기

접근법은 프로젝트 초기 설정을 위한 것이므로 프로그램에서 직접 액세스 할 일이 거의 없다. 따라서 **StarUML™**에서는 접근법의 조작을 위한 **COM** 자동화 개체를 지원하지 않는다. 다만 **IProjectManager**의 **GetAvailableApproachCount()**와 **GetAvailableApproachAt()**를 이용하면 시스템에 설치된 접근법의 개수와 이름들은 가져 올 수 있다. 다음은 접근법 관련 메소드의 호출 형식이다.

```

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

```

### 접근법을 사용하여 프로젝트 생성하기

**IProjectManager.NewProjectByApproach()**를 호출하면 주어진 접근법으로 새 프로젝트를 생성할 수 있다. 파라미터로 입력되는 ApproachName은 시스템에 설치된 접근법 이름 중의 하나가 되어야 한다. 그렇지 않으면 빈 프로젝트를 생성하게 될 것이다. **IProjectManager**의 **NewProjectByApproach()**의 형식은 다음과 같다.

```

IProjectManager.NewProjectByApproach(ApproachName: String)

```

다음은 "UMLComponents" 접근법으로 새로운 프로젝트를 생성하는 JScript 예제이다.

```

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

```

[처음으로](#)

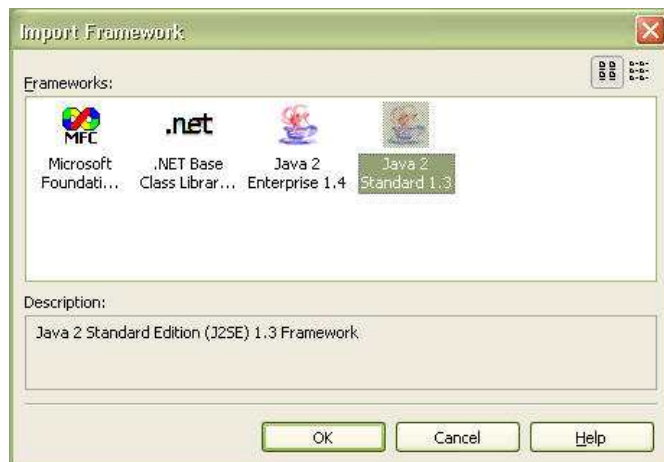
## 제 6 장. 프레임워크 사용하기

### 모델 프레임워크의 기본 개념

모델 프레임워크(Model Framework)는 애플리케이션 프레임워크(Application Framework)나 클래스 라이브러리(Class Library) 등을 **StarUML™**에서 사용할 수 있도록 구성한 것을 말한다. 예를 들어 JFC(Java Foundation Classes), MFC(Microsoft Foundation Classes), 델파이 VCL(Visual Component Library) 등이 모델 프레임워크의 대상이 될 수 있다. 그리고 이 장에서 설명하겠지만 사용자가 필요한 모델 프레임워크를 정의하여 사용할 수 있다. 모델 프레임워크를 사용하는 가장 큰 이점은 공통적이고 기반이 되는 모델링 요소 및 구조의 공유와 재사용성 이다.



StarUML™ 애플리케이션의 **[File]-[Import]-[Framework]** 메뉴를 실행했을 때 보여지는 "Import Framework 다이얼로그"(아래 그림)는 시스템에 설치된 모델 프레임워크들의 목록을 보여주는데, 목록에서 항목을 선택하고 실행하면 해당 모델 프레임워크에서 미리 정의하고 있는 모델링 구조가 지정한 경로에 자동으로 포함되게 된다. 모델 프레임워크는 여러 개의 유닛(Unit) 파일들로 구성되며, StarUML™에 포함된 모델 프레임워크는 실제로 유닛과 동일하게 취급된다.



StarUML™ 외부 API를 사용하여 위와 같이 시스템에 설치된 모델 프레임워크들의 목록을 얻거나 특정 모델 프레임워크를 프로젝트에 포함시킬 수 있다. 이에 대해서는 위에서 자세히 설명할 것이다.

## 새로운 모델 프레임워크 작성하기

모델 프레임워크는 여러 개의 유닛 파일(.unt)들과 하나의 모델 프레임워크 정의 문서 파일(.frw)로 구성되며, 선택 요소로 아이콘 파일(.ico)이 있을 수 있다. 새로운 모델 프레임워크를 정의하려면 다음과 같은 절차를 따른다.

1. 모델 프레임워크의 모델 정보를 담고 있는 유닛 파일들을 생성한다. ("Chapter 4. Using Open API"를 참고)
2. 모델 프레임워크를 정의하는 모델 프레임워크 문서 파일(.frw)을 작성한다.
3. 작성한 유닛파일과 모델 프레임워크 문서 파일 그리고 아이콘 파일을 StarUML™ 모듈 디렉토리 하부에 복사한다.

## 모델 프레임워크 문서 파일의 기본 구조

모델 프레임워크 문서 파일은 XML 포맷으로 정의되어 있으며 확장자는 .frw(Framework File)이다. 모델 프레임워크에 대한 내용은 **FRAMEWORK** 태그 내에 기술되며 구문이나 내용에 오류가 없도록 해야 한다.

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

- encoding 속성 : XML 문서의 인코딩 속성 값을 지정한다 (e.g. UTF-8, EUC-KR). 이 속성의 값에 대해서는 XML 관련 자료를 참조하기 바란다.
- version 속성 (FRAMEWORK 요소) : 모델 프레임워크 문서 포맷의 버전이다. (e.g 1.0)
- HEADER 요소 : **Header Contents** 섹션 참조
- BODY 요소 : **Body Contents** 섹션 참조

## Header Contents

Header 부분에는 모델 프레임워크의 이름과 설명 등 모델 프레임워크에 대한 개괄적인 정보를 기술한다.

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

- Name 요소: 모델 프레임워크의 명칭을 기록한다. 이것은 다른 모델 프레임워크들과 구분하는 ID 역할을 수행한다.
- DISPLAYNAME 요소: "모델 프레임워크 가져오기 다이얼로그" 등에서 사용자에게 보여지는 이름을 기록한다.
- DESCRIPTION 요소 : 모델 프레임워크에 대한 설명을 기술한다.

## Body Contents



Body 부분에서는 모델 프레임워크에 대한 실제 정보가 기술되는데, Body 부분은 크게 **IMPORTPROFILES**, **FRAMEWORKMODELS** 두 부분으로 구성된다.

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

- **IMPORTPROFILES** 요소: 해당 모델 프레임워크가 포함될 때 기본적으로 로드(Load)되어야 하는 UML 프로파일들을 나열한다.
- **PROFILE** 요소: 로드 되어야 하는 UML 프로파일 각각의 이름을 하나씩 지정한다.
- **FRAMEWORKMODELS** 요소: 해당 모델 프레임워크를 구성하고 있는 유닛 파일들을 나열한다.
- **UNIT** 요소: 유닛 파일의 이름을 하나씩 지정한다. 유닛 파일의 이름만을 기록하며 경로명은 포함하지 않는다. 해당 모델 프레임워크를 구성하는 유닛 파일들은 반드시 모델 프레임워크 문서 파일과 동일한 경로에 있어야 한다.

**노트:** "UNIT" 요소"에는 최상위 유닛에 해당하는 유닛 파일들만 지정한다. "Chapter 4. 프로젝트 다루기"에서 보았듯이 어떤 유닛이 하위 유닛들을 가지는 경우, 해당 상위 유닛이 로드 될 때 하위 유닛들은 자동으로 같이 로드 된다.

## 모델 프레임워크 문서 예제

다음은 Java 2 Standard Edition (J2SE) 1.3 모델 프레임워크를 정의하고 있는 모델 프레임워크 문서를 예로 보인 것이다.

```
<?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>
```

## 새로운 모델 프레임워크 등록하기

작성된 프레임워크를 프로그램에서 인식하려면 프레임워크와 관련 파일들을 StarUML™ 모듈 디렉토리(<install-dir>\modules) 하부 디렉토리로 옮겨야 한다. StarUML™은 프로그램 초기화 시에 모듈 디렉토리 하부를 검색하여 모든 프레임워크 파일들을 읽어들이고 후 프로그램에 자동 등록한다. 만약 프레임워크 파일이 문법에 맞지 않게 잘 못 작성되어 있거나, 확장자가 .frw가 아닌 경우에는 프레임워크를 정상적으로 읽어 들이지 않고 무시하게 될 것이다.

프레임워크가 모듈 디렉토리 하부에 존재하면 프로그램에 등록되는 데에는 문제가 없다. 그러나, 가급적이면 다른 모듈들과의 혼동을 피하기 위하여 별도의 서브디렉토리를 만들어 사용하는 것을 권고한다.

**노트:** 프레임워크의 아이콘을 등록하려면, 프레임워크와 동일한 이름의 아이콘 파일(.ico)을 만들어 프레임워크가 있는 디렉토리에 두면 된다. 프레임워크 아이콘은 **[Import Framework]** 대화상자에서 프레임워크의 이름과 함께 프레임워크 목록 리스트에 표시된다. 만약 프레임워크의 이름과 동일한 이름의 아이콘 파일이 없으면 기본 아이콘으로 프레임워크를 등록할 것이다.

**노트:** 프레임워크를 더 이상 사용하지 않으려면 StarUML™ 모듈 디렉토리(<install-dir>\modules) 하부에서 삭제하면 된다.

## 모델 프레임워크 관련 메소드 사용하기

### 시스템에 설치된 모델 프레임워크 정보 읽기

외부 API를 통해 시스템에 설치된 모델 프레임워크들의 목록을 확인할 수 있다. 이와 관련된 외부 API는 **IProjectManager** 인터페이스의 **GetAvailableFrameworkCount** 메소드와 **GetAvailableFrameworkAt** 메소드이다. 아래의 코드는 이 두 메소드의 형식을 보여준다.

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

### 모델 프레임워크 불러오기

**IProjectManager.ImportFramework** 메소드를 사용하여 시스템에 등록되어있는 특정 모델 프레임워크를 현재 프로젝트에 포함시킬 수 있다. 이 메소드의 형식은 아래와 같은데, **OwnerPackage** 인자는 해당 모델 프레임워크가 포함될 상위 모델 요소를 지정하며 **IUMLPackage** 타입 모델 요소여야 한다. 그리고 **FrameworkName** 인자는 불러올 모델 프레임워크의 이름으로, 모델 프레임워크의 정확한 이름(ID)을 문자열 값

으로 지정하면 된다.

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

다음 예제는 `IProjectManager.ImportFramework` 메소드를 사용하여 "J2SE1.3" 모델 프레임워크를 불러오는 것을 보여주고 있다.

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

var owner = ... // Get reference to IUMLPackage type element.
prjMgr.ImportFramework(owner, "J2SE1.3");
```

[처음으로](#)

## 제 7 장. UML 프로파일 사용하기

### UML 프로파일의 기본 개념

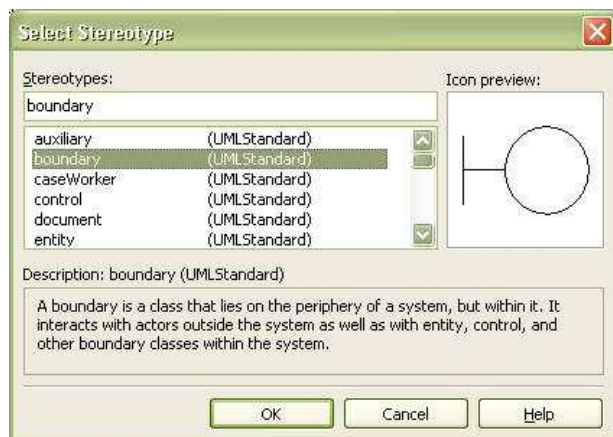
#### UML 확장 메커니즘

UML은 범용적인 소프트웨어 모델링 언어이며, 일반적인 소프트웨어 모델링 요구사항들을 충족할 수 있도록 잘 정의된 풍부한 모델링 개념들과 표기법들을 제공하고 있다. 그렇지만 오늘날 소프트웨어 모델링/개발 환경이 매우 다양하기 때문에 UML 표준에는 없는 추가적인 요소 또는 의미들에 대한 요구가 생길 수 있다. UML은 이러한 요구들을 지원할 수 있는 개념까지 표준 안에서 제공하고 있는데, 이것이 바로 UML 확장 메커니즘(Extension Mechanism)이다.

UML의 확장 메커니즘은 스테레오타입(**Stereotypes**), 제약사항(**Constraints**), 태그정의(**Tag Definitions**), 태그값(**Tag Values**)을 사용하여, UML 모델링 요소의 의미(**Semantics**)를 확장하거나 새로운 의미를 가지는 UML 모델링 요소를 정의할 수 있도록 하는 것이다.

#### 스테레오타입

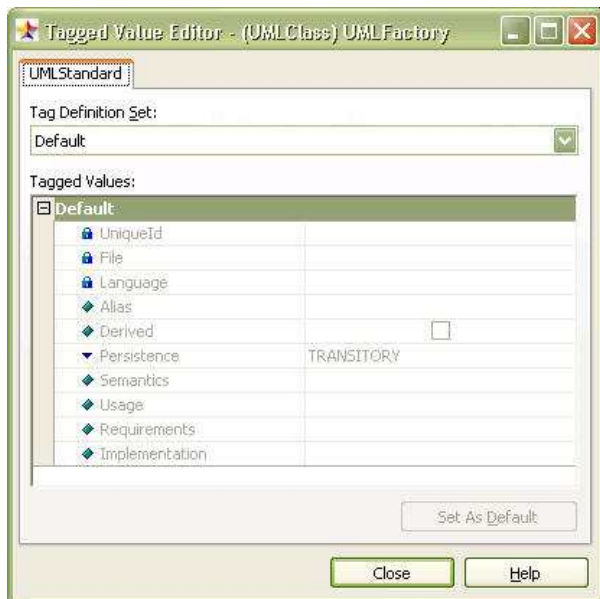
스테레오타입(**Stereotype**)은 표준 UML 모델링 요소에 새로운 속성과 제약사항을 추가할 수 있도록 정의하는 모델링 요소이다. 스테레오타입은 모델링 요소가 새로운 표기법(**Notation**)을 제공하도록 정의할 수도 있다. 아래의 그림은 StarUML™ 애플리케이션에서 스테레오타입 선택 버튼을 눌렀을 때 나타나는 스테레오타입 선택 다이얼로그를 보인 것이다. 스테레오타입 선택 다이얼로그는 현재 프로젝트에 포함되어 있는 UML 프로파일에 정의되어 있는 선택 가능한 스테레오타입 목록을 보여준다. 외부 API를 통해서도 스테레오타입을 설정하거나 변경할 수 있는데 이에 대해서는 뒤에서 설명한다.



**노트:** UML 표준에서는 하나의 확장 가능한 모델링 요소가 여러 개의 스테레오타입을 가질 수 있도록 하고 있지만, StarUML™에서는 각 모델링 요소는 스테레오타입을 하나만 가지도록 제한하고 있다.

#### 태그정의

태그정의(**Tag Definition**)는 어떤 모델링 요소에 추가될 수 있는 새로운 속성(**Property**) 항목을 정의하는 요소이다. 그리고 태그정의가 적용된 각 각의 요소에서 태그정의에 의해 추가된 속성에 대한 값을 지정한 것은 태그값(**Tagged Value**)에 해당한다. 태그값은 기본적인 데이터타입 값이거나 다른 모델링 요소에 대한 참조, 또는 컬렉션 등이 될 수 있다. 다음 그림은 StarUML™ 애플리케이션에서 확장 속성 편집기를 띄운 화면이다. 확장 속성 편집기는 현재 프로젝트에 포함되어 있는 UML 프로파일에서 정의하는 태그정의들 중에서 선택된 모델링 요소에 해당하는 항목들의 목록을 보여준다. 외부 API를 통해서도 모델링 요소의 태그값을 설정하거나 변경할 수 있는데 이에 대해서는 뒤에서 설명한다.



## 제약사항

제약사항(Constraint)은 어떤 모델링 요소에 특정 제약 사항을 추가함으로써 해당 모델링 요소의 의미를 재정의할 수 있도록 한다. 제약사항에 대한 설명은 "**Chapter 5. 모델링 요소 다루기**"의 "**ExtCore 요소들**" 섹션을 참고하도록 한다.

**노트:** StarUML™ UML 프로파일에서는 제약사항에 대한 정의는 제외 시키고 있다.

## UML 프로파일

UML 프로파일(Profile)은 UML 확장 메커니즘을 구현한 하나의 패키지라고 할 수 있다. 예를 들어 특정 소프트웨어 도메인 이라던지 개발 플랫폼 등에 필요한 스테레오타입, 제약사항, 태그정의, 데이터타입들을 하나로 묶은 것이다.

UML 프로파일은 스테레오타입, 제약사항, 태그정의, 데이터타입 요소로 구성된다. UML 표준에서는 프로파일(Profile)을 "<<profile>>" 스테레오타입이 붙은 패키지 요소로 정의하도록 하고 있는데, StarUML™에서는 XML 형식의 파일로 정의하여 사용할 수 있도록 하고 있다.

## StarUML™ 프로파일이 제공하는 추가적인 확장 메커니즘들

StarUML™의 프로파일에서는 UML에서 정의하고 있는 확장 메커니즘 이외에 몇 가지의 확장 메커니즘 요소들을 추가로 제공하고 있다. 추가된 확장 요소들은 **다이아그램타입(Diagram Type)**, **요소 프로토타입(Element Prototype)**, **모델 프로토타입(Model Prototype)**, **팔레트 확장(Palette)** 이다. 이들 확장 요소들은 기존의 요소의 의미를 확장하거나 요소 생성을 위한 정형적인 틀을 제공하고 이를 사용자 인터페이스에 반영한다.

## 다이아그램타입

다이아그램타입(Diagram Type)은 표준 UML 다이어그램들에 추가적인 의미를 부여하여 새로운 다이어그램을 정의할 수 있도록 하는 확장 요소이다. 이것은 데이터모델 다이어그램, Robustness 분석 다이어그램 등 각 설계 단계별로 특화된 용도의 다이어그램을 정의하거나 여러 도메인에서 사용되는 다양한 종류의 다이어그램을 StarUML™에 수용하는데에 유용하다. 다이어그램타입은 다이어그램 요소들의 "**DiagramType**" 속성으로 정의되고, 프로젝트에 프로파일을 포함시켰을 때, **[Add Diagram]** 메뉴에 추가되어 사용자가 해당 다이어그램을 생성할 수 있도록 해준다. 다이어그램타입 속성은 스테레오타입과 다르게 모델링 시에 변경할 수 없다.

## 요소 프로토타입

요소 프로토타입(Element Prototype)은 표준 UML 요소에 속성들을 미리 설정하여 요소 생성을 위한 하나의 견본을 정의하는 것이다. 요소 프로토타입은 팔레트에 연결하여 견본의 복사본에 해당하는 요소를 직접 생성하거나 외부 API를 통해 생성할 수 있도록 해 준다.

## 모델 프로토타입

모델 프로토타입(Model Prototype)은 요소 프로토타입과 유사하나 모델에만 적용할 수 있는 확장 요소이다. 요소 프로토타입이 팔레트와 연결되는 것과는 달리 모델 프로토타입으로 정의되면 **[Add Model]** 메뉴의 서브 메뉴로 삽입된다. 이 메뉴를 통해 견본의 복사본에 해당하는 모델 요소를 직접 생성할 수 있다.

## 팔레트 확장

팔레트 확장(Palette)는 메인 폼 좌측에 나오는 팔레트를 추가할 수 있도록 해 준다. 추가된 팔레트에는 프로파일 안에서 정의하고 있는 요소 프로토타입이나 표준 UML 요소들을 팔레트 항목으로 지정할 수 있다.

## 프로파일의 포함(Include)과 제외(Exclude)

StarUML™ 애플리케이션의 현재 프로젝트에 필요한 UML 프로파일이 있으면 해당 프로파일을 프로젝트로 포함시켜야 한다. "**UML Standard**

**Profile**"을 제외한 나머지 프로파일들을 자동으로 추가되지 않기 때문이다. StarUML™ 애플리케이션에서 프로파일 추가는 **[Model]->[Profiles]** 메뉴를 실행했을 때 보여주는 프로파일 다이얼로그(아래)를 통해 할 수 있다. 왼쪽의 **"Available profiles"** 목록에는 현재 사용자 시스템에 등록되어 있는 프로파일들의 목록이 보여지고, 오른쪽 **"Included profiles"** 목록에는 현재 프로젝트에 포함되어 있는 프로파일들의 목록이 보여진다. 프로파일을 추가는 **"Available profiles"** 목록에서 추가할 프로파일 항목을 선택한 후, 종간의 **"Include"** 버튼을 누르는 것으로 간단하게 끝난다. 프로파일을 추가하면 위에서 보았던 스테레وتا입 선택 다이얼로그나 확장 속성 편집기 등에 포함된 프로파일에 정의된 스테레وتا입들과 태그정규식들이 추가될 것이다. 현재 프로젝트에 포함되어 있는 프로파일들 중에서 필요 없는 항목이 있다면 **"Exclude"** 버튼을 눌러 간단하게 제외시킬 수 있다. 프로파일을 제외시키면 프로젝트 내에서 해당 프로파일을 참고하고 있던 정보들이 모두 삭제된다는 점에 주의해야 한다. 외부 API를 통해서도 프로파일을 추가하거나 삭제할 수 있는데 이에 대해서는 위에서 설명한다.



## UML 프로파일 작성하기

### 프로파일 문서 파일의 기본 구조

프로파일 문서 파일은 XML 포맷으로 정의되어 있으며 확장자는 .prf이다. 프로파일에 대한 내용은 **PROFILE** 태그 내에 기술되며 구문이나 내용에 오류가 없도록 해야 한다.

프로파일 문서의 기본 구조는 다음과 같다.

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

- **encoding** 속성 : XML 문서의 인코딩 속성 값을 지정한다 (e.g. UTF-8, EUC-KR). 이 속성의 값에 대해서는 XML 관련 자료를 참조하기 바란다.
- **version** 속성 (PROFILE 요소) : PRF 문서 포맷의 버전이다. (e.g 1.0)
- **HEADER** 요소 : **Header Contents** 섹션 참조
- **BODY** 요소 : **Body Contents** 섹션 참조

### Header Contents

프로파일 문서의 **HEADER** 부분에는 프로파일의 이름과 상세설명 등 프로파일에 대한 개괄적인 정보를 기술한다.

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

- **NAME** 요소 : 프로파일의 명칭을 기록한다. 이것은 실질적인 프로파일 ID의 역할을 수행한다.
- **DISPLAYNAME** 요소 : 프로파일 다이얼로그 등 사용자 인터페이스에서 캡션으로 사용되는 명칭이다.
- **DESCRIPTION** 요소 : 프로파일에 대한 상세한 설명을 기술한다.
- **AUTOINCLUDE** 요소 : 프로젝트를 새로 만들 때 프로파일을 자동으로 포함할 것인지를 명시한다.

### Body Contents

프로파일 문서의 **BODY** 부분에는 프로파일의 구체적인 내용을 포함하게 된다. 여기에는 크게 스테레وتا입(Stereotype)들과 데이터타입

(DataType)들, 태그정의집합(TagDefinitionSet)들과 추가적인 확장 요소들이 기술된다.

```
<BODY>
  <STEREOTYPELIST>

  <STEREOTYPELIST>
  <TAGDEFINITIONSETLIST>

  <TAGDEFINITIONSETLIST>
  <DATATYPELIST>

  <DATATYPELIST>
  <ELEMENTPROTOTYPELIST>

  <ELEMENTPROTOTYPELIST>
  <MODELPROTOTYPELIST>

  <MODELPROTOTYPELIST>
  <PALETTELIST>

  <PALETTELIST>
  <DIAGRAMTYPELIST>

  <DIAGRAMTYPELIST>
</BODY>
```

- STEREOTYPELIST 요소 : 여러 개의 스테레오타입(STEREO TYPE 요소)을 정의한다. 스테레오 타입에 대한 정의는 **Stereotype** 섹션 참조.
- TAGDEFINITIONSETLIST 요소 : 여러 개의 태그정의집합(TAGDEFINITIONSET 요소)들을 정의한다. 태그정의집합에 대한 정의는 **TagDefinitionSet** 섹션 참조.
- DATATYPELIST 요소 : 여러 개의 데이터 타입(DATATYPE 요소)들을 정의한다. 데이터 타입에 대한 정의는 **DataType** 섹션 참조.
- ELEMENTPROTOTYPELIST 요소 : 여러 개의 요소 프로토타입(ELEMENTPROTOTYPE)들을 정의한다. 요소 프로토타입에 대한 정의는 **ElementPrototype** 섹션 참조.
- MODELPROTOTYPELIST 요소 : 여러 개의 모델 프로토타입(MODELPROTOTYPE)들을 정의한다. 모델 프로토타입에 대한 정의는 **ModelPrototype** 섹션 참조.
- PALETTELIST 요소 : 여러 개의 팔레트 확장(PALETTE 요소)들을 정의한다. 팔레트 확장에 대한 정의는 **Palette** 섹션 참조.
- DIAGRAMTYPELIST 요소 : 여러 개의 다이어그램 타입(DIAGRAMTYPE 요소)들을 정의한다. 다이어그램 타입에 대한 정의는 **DiagramType** 섹션 참조.

## Stereotype

STEREOTYPE 요소에는 스테레오타입에 대한 정보와 상속구조 등을 정의한다.

```
<STEREOTYPE>
  <NAME>...</NAME>
  <DESCRIPTION>...</DESCRIPTION>
  <BASECLASSES>
    <BASECLASS>...</BASECLASS>

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

- NAME 요소 : 스테레오타입의 명칭을 기록한다. 이 명칭은 프로파일 파일 내에서 유일해야 한다.
- DESCRIPTION 요소 : 스테레오타입에 대한 설명을 기술한다.
- BASECLASSES 요소 : 여기에는 스테레오타입이 적용될 수 있는 UML 모델링 요소의 명칭을 여러 개 나열할 수 있다. 이 부분에 쓰여지는 요소의 명칭은 UML 요소의 이름을 사용한다 (e.g., UMLClass, UMLClassifier, UMLAttribute, UMLPackage, ...).  
**노트:** 만약, UMLClassifier와 같은 추상클래스의 이름을 적으면 그것으로부터 상속 받은 모든 요소에 적용된다. 그리고 상위 스테레오타입(PARENT 요소)이 지정된 경우에는 이 부분은 정의하지 않으며, 만약 정의되더라도 무시되고 상위 스테레오타입의 BASECLASSES 값이 적용된다.
- PARENT 요소 : 스테레오타입들 간에는 상속관계를 가질 수 있다. PARENT 요소에는 상위 스테레오타입의 이름을 기록한다. 상속관계에 있는 스테레오타입은 반드시 동일한 프로파일 내에 정의된 것이어야 하며, 상위 스테레오타입이 없는 경우에는 지정하지 않거나 생략할 수 있다.
- RELATEDTAGDEFINITIONSET 요소 : 스테레오타입과 연관된 태그정의집합(TagDefinitionSet)의 이름을 기록한다. 이것은 스테레오타입이 지정된 요소에 제공되는 추가적인 속성들로 이해할 수 있으며 없는 경우 생략할 수 있다. 여기에서 지정된 태그정의집합은 반드시 동일한 프로파일 내에 정의되어 있어야 한다.
- ICON 요소 : 스테레오타입은 사용자의 선택에 따라 아이콘형태로 표시할 수 있다. 여기에는 스테레오타입의 아이콘의 파일 이름을 기록한다. 스테레오 타입의 아이콘 파일은 .WMF, .EMF 와 .BMP 세 가지가 지원된다. 아이콘 파일은 반드시 Profile Document가 있는 디렉토리 내에 위치해 있어야 하며, Profile Document에는 경로명을 제외한 파일명만 기술하도록 한다.
- minWidth 속성 (ICON 요소) : 스테레오타입 아이콘의 최소 너비를 지정한다.
- minHeight 속성 (ICON 요소) : 스테레오타입 아이콘의 최소 높이를 지정한다.
- NOTATION 요소 : 스테레오타입은 아이콘 형태로도 표시할 수 있지만 노테이션 기술 언어를 작성하여 드로잉하는 방법을 재정의할 수도

있다. 이 요소에는 노테이션을 기술하기 위한 노테이션 확장 파일(.nxt)의 경로를 지정한다. 노테이션 확장이 사용된 요소는 UML 표준 노테이션으로 그려지지 않고, 노테이션 확장 파일에 기술된 대로 도식하게 된다. 노테이션 확장 파일은 반드시 Profile Document가 있는 디렉토리에 위치해 있어야 하며, 경로명을 제외한 파일명만 기술하도록 한다.

## TagDefinitionSet

TAGDEFINITIONSET 요소에는 태그요소집합의 기본 정보를 기술하고, TAGDEFINITIONLIST 요소 밑에 여러 개의 TAGDEFINITION 요소를 두어 태그정의집합에 포함된 태그정의들을 나열한다.

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

- **NAME** 요소 : 태그정의집합의 이름을 기록한다. 특정 스테레오타입에 연관된 태그정의들인 경우에는 이름을 스테레오타입과 동일하게 사용할 것을 권장한다. (이 경우 사용자 인터페이스에서 스테레오타입과 동일한 이름의 태그정의집합이 존재하면 그것을 제일 먼저 표시해 줄 수 있다.)
- **BASECLASSES** 요소 : 태그정의집합이 적용될 UML 요소의 이름들을 열거한다. (Stereotype의 BASECLASSESES 요소와 동일한 원칙으로 적용) 만약 어떤 스테레오타입에 연관된 태그정의집합으로 지정된 경우에는 이 부분을 정의하지 않으며, 정의하더라도 무시되고 해당 스테레오타입의 BASECLASSES와 동일한 것으로 인식한다.
- **TAGDEFINITIONLIST** 요소 : 집합에 포함되는 태그정의(TagDefinition)를 여러 개 열거한다. **TagDefinition** 섹션 참조.

## TagDefinition

TAGDEFINITION 요소에는 태그정의집합에 포함되어 있는 태그정의(TagDefinition) 각각에 대해서 상세히 정의한다.

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

- **lock** 속성 (TAGDEFINITION 요소) : 태그값을 UI에서 편집 가능하게 할 것인지를 설정한다. 이것을 “True”로 두면 확장속성 편집기에 의해서는 편집이 불가능하고 외부 COM 인터페이스를 통해서만 값을 변경할 수 있다. 이 속성을 생략가능하며 기본 값은 “False”이다.
- **NAME** 요소 : 태그의 이름이다. 이것은 TagDefinitionSet 내에서 유일해야 한다.
- **TAGTYPE** 요소 : 태그의 타입이다. 여기에는 Integer, Boolean, Real, String, Enumeration, Reference, Collection의 5가지 타입이 올 수 있다.
- **referenceType** 속성 (TAGTYPE 요소) : 태그의 타입이 Reference 혹은 Collection인 경우, 여기에 지정될 수 있는 객체의 참조가 어떠한 타입으로 제한되는가를 표현한다. 예를 들어, “UMLClass”라고 지정하면 Class 타입 만이 연결될 수 있다. 만약 이 부분을 생략한 경우에는 “UMLModelElement”로 간주한다. 태그의 타입이 Integer, Boolean, Real, String, Enumeration일 경우 무시된다.
- **DEFAULTVALUE** 요소 : 태그의 기본 값을 기록한다. Reference Type이나 Collection Type일 경우에는 무시되며, null이 기본 값으로 설정된다.
- **LITERALS** 요소 : 만약 태그 타입이 Enumeration인 경우에 여기에 열거할 수 있는 리터럴을 정의한다. 다른 타입일 경우 무시된다.

## Data Type

DataType요소에는 하나의 데이터타입을 정의한다. 이 요소는 NAME을 서브요소로 갖는다.

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

- **NAME** 요소 : 데이터 타입의 명칭을 기록한다.

## ElementPrototype

ELEMENTPROTOTYPE 요소에는 생성할 요소의 패턴을 정의하는 요소 프로토타입의 정보를 기술한다.

```
<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>

```

- **NAME** 요소 : 요소 프로토타입의 명칭이다. 이 명칭은 프로파일 내에서 유일해야 한다.
- **DISPLAYNAME** 요소 : 요소 프로토타입이 팔레트 등 사용자 인터페이스 상에서 표시될 때의 이름이다.
- **ICON** 요소 : 요소 프로토타입이 팔레트 등 사용자 인터페이스 상에서 표시될 때의 아이콘 파일의 이름이다. 요소 프로토타입의 아이콘 파일은 16 x 16 크기의 .BMP 포맷 형식의 이미지 파일이어야 한다. 아이콘 파일은 반드시 **Profile Document**가 있는 디렉토리에 위치해야 하며 경로명을 제외한 파일명만 기술하도록 한다.
- **DRAGTYPE** 요소 : 요소 프로토타입에 해당하는 요소를 생성하기 위해 사용자가 다이어그램 위에 마우스로 드래그 하여 위치와 크기를 지정할 때 어떤 방식으로 보여 줄 것인가를 지정한다. **Rect** (사각형), **Line** (선)의 두 가지의 값 중 하나이어야 한다.
- **BASEELEMENT** 요소 : 요소 프로토타입의 복사본을 생성하기 위한 원래 **UML** 표준 요소의 명칭을 지정한다. 이 요소는 생략할 수 없으며, 값을 지정하지 않으면 요소 프로토타입을 인식하지 못한다. 사용할 수 있는 **UML** 표준 요소의 명칭은 다음과 같다.

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

- **argument** 속성 : Base 요소가 Association, Pseudostate 등 일부 요소의 경우 생성을 위한 인자가 필요할 수 있다. 이들 요소들은 인자에 따라 생성되는 요소의 특정 속성값들이 미리 설정된다. 기본값은 0이며, 대부분의 경우 지정할 필요가 없다. **StarUML™**에서 사용하는 요소의 인자값은 다음과 같다.

요소 명칭	의미와 값
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 with Call Action = 0, Message with Send Action = 1,



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

- **STEREOTYPENAME** 요소 : 요소 프로토타입의 **Stereotype** 이름을 지정한다. 여기에 값을 설정해 두면 요소 생성 시 **Stereotype** 속성 값으로 입력된다. 이 값은 생략 가능하다.
- **STEREOTYPEDISPLAY** 요소 : 요소 프로토타입에 따라 요소를 생성 할 때 뷰의 스테레오타입을 어떻게 표시할 것인가를 지정한다. **sdkText**(텍스트로 표시), **sdkIcon**(아이콘으로 표시), **sdkNone**(표시하지 않음), **sdkDecoration**(데코레이션으로 표시) 중 하나를 선택 할 수 있다. 이 값은 생략 가능하며, 기본값은 **sdkText** 이다.
- **SHOWEXTENDEDNOTATION** 요소 : 만약 **STEREOTYPENAME** 으로 지정한 스테레오타입에 확장 노테이션 파일(.nxt)가 지정되어 있을 경우 확장 노테이션으로 표시할 것인가를 지정한다. **True**로 설정하면 **StarUML™**은 요소 프로토타입에 의해 생성된 요소의 뷰를 표시 할 때 확장 노테이션에 기술된 방식으로 그려준다. 이 값은 생략 가능하며, 기본값은 **False** 이다.
- **MODELPROPERTYLIST** 요소 : 이 요소는 모델 속성값 설정을 위한 **MODELPROPERTY** 요소들의 리스트이다.
- **MODELPROPERTY** 요소 : 요소를 생성할 때 설정할 모델 속성 값들을 명시한다. 속성의 이름을 나타내는 **name** 속성은 반드시 기재하여야 한다. **name**에 기재된 값이 **Base** 요소에 해당하는 모델의 속성명이 아니거나 명시된 값이 유효하지 않으면 요소 생성 시 제대로 생성되지 않을 수 있다. 사용 가능한 속성의 이름과 유효한 값의 범위를 알려면 **Chapter 4. "Using Open API"**를 참고하라.
- **VIEWPROPERTYLIST** 요소 : 이 요소는 뷰 속성값 설정을 위한 **VIEWPROPERTY** 요소들의 리스트이다.
- **VIEWPROPERTY** 요소 : 요소를 생성할 때 설정할 뷰의 속성 값들을 명시한다. 속성의 이름을 나타내는 **name** 속성은 반드시 기재하여야 한다. **name**에 기재된 값이 **Base** 요소에 해당하는 뷰의 속성명이 아니거나 명시된 값이 유효하지 않으면 요소가 제대로 생성되지 않을 수 있다. 사용 가능한 속성의 이름과 유효한 값의 범위를 알려면 **Chapter 4. "Using Open API"**를 참고하라.
- **TAGGEDVALUELIST** 요소 : 이 요소는 모델의 태그값(TaggedValue)을 설정하기 위한 **TAGGEDVALUE** 요소들의 리스트이다.
- **TAGGEDVALUE** 요소 : 요소를 생성할 때 설정할 모델의 태그값들을 명시한다. 태그값을 명시하기 위해서는 태그값을 정의하고 있는 태그 정의(TagDefinition)를 지정해 주어야 한다.
- **profile** 속성 : 태그정의(TagDefinition)가 속하는 프로파일명을 기술한다. 이 값은 생략 가능하며 생략할 경우 **ELEMENTPROTOTYPE**이 속해 있는 프로파일명으로 인식한다.
- **tagDefinitionSet** 속성 : 태그정의가 속하는 태그정의집합(TagDefinitionSet)명을 기술한다.
- **tagDefinition** 속성 : 태그정의의 이름을 기술한다.

## ModelPrototype

**MODELPROTOTYPE** 요소에는 생성할 모델 요소의 패턴을 정의하는 모델 프로토타입의 정보를 기술한다.

```
<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>
```

- **NAME** 요소 : 모델 프로토타입의 명칭이다. 이 명칭은 프로파일 내에서 유일해야 한다.
- **DISPLAYNAME** 요소 : 모델 프로토타입이 **[Add Model]** 메뉴 등 사용자 인터페이스 상에서 표시될 때의 이름이다.
- **ICON** 요소 : 모델 프로토타입이 **[Add Model]** 메뉴 등 사용자 인터페이스 상에서 표시될 때의 아이콘 파일의 이름이다. 모델 프로토타입의 아이콘 파일은 16 x 16 크기의 .BMP 포맷 형식의 이미지 파일이어야 한다. 아이콘 파일은 반드시 **Profile Document**가 있는 디렉토리에 위치해야 하며 경로명을 제외한 파일명만 기술하도록 한다.
- **BASEMODEL** 요소 : 모델 프로토타입의 복사본을 생성하기 위한 원래 **UML** 표준 요소의 명칭을 지정한다. 이 요소는 생략할 수 없으며, 값을 지정하지 않으면 모델 프로토타입을 인식하지 못한다. 사용할 수 있는 **UML** 표준 요소의 명칭은 **ElementPrototype**의 **BASEELEMENT** 요소 설명부에 나열한 것과 같으며, **Note**, **NoteLink** 등 뷰만 가질 수 있는 요소는 사용할 수 없다.
- **argument** 속성 : **Base** 모델 요소가 **Association**, **Pseudostate** 등 일부 요소의 경우 생성을 위한 인자가 필요할 수 있다. 이들 요소들은 인자에 따라 생성되는 모델 요소의 특정 속성값들이 미리 설정된다. 기본값은 0이며, 대부분의 경우 지정할 필요가 없다. 여기에 사용할 수 있는 값은 **ElementPrototype**의 **BASEELEMENT** 요소 설명부에 기술된 바와 같다.

- **STEREOTYPENAME** 요소 : 모델 프로토타입의 **Stereotype** 이름을 지정한다. 여기에 값을 설정해 두면 모델 요소 생성 시 **Stereotype** 속성 값으로 입력된다. 이 값은 생략 가능하다.
- **PROPERTYLIST** 요소 : 이 요소는 모델 속성값 설정을 위한 **PROPERTY** 요소들의 리스트이다.
- **PROPERTY** 요소 : 모델 요소를 생성할 때 설정할 모델 속성 값들을 명시한다. 속성의 이름을 나타내는 **name** 속성은 반드시 기재하여야 한다. **name**에 기재된 값이 **Base** 모델에 포함된 속성명이 아니거나 명시된 값이 유효하지 않으면 모델 요소가 제대로 생성되지 않을 수 있다. 사용 가능한 속성의 이름과 유효한 값의 범위를 알려면 **Chapter 4. "Using Open API"**를 참고하라.
- **TAGGEDVALUELIST** 요소 : 이 요소는 모델의 태그값(TaggedValue)를 설정하기 위한 **TAGGEDVALUE** 요소들의 리스트이다.
- **TAGGEDVALUE** 요소 : 모델 요소를 생성할 때 설정할 태그값들을 명시한다. 태그값을 명시하기 위해서는 태그값을 정의하고 있는 태그 정의(TagDefinition)를 지정해 주어야 한다.
- **profile** 속성 : 태그정의(TagDefinition)가 속하는 프로파일명을 기술한다. 이 값은 생략 가능하며 생략할 경우 **ELEMENTPROTOTYPE**이 속해 있는 프로파일명으로 인식한다.
- **tagDefinitionSet** 속성 : 태그정의가 속하는 태그정의집합(TagDefinitionSet)명을 기술한다.
- **tagDefinition** 속성 : 태그정의의 이름을 기술한다.
- **CONTAINERMODELLIST** 요소 : 모델 요소 생성 제약을 위한 **CONTAINERMODEL** 요소들의 리스트이다.
- **CONTAINERMODEL** 요소 : 모델 프로토타입이 기술하는 모델 요소를 생성할 수 있는 상위 모델 요소를 제약한다. 여기에 값이 설정되어 있을 경우 기술된 요소가 선택될 경우에만 **[Add Model]** 메뉴 하위에 생성 메뉴가 활성화된다.

## Palette

PALETTE 요소에는 새로 추가할 팔레트와 팔레트에 포함할 항목들을 기술한다.

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

- **NAME** 요소 : 팔레트의 명칭이다. 이 명칭은 프로파일 내에서 유일해야 한다.
- **DISPLAYNAME** 요소 : 팔레트가 화면에 표시될 때의 이름이다.
- **PALETTEITEMLIST** : 이 요소에는 팔레트에 속하는 팔레트 항목들을 나열한다.
- **PALETTEITEM** : 이 요소에 팔레트 항목으로 만들 요소명을 기술한다. 프로파일내에 정의된 요소 프로토타입명 또는 이미 툴에 포함되어 있는 표준 UML 요소명을 적는다.

## DiagramType

DIAGRAMTYPE 요소는 다이어그램 타입의 정보와 포함 요소들을 기술한다.

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

- **NAME** 요소 : 다이어그램타입 명칭이다. 이 명칭은 프로파일 내에서 유일해야 한다.
- **DISPLAYNAME** 요소 : 다이어그램타입이 **[Add Diagram]** 메뉴 등 사용자 인터페이스 상에서 표시될 때의 이름이다.
- **ICON** 요소 : 다이어그램타입이 **[Add Diagram]** 메뉴 등 사용자 인터페이스 상에서 표시될 때의 아이콘 파일의 이름이다. 다이어그램 타입의 아이콘은 16 x 16 크기의 .BMP 포맷 형식의 이미지 파일이어야 한다. 아이콘 파일은 반드시 **Profile Document**가 있는 디렉토리에 위치해야 하며 경로명을 제외한 파일명만 기술하도록 한다.
- **BASEDIAGRAM** 요소 : 다이어그램타입을 생성하기 위한 원래 UML 표준 다이어그램의 명칭을 지정한다. 이 요소는 생략할 수 없으며, 값을 지정하지 않으면 다이어그램타입을 인식하지 못한다. 사용할 수 있는 UML 표준 요소의 명칭은 다음과 같다.

Diagram names
ClassDiagram UseCaseDiagram SequenceDiagram SequenceRoleDiagram CollaborationDiagram CollaborationRoleDiagram StatechartDiagram ActivityDiagram ComponentDiagram DeploymentDiagram

## CompositeStructureDiagram

- **AVAILABLEPALETTELIST** 요소 : 다이어그램타입으로 다이어그램을 생성했을 때 활성화되는 팔레트들의 목록을 나열한다.
- **AVAILABLEPATTE** 요소 : 다이어그램타입으로 다이어그램을 생성했을 때 활성화되는 팔레트를 지정한다. 이 값은 프로파일내에서 정의한 팔레트의 이름을 사용하거나 StarUML™에 기본적으로 포함되어 있는 팔레트명을 사용한다. StarUML™에서 미리 정의하고 있는 팔레트는 다음과 같다.

## Built-in palette names

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

## UML 프로파일 등록하기

작성된 프로파일을 프로그램에서 인식하려면 프로파일과 관련 파일들을 StarUML™ 모듈 디렉토리(<install-dir>\modules) 하부 디렉토리로 옮겨야 한다. StarUML™은 프로그램 초기화 시에 모듈 디렉토리 하부를 검색하여 모든 프로파일들을 읽어 들인 후 프로그램에 자동 등록한다. 만약 프로파일 파일이 문법에 맞지 않게 잘 못 작성되어 있거나, 확장자가 .prf가 아닌 경우에는 프로파일을 정상적으로 읽어 들이지 않고 무시하게 될 것이다. 작성한 프로파일과 다른 모듈들과의 혼동을 피하기 위하여 모듈 디렉토리 밑에 별도의 서브디렉토리를 만들어 사용하는 것을 권고한다.

**노트:** 프로파일의 아이콘을 등록하려면, 프로파일과 동일한 이름의 아이콘 파일(.ico)을 만들어 프로파일이 있는 디렉토리에 두면 된다. 프로파일 아이콘은 **[Profiles]** 대화상자에서 프로파일 이름과 함께 프로파일 목록 리스트에 표시된다. 만약 프로파일의 이름과 동일한 이름의 아이콘 파일이 없으면 기본 아이콘으로 프로파일을 등록할 것이다.

**노트:** 프로파일을 더 이상 사용하지 않으려면 StarUML™ 모듈 디렉토리(<install-dir>\modules) 하부에서 삭제하면 된다.

## 확장 요소 개체 다루기

### 확장 요소 개요

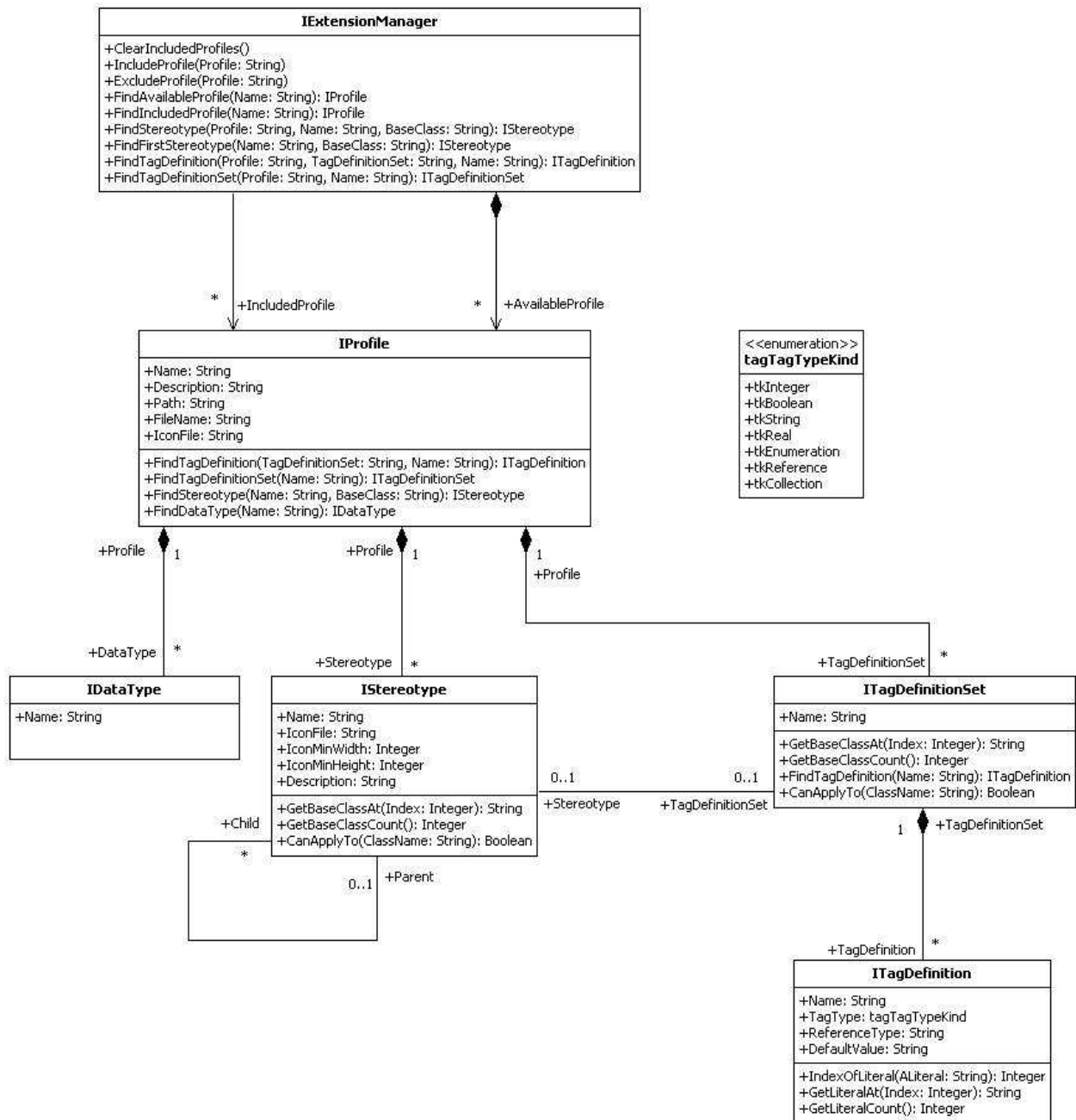
프로파일에 정의되는 확장 요소들(Extension Elements)은 StarUML™의 외부 API를 통해 접근할 수 있다. StarUML™의 확장 관련 COM 인터페이스는 실제 UML의 확장구조와 동일하게 구성되어 있으며 **IExtensionManager**에 의해 관리되어진다. 개발자가 **Extension Element**개체를 직접적으로 다루는 경우는 흔치 않다. 오히려 실제 확장되어진 모델 요소에서 스테레오타입이나 태그값(Tagged-Value)을 얻어오는 경우가 더 빈번할 것이다. 이런 경우에는 **IExtensionModel** 인터페이스에서 제공되는 메소드를 사용해야 한다. **IExtensibleModel** 인터페이스나 모델링 요소에 대한 자세한 내용은 **Chapter 4. "Using Open API"**를 참조하길 바란다.

앞서 언급한 것과 같이 확장 요소들은 실제 모델링 시에 생성되는 값이 아니고 확장 구조를 정의해 놓은 것이다. 이것들은 프로그램 또는 프로젝트 초기 시에 로드 되어 변경되어서는 안되기 때문에 이들 인터페이스들에 정의되어 있는 프로퍼티들은 대부분 읽기 전용이다.

Extension Element 개체들을 다루기 위한 인터페이스들은 다음과 같다.

- **IExtensionManager** : 프로그램에 등록된 프로파일들을 관리하고, Extension Element들을 검색하기 위한 메소드를 제공한다. IExtensionManager는 프로파일이나 프로파일에 정의된 Extension Element들을 액세스하기 위하여 최초로 접근하는 인터페이스이다.
- **IProfile** : 프로파일에 정의된 Extension Element들을 관리하고 이들에게 접근하고 검색하기 위한 메소드를 제공한다. 또한 프로파일 자체에 대한 정보도 포함하고 있다. IProfile은 프로파일에 정의된 Extension Element들을 IStereotype, ITagDefinition, IDataType 등의 컬렉션으로 유지한다.
- **IStereotype** : 스테레오타입에 대한 정보를 제공한다.
- **ITagDefinitionSet** : 태그정의집합에 대한 정보를 제공하고, 태그정의집합에 정의된 태그정의들을 ITagDefintion의 컬렉션으로 관리한다.
- **ITagDefintion** : 태그정의에 대한 정보를 제공한다.
- **IDataType** : 데이터타입에 대한 정보를 제공한다.

아래 다이어그램은 StarUML™의 확장 요소들에 대한 COM 인터페이스의 구조를 나타낸 것이다.



## IExtensionManager 다루기

프로파일과 Extension Elements들을 다루기 위해서는 먼저 **IExtensionManager** 인터페이스의 참조를 얻어와야 한다. IStarUMLApplication은 ExtensionManager개체에 접근하기 위한 프로퍼티를 제공한다. 다음 코드는 IExtensionManager를 가져오는 JScript 예제이다.

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

## 프로파일 포함(Include) 및 제외(Exclude)

**IExtensionManager**는 프로젝트에 프로파일을 포함하거나 제외시키는 메소드를 제공한다. **IncludeProfile()**은 현재 프로젝트에 입력된 프로파일을 포함시키고 **ExcludeProfile()**은 현재 프로젝트에서 입력된 프로파일을 제외시킨다. 메소드의 파라미터로 입력되는 프로파일은 반드시 시스템에 등록되어 있어야 한다. 만약 입력된 프로파일이 시스템에 존재하지 않거나 등록되어 있지 않으면 예외가 발생한다. 호출형식은 다음과 같다.

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

다음은 현재 프로젝트에서 "StandardProfile"을 제외시키는 JScript 예제이다.

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

## 프로파일에 정의된 확장 요소(Extension Element) 얻어오기

프로파일을 구성하는 확장 개체들은 **IProfile** 인터페이스를 통해 접근할 수 있다. **IProfile**은 확장 개체들의 인터페이스(**IStereotype**, **ITagDefinitionSet**, **IDataType**)에 접근하기 위하여 아래와 같이 컬렉션 접근 메소드를 제공한다. **GetStereotypeAt()**, **GetTagDefinitionSetAt()**, **GetDataTypeAt()**등에 쓰이는 Index 인자는 반드시 해당 컬렉션의 Count - 1 과 같거나 작아야 한다.

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

다음은 프로파일에 정의되어 있는 스테레오타입들을 순회하는 JScript 예제이다.

```
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...
    }
}
```

## 확장 요소(Extension Element) 검색하기

**IProfile** 인터페이스는 프로파일에 정의된 **Extension Element**들의 인터페이스를 검색하기 위한 메소드를 제공한다.

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

## 스테레오타입(Stereotype) 다루기

**IStereotype** 인터페이스는 프로파일에서 정의된 스테레오타입에 대한 정보를 제공한다. 스테레오타입의 이름과 설명, 아이콘 파일 등 스테레오타입의 일반적인 정보는 **IStereotype** 인터페이스의 읽기전용 프로퍼티를 통해서 얻을 수 있다. **IStereotype**에는 스테레오타입이 적용될 수 있는 UML요소들을 알기 위한 메소드가 정의되어 있는데, **GetBaseClassCount()**와 **GetBaseClassAt()**, **CanApplyTo()** 등이 그것이다.

**GetBaseClassCount()**와 **GetBaseClassAt()** 메소드는 스테레오타입이 적용될 수 있는 UML 요소들의 이름을 얻을 수 있게 해준다. **CanApplyTo()** 메소드는 인자로 받은 UML 요소가 해당 스테레오타입을 적용할 수 있는지를 논리형 값으로 리턴 해준다. 스테레오타입의 **BaseClass**에는 다이어그램상에서 도식할 수 있는 UML요소 뿐만 아니라 **UMLClassifier**와 같은 더 상위 요소를 지정할 수 도 있다. 이 경우 상위 요소의 모든 하위 요소들에게 해당 스테레오타입을 적용할 수 있다. 예를 들어 **UMLClassifier**가 **BaseClass**로 지정되어 있으면 **UMLClass**, **UMLInterface**, **UMLUseCase**, **UMLActor** 등 하위의 모든 요소이 **BaseClass**로 지정된 것과 동일한 의미를 가진다. 요소간의 상속구조는 **StarUML Application Model** 을 참고하기를 바란다.

**IExtensibleModel**의 **GetStereotype()**은 스테레오타입이 적용된 모델(Stereotyped-Model)로부터 **IStereotype** 개체를 리턴해준다. 모델의 스테레오타입이 프로파일에 정의되지 않았으면 null 값을 리턴 한다. 이 때에는 **IExtensibleModel**의 **StereotypeName** 속성으로 스테레오타입의 이름을 얻을 수 있다.

다음은 메세지 상자에 현재 선택된 모델의 스테레오 타입에 대한 설명을 보여주는 JScript 예제이다.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var selMgr = app.SelectionManager;

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

## 태그정의(TagDefinition) 다루기

**ITagDefinition** 인터페이스는 프로파일에서 정의된 태그정의에 대한 정보를 제공한다. **ITagDefinition**에는 다음과 같은 프로퍼티들이 있다.

프로퍼티	설 명
Name: String	태그 정의의 이름이다. 태그 정의의 이름은 태그 정의 집합(TagDefinitionSet)에서 유일해야 한다.
TagType: tagTagTypeKind	태그의 타입이다. 태그의 타입으로는 다음과 같은 것들을 지정할 수 있다. <ul style="list-style-type: none"> <li>tkInteger = 0 (정수)</li> <li>tkBoolean = 1 (논리값)</li> <li>tkString = 2 (문자열)</li> <li>tkReal = 3 (실수)</li> <li>tkEnumeration = 4 (열거형 값)</li> <li>tkReference = 5 (참조형 값)</li> <li>tkCollection = 6 (컬렉션)</li> </ul> 태그의 타입에 따라 모델에서 태그값을 얻어올 때 사용되는 메소드의 종류가 달라진다. IExtensibleModel에는 각 태그 타입 별로 태그 값을 얻어오는 메소드들이 정의되어 있다
ReferenceType: String	TagType이 tkReference 혹은 tkCollection인 경우, 태그값으로 지정될 수 있는 객체 레퍼런스의 타입을 가리킨다. 예를 들어, "UMLClass"라고 지정하면 Class 타입 만이 연결될 수 있다. 만약 프로파일 문서에서 ReferenceType에 대한 정의가 생략한 경우에는 "UMLModelElement"가 기본값으로 인식된다. TagType이 tkReferece 또는 tkCollection이 아니면 이 프로퍼티의 값은 의미가 없다.
DefaultValue: String	태그의 기본값을 정의한다. TagType이 tkEnumeration이면 열거순서에 해당하는 정수의 문자열 형식값이다. TagType이 tkReference나 tkCollection일 경우에는 기본값이 항상 null로 설정되므로 이 프로퍼티는 아무런 의미를 가지지 않는다.

다음은 태그의 기본값을 메세지 상자에 출력하는 JScript 예제이다.

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

[처음으로](#)

## 제 8 장. 메뉴 확장하기

### 메뉴 확장의 기본 개념

사용자가 정의한 Add-In의 기능을 호출하는 방법을 제공하기 위하여 StarUML™ 메뉴 시스템을 확장할 수 있는데, 이를 위해서 각 Add-In 개발자는 메뉴 확장 파일을 제공해야 한다. 이와 관련된 절차는 크게 다음과 같다.

1. 메뉴 확장 파일 작성
2. 메뉴 확장 파일 등록

Add-In 메뉴 확장 파일(\*.mnu)은 XML 형식의 텍스트 파일로, StarUML™에 플러그인(plug-in)되는 각 Add-In은 반드시 하나의 메뉴 확장 파일을 제공해야 한다. StarUML™은 이 메뉴 파일이 정의하는 내용을 바탕으로 어플리케이션의 메인 메뉴(main menu)와 팝업 메뉴(popup menu)를 확장하여 새로운 메뉴 항목들을 추가하고, 각 메뉴 항목이 클릭되었을 때 정의된 동작을 수행하거나 관련 있는 Add-In 개체로 메세지를 전파한다. StarUML™ Add-In 메뉴 확장 파일에는 다음과 같은 사항들이 정의될 수 있다.

- 추가될 새로운 메뉴 항목들
- 메인 메뉴 항목과 팝업 메뉴 항목의 구분
- 새로운 메뉴 항목이 추가될 StarUML™ 기본 메뉴 항목
- 메뉴 항목의 표현되는 이름과 핫-키
- 메뉴 항목이 활성화되거나 비활성화되는 시점
- 메뉴 항목이 선택되었을 때 실행되는 스크립트 파일
- 메뉴 항목이 선택되었을 때 Add-In 개체로 전달되는 메뉴 항목의 ID
- 메뉴 항목의 상위 그룹 메뉴 상의 위치
- 각 메뉴 항목에 대한 아이콘 파일

메뉴 확장 파일은 XML 형식으로 기술되며 'Well-formed document' 여야 하고 또한 유효(valid)해야 한다. 이 장에서는 메뉴 확장 파일이 유효하기 위해 준수해야 하는 XML DTD(Document Type Definition)와 메뉴 확장 파일의 구조를 설명하며, 또한 관련된 예제를 제공한다.

**노트:** Add-In 메뉴 확장 파일은 반드시 \*.mnu 확장자를 가져야 하며, StarUML™ 모듈 디렉토리(<install-dir>\modules) 하부 디렉토리에 존재해야 한다.

### 메뉴 확장 파일 작성하기

#### 메뉴 확장 파일의 DTD



StarUML™ Add-In 메뉴 확장 파일은 정의된 DTD를 준수하는 유효한 XML 문서여야 한다. 다음은 메뉴 확장 파일을 위해 정의된 DTD 전체이다.

```
<?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 MAINMENU (MAINITEM)*>
<!--ELEMENT POPUPMENU (POPUPITEM)*-->

<!--ELEMENT HEADER (NAME?, VERSION?, DESCRIPTION?, COMPANY?, COPYRIGHT?)>
<!--ELEMENT BODY (MAINMENU?, POPUPMENU?)>

<!--ELEMENT ADDINMENU (HEADER?, BODY)>
<!--ATTLIST ADDINMENU addInID CDATA #REQUIRED-->
```

**주의:** 모든 XML 요소(Element)의 이름은 대문자로만 기술해야 하며, 속성(Attribute)의 이름은 모두 소문자로 시작한다. 그리고 미리 정의된 심벌(Symbol) 값은 대문자와 '\_'(underscore)로 구성된다. 메뉴 파일 전체에 걸쳐 이와 같은 대소문자 규칙을 지켜야 하며 미리 정의된 심벌 값들을 정확하게 사용해야 한다.

## 메뉴 확장 파일의 개략적 구조

메뉴 확장 파일은 XML 문서의 규칙을 따르며, 사용자 정의 메뉴 항목들은 'ADDINMENU' 요소 내에 기술된다.

```
<?xml version="1.0" encoding="..."?>

<ADDINMENU addInID="...">
  <HEADER>...</HEADER>
  <BODY>...</BODY>
</ADDINMENU>
```

- **encoding** 속성: XML 문서의 인코딩 속성 값을 지정한다 (e.g. UTF-8, EUC-KR). 이 속성의 값에 대해서는 XML 관련 자료를 참조하기 바란다.
- **addInID** 속성: 각 Add-In의 고유한 명칭을 기술한다. 이것은 다른 Add-In과 구분될 수 있는 유일한 것이어야 하며 회사명이나 제품명과 연결하여 사용할 것을 권장한다. (e.g. StarUML.StandardAddIn)
- **HEADER** 요소: Add-In에 대한 기본 정보를 기술한다. **Header Contents** 섹션 참조.
- **BODY** 요소: 실제 메뉴 항목들에 대한 기술 부분이다. **Body Contents** 섹션 참조.

## Header Contents

메뉴 확장 파일의 Header 요소에서는 Add-In과 메뉴 파일에 대한 정보를 기술한다. Header 요소에 기술되는 내용은 실제 메뉴 항목의 구성에 영향을 미치지 않으며 생략이 가능하지만, 자기 설명적인 메뉴 확장 파일을 제공하기 위해 포함시킬 것이 권장된다.

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



- NAME 요소: Add-In의 설명적인 이름을 기술한다. (문자열 값)
- VERSION 요소: 버전 정보를 기술한다. (문자열 값)
- DESCRIPTION 요소: Add-In에 대한 짧은 설명을 기술한다. (문자열 값)
- COMPANY 요소: Add-In을 개발한 회사/개인의 정보를 기술한다. (문자열 값)
- COPYRIGHT 요소: 저작권에 대한 공지를 기술한다. (문자열 값)

## BODY CONTENTS

메뉴 확장 파일의 바디 요소에는 실제로 추가될 메뉴 항목(menu item)들이 기술된다. 따라서 주의 깊게 기술되어야 하는 부분이다.

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

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

Body 요소는 크게 메인 메뉴에 대한 정의와 팝업 메뉴에 대한 정의 부분으로 구성된다.

- MAINMENU 요소: 메인 메뉴에 추가될 메뉴 항목들이 아래에 기술된다.
- POPUPMENU 요소: 팝업 메뉴에 추가될 메뉴 항목들이 아래에 기술된다.
- MAINITEM 요소: 실제 메뉴 항목에 대한 정보를 기술한다. (메인 메뉴)
- POPUPITEM 요소: 실제 메뉴 항목에 대한 정보를 기술한다. (팝업 메뉴)

메인 메뉴의 항목과 팝업 메뉴의 항목을 구분하여 기술하는데, 각 Add-In이 제공하는 기능에 따라 메뉴 항목을 메인 메뉴에만 추가하거나 팝업 메뉴에만 추가할 수 있다. MAINMENU 요소와 POPUPMENU 요소는 각각 생략될 수 있지만 두 요소 중에서 최소한 하나의 요소는 존재해야 한다. 동일한 기능을 수행하는 메뉴 항목이 메인 메뉴와 팝업 메뉴 두 곳에 모두 추가되어야 하는 경우에는, MAINMENU 와 POPUPMENU에 모두 기술한다. 이 경우 두 항목의 script 속성이나 actionId 속성을 동일하게 지정하면 된다. 예외적으로, StarUML™의 기본 메뉴 항목 중에서 **[Format]**, **[Unit]** 항목과 같이 메인 메뉴와 팝업 메뉴에서 공유되는 기본 메뉴 항목의 하위 메뉴 항목을 추가하는 경우에는 MAINMENU 에만 기술하도록 한다.

## MAINMENU

MAINMENU 요소는 여러 개의 MAINITEM 요소를 포함할 수 있다. 각각의 MAINITEM 요소는 실제로 하나의 메인 메뉴 항목이 된다. 하위 메뉴 항목을 가지는 그룹 메뉴 항목을 정의하는 경우 MAINITEM 요소는 다시 여러 개의 MAINITEM 요소를 포함할 수 있다.

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

속성	설명	값의 범위	생략 여부
base	StarUML™ 기본 메뉴 항목 중에서 이 메뉴 항목이 추가될 상위 메뉴 항목을 지정한다. 이 속성은 다른 MAINITEM 요소의 하위 MAINITEM 요소인 경우에는 의미가 없다.	FILE, EDIT, FORMAT, MODEL, VIEW, TOOLS, HELP, UNITS, IMPORT, EXPORT, NEW_TOP 중의 하나여야 한다. *	이 값이 생략되면 <b>[Tools]</b> 메뉴의 하위 메뉴 항목으로 추가된다.
caption	메뉴 항목의 표현되는 이름을 기술한다. 이 값은 핫-키(Hot key or Shortcut)를 포함할 수 있다. 핫-키를 정의하는 방법은 이 속성의 값 다음에 '&' 와 함께 핫-키로 지정할 문자를 기록하는 것이다. 단, StarUML™ 프로그램은 정의된 핫-키가 다른 메뉴 항목과 중복되는지 여부를 검사하지 않는다.	문자열 값	이 값은 생략될 수 없다.
index	상위 메뉴의 하위 메뉴 항목들에 대한 이 메뉴 항목의 순서를 지정한다. 예를 들어 이 값이 '0' 이면 base 메뉴 항목의 첫 번째 하위 메뉴 항목이 된다. 이 속성의 값이 다른 메뉴 항목의 값과 충돌되는 경우 정확하게 표현되지 않을 가능성이 있다.	0 이상의 정수 값	일반적으로 생략되며, 이 값이 생략되면 Add-In이 등록된 순서대로 추가된다.
beginGroup	이 메뉴 항목의 앞에 구분자(separator)를 먼저 추가할 지를 지정한다.	TRUE, FALSE 중의 하나여야 한다.	생략되면 FALSE
script	이 메뉴 항목이 선택되었을 때 스크립트를 실행하게 하려면 이 속성의 값에 해당 스크립트 파일의 경로와 이름을 지정한다. 이 때 경로는 Add-In 프로그램이 위치하는 폴더에 대한 상대 경로 값이다. 이 속성의 값으로 웹 사이트의 URL을 지정할	문자열 값	생략 가능

	수도 있다.		
actionId	이 메뉴 항목이 선택된 경우 COM 개체에서 이것을 처리하려는 경우 이 속성의 값으로 '0' 보다 큰 정수를 지정한다. Add-In이 하나 이상의 메뉴 항목을 추가하는 경우, 각 메뉴 항목의 actionId 값을 다르게 지정하여 이들을 구분할 수 있다.	0 보다 큰 정수 값	생략 가능
availableWhen	메뉴 항목이 활성화(Enabled)되는 경우를 지정한다.	ALWAYS, PROJECT_OPENED, MODEL_SELECTED, VIEW_SELECTED, UNIT_SELECTED, DIAGRAM_ACTIVATED 중의 하나여야 한다. **	생략되면 PROJECT_OPENED가 지정된다.
iconFile	이 메뉴 항목에 대한 아이콘 파일이 있는 경우 아이콘 파일에 대한 경로와 파일 이름을 지정한다. 이 때 경로는 Add-In 프로그램이 위치하는 폴더에 대한 상대 경로 값이다.	문자열 값	생략 가능

**노트:** 메뉴 항목이 하위 메뉴 항목들을 그룹화(grouping)하는 경우가 아닌 경우에는 script, actionId 속성 중에서 최소한 하나의 값이 지정되어야 한다.

#### \* base 속성 값의 범위

- FILE: **[File]** 메뉴의 하위 메뉴 항목으로 추가된다.
- EDIT: **[Edit]** 메뉴의 하위 메뉴 항목으로 추가된다.
- FORMAT: **[Format]** 메뉴의 하위 메뉴 항목으로 추가된다.
- MODEL: **[Model]** 메뉴의 하위 메뉴 항목으로 추가된다.
- VIEW: **[View]** 메뉴의 하위 메뉴 항목으로 추가된다.
- TOOLS: **[Tools]** 메뉴의 하위 메뉴 항목으로 추가된다. (default)
- HELP: **[Help]** 메뉴의 하위 메뉴 항목으로 추가된다.
- UNITS: **[File]->[Unit]** 메뉴의 하위 메뉴 항목으로 추가된다.
- IMPORT: **[File]->[Import]** 메뉴의 하위 메뉴 항목으로 추가된다.
- EXPORT: **[File]->[Export]** 메뉴의 하위 메뉴 항목으로 추가된다.
- NEW\_TOP: 새로운 최상위 메인 메뉴 항목을 생성하는 경우에는 이 값을 사용한다.

#### \*\* availableWhen 속성 값의 범위

- ALWAYS: StarUML™ 어플리케이션이 실행되는 동안 항상 활성화된다.
- PROJECT\_OPENED: 프로젝트 요소가 있는 경우에 항상 활성화된다. (default)
- MODEL\_SELECTED: 모델 요소가 선택된 경우에만 활성화된다.
- VIEW\_SELECTED: 뷰 요소가 선택된 경우에만 활성화된다.
- UNIT\_SELECTED: 유닛으로 분리된 요소가 선택되었을 때만 활성화된다.
- DIAGRAM\_ACTIVATED: 열려있는 다이어그램이 있으면 활성화된다.
- ALWAYS: Enabled as long as the StarUML™ application is running.
- PROJECT\_OPENED: Enabled when a project element is present. (default)
- MODEL\_SELECTED: Enabled when a model element is selected.
- VIEW\_SELECTED: Enabled when a view element is selected.
- UNIT\_SELECTED: Enabled when a unit element is selected.
- DIAGRAM\_ACTIVATED: Enabled when a diagram is opened.

## POPUPMENU

POPUPMENU 요소는 여러 개의 POPUPITEM 요소를 포함할 수 있다. 각각의 POPUPITEM 요소는 실제로 하나의 팝업 메뉴 항목이 된다. 하위 메뉴 항목을 가지는 메뉴 항목을 정의하는 경우 POPUPITEM 요소는 다시 여러 개의 POPUPITEM 요소를 포함할 수 있다.

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

속성	설명	값의 범위	생략 여부
location	팝업 메뉴 항목이 추가될 팝업 메뉴 시스템을 지정한다. 이 속성은 다른 POPUPITEM 요소의 하위 POPUPITEM 요소인 경우에는 의미가 없다.	EXPLORER, DIAGRAM, BOTH 중의 하나여야 한다. *	이 값이 생략되면 BOTH로 지정된다.
	메뉴 항목의 표현되는 이름을 기술한다. 이 값은 핫-키(Hot key or Shortcut)를 포함할 수 있다. 핫-키를 정의하는 방법은 이 속성의 값		

caption	다음에 "&" 와 함께 핫-키로 지정할 문자를 기록하는 것이다. 단, StarUML™ 프로그램은 정의된 핫-키가 다른 메뉴 항목과 중복되는지 여부를 검사하지 않는다.	문자열 값	이 값은 생략될 수 없다.
index	상위 메뉴의 하위 메뉴 항목들에 대한 이 메뉴 항목의 순서를 지정한다. 예를 들어 이 값이 '0' 이면 base 메뉴 항목의 첫 번째 하위 메뉴 항목이 된다. 이 속성의 값이 다른 메뉴 항목의 값과 충돌되는 경우 정확하게 표현되지 않을 가능성이 있다.	0 이상의 정수 값	일반적으로 생략되며, 이 값이 생략되면 Add-In이 등록된 순서대로 추가된다.
beginGroup	이 메뉴 항목의 앞에 구분자(separator)를 먼저 추가할 지를 지정한다.	TRUE, FALSE 중의 하나여야 한다.	생략되면 FALSE
script	이 메뉴 항목이 선택되었을 때 스크립트를 실행하게 하려면 이 속성의 값에 해당 스크립트 파일의 경로와 이름을 지정한다. 이 때 경로는 Add-In 프로그램이 위치하는 폴더에 대한 상대 경로 값이다. 이 속성의 값으로 웹 사이트의 URL을 지정할 수도 있다.	문자열 값	생략 가능
actionId	이 메뉴 항목이 선택된 경우 COM 개체에서 이것을 처리하려는 경우 이 속성의 값으로 '0' 보다 큰 정수를 지정한다. Add-In이 하나 이상의 메뉴 항목을 추가하는 경우, 각 메뉴 항목의 actionId 값을 다르게 지정하여 이들을 구분할 수 있다.	0 보다 큰 정수 값	생략 가능
availableWhen	메뉴 항목이 활성화되는 경우를 지정한다.	ALWAYS, PROJECT_OPENED, MODEL_SELECTED, VIEW_SELECTED, UNIT_SELECTED, DIAGRAM_ACTIVATED 중의 하나여야 한다. **	생략되면 PROJECT_OPENED가 지정된다.
iconFile	이 메뉴 항목에 대한 아이콘 파일이 있는 경우 아이콘 파일에 대한 경로와 파일 이름을 지정한다. 이 때 경로는 Add-In 프로그램이 위치하는 폴더에 대한 상대 경로 값이다.	문자열 값	생략 가능

**노트:** 메뉴 항목이 하위 메뉴 항목들을 그룹화(grouping)하는 경우가 아닌 경우에는 script, actionId 속성 중에서 최소한 하나의 값이 지정되어야 한다.

#### \* location 속성 값의 범위

- EXPLORER: 브라우저 창의 **모델 탐색기** 팝업 메뉴로 추가된다.
- DIAGRAM: **다이어그램**의 팝업 메뉴로 추가된다.
- BOTH : **모델 탐색기** 팝업 메뉴와 **다이어그램** 팝업 메뉴 모두에 추가된다. (default)

**\*\* availableWhen 속성 값의 범위** - MAINMENU 요소의 경우와 동일하다.

## 메뉴 확장 파일의 예제

다음의 예는 StarUML™ 프로그램을 설치했을 때 함께 설치되는 StarUML™ 기본 확장팩의 메뉴 파일의 전체 내용이다.

```
<?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"/>
      </MAINITEM>
    </MAINMENU>
  </BODY>
</ADDINMENU>
```

## 메뉴 확장 파일 등록하기

작성된 메뉴 확장 파일을 프로그램에서 인식하려면 메뉴 확장 파일을 StarUML™ 모듈 디렉토리(<install-dir>\modules) 하부 디렉토리로 옮겨야 한다. StarUML™은 프로그램 초기화 시에 모듈 디렉토리 하부를 검색하여 모든 메뉴 확장 파일들을 읽어 들인 후 프로그램에 자동 등록한다. 만약 메뉴 확장 파일이 문법에 맞지 않게 잘 못 작성되어 있거나, 확장자가 .mnu가 아닌 경우에는 메뉴 확장 파일을 정상적으로 읽어 들이지 않고 무시하게 될 것이다.

메뉴 확장 파일은 가급적이면 모듈 디렉토리 하부에 메뉴 확장 파일이 명시하고 있는 모듈을 담기 위한 디렉토리를 만들고 거기에 저장하기를 권장한다.

**노트:** 메뉴 확장을 더 이상 사용하지 않으려면 StarUML™ 모듈 디렉토리(<install-dir>\modules) 하부에서 삭제하면 된다.

[처음으로](#)

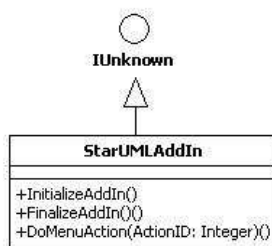
## 제 9 장. Add-In COM Object 사용하기

### Add-In COM Object의 기본 개념

StarUML™에 새로운 기능을 추가하고자 할 때 이전 '**Chapter 3. Hello world Example**'에서 보았듯이 간단한 Script 코드를 정의할 수도 있지만, 좀 더 복잡하거나 유용한 기능을 제공하기 위해서는 COM 개체를 지원하는 프로그램 개발 환경을 사용하는 것이 나을 것이다. StarUML™ Add-In COM Object를 구현할 때 Visual C++, Delphi, C#, Visual Basic 등과 같이 COM 기술을 지원하는 어떤 프로그램 환경을 선택하든지 상관이 없다.

StarUML™ Add-In COM Object를 구현할 때 지켜야 하는 가장 중요한 점은 StarUML™이 정의하고 있는 **IStarUMLAddIn** 인터페이스를 구현해야 한다는 것이다.

**IStarUMLAddIn** 인터페이스는 위의 그림과 같이 IUnknown을 상속하며 **InitializeAddIn()**, **FinalizeAddIn()**, **DoMenuAction()** 세 개의 추가적인 인터페이스 메소드를 정의하고 있다.



### IStarUMLAddIn 인터페이스 메소드

**IStarUMLAddIn** 인터페이스를 구현할 때 정의해야 하는 메소드는 다음과 같다.

메소드	설명
InitializeAddIn()	<b>InitializeAddIn()</b> 메소드는 StarUMLApplication 개체가 각 Add-In COM Object를 생성한 후에 호출하는 초기화 메소드이다. 이것은 StarUML™의 이벤트 수신 등록과 같이 Add-In COM Object의 초기화에 필요한 동작을 정의할 때 사용한다.
FinalizeAddIn()	<b>FinalizeAddIn()</b> 메소드는 StarUMLApplication 개체가 Add-In COM Object에 대한 참조를 끊기 바로 전에 호출하는 메소드이다. 이것은 StarUML™의 이벤트 수신 해제와 같이 Add-In COM Object의 소멸 전에 필요한 동작을 정의할 때 사용한다.
DoMenuAction(ActionID: Integer)	<b>DoMenuAction()</b> 메소드는 8장 ' <b>Extending Menu</b> '에서 보았듯이 각 Add-In이 정의한 확장 메뉴 항목이 사용자에게 의해 선택되었을 때 호출되며, 메뉴 확장 파일에서 정의한 각 메뉴 항목의 ' <b>actionId</b> ' 값이 인자로 전달된다.

### Add-In COM Object 예제

다음은 IStarUMLAddIn 인터페이스를 구현하는 StarUML™ Add-In COM Object의 간단한 예를 보이고 있다. 이것은 델파이 파스칼 문법으로 정의된 것이다.

```

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;

```

## Add-In 설명 파일 작성하기

### Add-In 설명 파일의 기본 개념

Add-In Description 파일(\*.aid)은 XML 형식의 텍스트 파일로, StarUML™에 플러그인(plug-in)되는 각 Add-In은 반드시 하나의 Add-In 파일을 제공해야 한다. StarUML™은 Add-In Description 파일이 정의하는 내용을 바탕으로 Add-In을 레지스터리에 등록하고 초기화하며, Add-In과 연결된 메뉴 확장 파일을 초기화한다.

**노트:** Add-In Description 파일은 반드시 \*.aid 확장자를 가져야 하며, StarUML™ 모듈 디렉토리(<install-dir>\modules) 하부 디렉토리에 존재해야 한다.

### Add-In Description 파일의 전체 구조

Add-In Description 파일은 XML 문서의 규칙을 따르며, 사용자 정의 항목들은 'ADDIN' 요소 내에 기술된다.

```

<?xml version="1.0" encoding="..."?>

<ADDIN>
  <NAME>...</NAME>
  <DISPLAYNAME>...</DISPLAYNAME>
  <COMOBJ>...</COMOBJ>
  <FILENAME>...</FILENAME>
  <COMPANY>...</COMPANY>
  <COPYRIGHT>...</COPYRIGHT>
  <HELPPFILE>...</HELPPFILE>
  <ICONFILE>...</ICONFILE>
  <ISACTIVE>...</ISACTIVE>
  <MENUFILE>...</MENUFILE>
  <VERSION>...</VERSION>
  <MODULES>
    <MODULEFILENAME>...</MODULEFILENAME>
  </MODULES>
</ADDIN>

```

- **encoding** 속성: XML 문서의 인코딩 속성 값을 지정한다 (e.g. UTF-8, EUC-KR). 이 속성의 값에 대해서는 XML 관련 자료를 참조하기 바란다.
- **NAME** 요소: Add-In의 이름을 기술한다. (문자열 값)
- **DISPLAYNAME** 요소 : StarUML UI상에서 보여질 Add-In의 이름 (문자열 값)
- **COMOBJ** 요소: Add-In의 형태가 COM Add-In인 경우에만 사용되며, COM의 ProgID를 기술한다. (문자열 값)
- **FILENAME** 요소 : Add-In 파일명을 기술한다. (문자열 값)
- **COMPANY** 요소: Add-In을 개발한 회사/개인의 정보를 기술한다. (문자열 값)
- **COPYRIGHT** 요소: 저작권에 대한 공지를 기술한다. (문자열 값)
- **HELPPFILE** 요소: Add-In의 Help가 기술되어진 URL 경로를 기술한다. (문자열 값)

- **ICONFILE** 요소: StarUML 화면상에서 **Add-In**과 연결될 아이콘 파일명을 기술한다. (문자열 값)
- **ISACTIVE** 요소: 이 요소의 값이 **true**이면 StarUML은 자동으로 **Add-In**을 로딩하고, **false**이면 로딩하지 않는다. (불린 값)
- **MENUEFILE** 요소: **Add-In**과 연결된 메뉴 확장 파일명을 기술한다. (문자열 값)
- **VERSION** 요소: **Add-In**의 버전 정보를 기술한다. (문자열 값)
- **MODULES/MODULEFILENAME** 요소: 하나의 **Add-In**이 여러개의 또 다른 **COM**을 사용할 경우에 **Add-In**이 최초로 실행 되기 전에 관련 된 **COM**이 등록될수 있도록 사용하는 **COM**의 파일명을 **MODULE** 요소에 기술한다. (문자열 값)

## Add-In 설명 파일 등록하기

작성된 **Add-In Description** 파일을 StarUML에서 자동으로 인식하려면 **Add-In Description** 파일이 StarUML™ 모듈 디렉토리(<install-dir>\modules) 하부 디렉토리로 옮겨야 한다. StarUML™은 프로그램 초기화 시에 모듈 디렉토리 하부를 검색하여 모든 **Add-In Description** 파일들을 읽어 들인 후 프로그램에 자동 등록한다. 만약 **Add-In Description** 파일이 문법에 맞지 않게 잘 못 작성되어 있거나, 확장자가 .aid가 아닌 경우에는 **Add-In Description** 파일을 정상적으로 읽어 들이지 않고 무시하게 될 것이다.

**Add-In Description** 파일은 가급적이면 모듈 디렉토리 하부에 새로운 디렉토리를 만들고 거기에 저장하기를 권장한다.

**노트:** **Add-In Description** 파일을 더 이상 사용하지 않으려면 StarUML™ 모듈 디렉토리(<install-dir>\modules) 하부에서 삭제하면 된다.

## 옵션 확장

### 옵션 확장의 기본 개념

StarUML™에서는 사용자들이 StarUML™의 환경과 세부적인 기능을 조절할 수 있도록 옵션 편집을 지원한다. 이런 옵션들은 StarUML™ 애플리케이션 자체 뿐만 아니라 써드파티 벤더들에 의해 제공되는 **Add-in**들에게도 필요하다. StarUML™의 옵션 확장은 **Add-in**들에게 별도의 구현 과정 없이 옵션 설정 기능을 가능하게 해준다. 이를 위해서 **Add-in** 개발자는 텍스트 파일로 간단하게 옵션에 대한 정의만 기술하고 이 옵션 정의 파일을 레지스트리에 등록해 주지만 하면 된다. 등록된 옵션 정의는 프로그램 초기화 시에 로드되며 사용자가 옵션 다이얼로그를 띄웠을 때 트리뷰와 인스펙터에 표시된다. 옵션 확장 기능을 통해서 개발자들은 **Add-in** 구현을 위한 시간과 노력을 절감할 수 있고, 사용자들에게 옵션 설정을 위한 일관된 유저 인터페이스를 제공할 수 있다.

**Add-in**에서 옵션 설정을 지원하기 위해서는 다음과 같은 절차가 필요하다.

1. **Add-in**에 적용할 옵션 항목을 정의하기 위해 옵션 스키마 문서(.opt)를 작성한다.
2. 옵션 스키마 파일을 **Add-in**이 설치된 디렉토리에 복사한다.

### 옵션의 계층 구조

StarUML™은 애플리케이션과 각 **Add-in**에서 작성된 많은 옵션 항목들을 통합적으로 관리하기 위하여 다음과 같이 계층적으로 구성하였다.

- **Option Schema** : **Option Schema**는 **Option** 구성의 최상위의 분류로 옵션 스키마 파일의 단위가 되고, 옵션 다이얼로그의 왼쪽 트리뷰에서 최상위 계층에 폴더 모양의 아이콘으로 나타난다. **Option Schema**는 StarUML™ 애플리케이션과 각 **Add-in**에 대응되는 옵션 구조의 가장 큰 단위이다.
- **Option Category** : **Option Category**는 **Option Schema**를 기능별로 크게 분류한 것으로 옵션 다이얼로그의 왼쪽 트리뷰에서 하위 계층으로 표시된다. **Option Category**는 옵션 다이얼로그에서 오른쪽의 옵션 인스펙터의 표시 단위가 된다.
- **Option Classification** : **Option Classification**은 **Option Category**를 다시 세부적으로 구분한 것으로 옵션 다이얼로그의 옵션 인스펙터에서 분류 행에 해당된다. **Option Classification**은 실제 옵션 값을 편집할 수 있는 여러 개의 **Option Item**을 가진다.
- **Option Item** : **Option Item**은 실제 옵션 값을 편집하는 단위이며, 옵션 다이얼로그의 옵션 인스펙터에서 하나의 행에 해당된다.

## 옵션 스키마 작성하기

옵션 스키마의 구조와 옵션 항목을 정의하기 위한 옵션 스키마 파일은 확장자가 .opt 인 XML 포맷의 텍스트 파일이다. 옵션 스키마에 대한 내용은 **OPTIONSHEMA** 요소 내에 기술되며, 구문이나 내용에 오류가 없도록 해야 한다.

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

- **encoding** 속성 : XML 문서의 인코딩 속성 값을 지정한다 (e.g. UTF-8, EUC-KR). 이 속성의 값에 대해서는 XML 관련 자료를 참조하기 바란다.
- **id** 속성 (**OPTIONSHEMA** 요소) : 각 옵션스키마의 고유한 명칭을 기술한다. 이것은 다른 옵션 스키마와 구분될 수 있는 유일한 것이어야 한다.
- **HEADER** 요소 : **Header Contents** 섹션 참조
- **BODY** 요소 : **Body Contents** 섹션 참조

## Header Contents

**HEADER** 부분에는 옵션 스키마의 제목과 상세설명 등 옵션 스키마에 대한 개괄적인 정보를 기술한다. **HEADER** 부분의 형식은 다음과 같다.

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

- **CAPTION** 요소 : 옵션 스키마의 타이틀이고 옵션 다이얼로그의 트리뷰에서 노드에 캡션으로 표시된다.
- **DESCRIPTION** 요소 : 옵션 스키마에 대한 간략한 설명으로 옵션 다이얼로그에서 항목 설명 텍스트 박스에 표시된다.

## Body Contents

BODY 부분에는 실제로 옵션 스키마에 포함되는 모든 옵션 항목들을 계층적으로 구분하여 정의한다.

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

- **OPTIONCATEGORY** 요소 : 옵션 카테고리에 대한 구조를 정의한다.
  - **CAPTION** 요소 : 옵션 카테고리의 캡션으로 옵션 다이얼로그의 트리뷰에서 노드에 표시된다.
  - **DESCRIPTION** 요소 : 옵션 카테고리에 대한 간략한 설명으로 옵션 다이얼로그에서 해당 카테고리를 선택했을 때 항목 설명 텍스트 상자에 표시된다.
- **OPTIONCLASSIFICATION** 요소 : 옵션 클래스피케이션에 대한 구조를 정의한다.
  - **CAPTION** 요소 : 옵션 클래스피케이션의 캡션으로 옵션 다이얼로그의 인스펙터에서 분류 행의 이름으로 표시된다
  - **DESCRIPTION** 요소 : 옵션 클래스피케이션의 대한 간략한 설명으로 옵션 다이얼로그에서 해당 클래스피케이션을 선택했을 때 항목 설명 텍스트 상자에 표시된다.
- **OPTIONITEM** 요소 : 여러 개의 옵션 항목들을 정의한다. **Option Item Definition** 섹션 참조

## 옵션 항목 정의

**OPTIONCLASSIFICATION** 요소 하부에는 여러 개의 옵션 항목을 정의할 수 있다. 옵션 항목의 타입은 단순한 문자열만이 아니라 정수, 실수, 논리형, 열거형 등 몇 가지 형태로 명시화되어 있다. 옵션 다이얼로그에서는 옵션 항목의 타입에 따라 옵션 값 입력을 위한 정보를 제공하거나 입력할 수 있는 값을 제한한다.

StarUML™에서 제공하는 옵션 항목의 타입은 다음과 같다.

옵션 항목 타입	XML 요소 이름	옵션 다이얼로그에서의 입력 형태
정수 (integer)	OPTIONITEM-INTEGGER	정수 범위의 숫자만 입력 할 수 있다.
실수 (real)	OPTIONITEM-REAL	실수 범위의 숫자만 입력 할 수 있다.
문자열 (string)	OPTIONITEM-STRING	문자열 값을 입력 받는다.
논리형 (boolean)	OPTIONITEM-BOOLEAN	체크박스로 True / False를 입력 받는다.
텍스트 (text)	OPTIONITEM-TEXT	입력 시 텍스트박스가 팝업 되어 거기에 여러 줄을 입력할 수 있다.
열거형 (enumeration)	OPTIONITEM-ENUMERATION	<b>OPTION-ENUMERATIONITEM</b> 으로 정의된 여러 개의 열거항목을 콤보박스에서 선택할 수 있다.
글자체 (font name)	OPTIONITEM-FONTNAME	시스템에 설치된 글자체 목록 중에서 하나를 선택할 수 있다.
파일명 (file name)	OPTIONITEM-FILENAME	파일명을 직접 입력하거나 파일 열기 다이얼로그를 통해서 파일을 지정한다.
경로명 (path name)	OPTIONITEM-PATHNAME	디렉토리(directory)를 직접 입력하거나 경로 지정 다이얼로그를 통해서 경로를 지정한다.
색상 (color)	OPTIONITEM-COLOR	색상 콤보박스에서 색상을 선택하거나 색상 다이얼로그에서 색을 지정한다.
범위형 (range)	OPTIONITEM-RANGE	지정된 범위 내의 정수 값을 입력 받는다. 스펀버튼으로 지정된 스텝만큼 값을 변경시킬 수 있다.

아래는 옵션 스키마 파일에서 **OPTIONCLASSIFICATION** 하부에 옵션 항목 정의 포맷을 보인 것이다.

```
<OPTIONCLASSIFICATION>
```



```

<OPTIONITEM-INTEGGER key="...">
  <CAPTION>...</CAPTION>
  <DESCRIPTION>...</DESCRIPTION>
  <DEFAULTVALUE>...</DEFAULTVALUE>
</OPTIONITEM-INTEGGER>
<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>

```

- **key** 속성 (모든 **OPTIONITEM** 요소들) : 옵션 항목의 고유한 키 값으로 옵션 스키마내에서 유일해야 한다. **COM** 인터페이스를 통해서 옵션 값을 읽어 올 때 파라미터로 사용된다.
- **CAPTION** 요소 : 옵션 다이얼로그의 입력 행에서의 캡션이다.
- **DESCRIPTION** 요소 : 옵션 항목에 대한 간략한 설명을 기술하는 것으로 옵션 다이얼로그에서 해당 옵션 항목이 선택되었을 때 항목설명 텍스트 상자에 표시된다.
- **DEFAULTVALUE** 요소 : 옵션 항목의 기본값으로 각 타입별로 아래와 같이 유효한 범위내의 값으로 입력되어야 한다. **DefaultValue**가 해당 타입의 적합한 값이 아니면 옵션 다이얼로그에서 값을 편집할 수 없다.

옵션 항목 타입	Default Value의 유효 범위
OPTIONITEM-INTEGGER	-2147483648 ~ 2147483647 사이의 정수
OPTIONITEM-REAL	정수 또는 소수점 형태의 실수
OPTIONITEM-STRING	문자열
OPTIONITEM-BOOLEAN	True 또는 False
OPTIONITEM-TEXT	문자열
OPTIONITEM-ENUMERATION	ENUMERATIONITEM 으로 정의된 문자열 String value defined by ENUMERATIONITEM

OPTIONITEM-FONTNAME	글자체 예)굴림								
OPTIONITEM-FILENAME	경로를 포함하는 파일명 형식 또는 빈 문자열 예) C:\My Document\Default.xml								
OPTIONITEM-PATHNAME	적합한 경로명 또는 빈 문자열 예) C:\My Document								
OPTIONITEM-COLOR	<p>다음과 같은 형식의 문자열 \${W}{B}{G}{R}</p> <table border="1"> <tr> <td>{W}</td><td>예약된 부분 . 무조건 00</td></tr> <tr> <td>{B}</td><td>색 구성 중 Blue 값의 비율 . 0 ~ 255 사이의 16 진수 값 (00 ~ FF)</td></tr> <tr> <td>{G}</td><td>색 구성 중 Green 값의 비율 . 0 ~ 255 사이의 16 진수 값 (00 ~ FF)</td></tr> <tr> <td>{R}</td><td>색 구성 중 Red 값의 비율 . 0 ~ 255 사이의 16 진수 값 (00 ~ FF)</td></tr> </table> <p>예) \$00FF0000 , \$00A0A0A0, \$00FF00FF</p>	{W}	예약된 부분 . 무조건 00	{B}	색 구성 중 Blue 값의 비율 . 0 ~ 255 사이의 16 진수 값 (00 ~ FF)	{G}	색 구성 중 Green 값의 비율 . 0 ~ 255 사이의 16 진수 값 (00 ~ FF)	{R}	색 구성 중 Red 값의 비율 . 0 ~ 255 사이의 16 진수 값 (00 ~ FF)
{W}	예약된 부분 . 무조건 00								
{B}	색 구성 중 Blue 값의 비율 . 0 ~ 255 사이의 16 진수 값 (00 ~ FF)								
{G}	색 구성 중 Green 값의 비율 . 0 ~ 255 사이의 16 진수 값 (00 ~ FF)								
{R}	색 구성 중 Red 값의 비율 . 0 ~ 255 사이의 16 진수 값 (00 ~ FF)								
OPTIONITEM-RANGE	MINVALUE와 MAXVALUE에 명시된 최소값과 최대값 사이의 정수								

- ENUMERATIONITEM 요소 : 열거형 타입의 옵션 항목(OPTION-ENUMERATION)에서 선택할 수 있는 열거 항목이다. OPTION-ENUMERATION에서 최소 한 개 이상의 EnumerationItem이 정의되어야 한다.
- MINVALUE 요소 : 범위형 타입의 옵션 항목(OPTION-RANGE)에서 유효한 최소 정수 값이다.
- MAXVALUE 요소 : 범위형 타입의 옵션 항목(OPTION-RANGE)에서 유효한 최대 정수 값이다.
- STEP 요소 : 범위형 타입의 옵션 값 편집 시 스텝 버튼을 클릭했을 때의 증가치 정수 값이다.

다음은 StarUML™의 기본 옵션 스키마의 일부분을 예제로 보인 것이다.

```
<?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>
```

## 옵션 스키마 등록하기

작성된 옵션 스키마 파일을 프로그램에서 인식하려면 메뉴 확장 파일을 StarUML™ 모듈 디렉토리(<install-dir>\modules) 하부 디렉토리로 옮겨야 한다. StarUML™은 프로그램 초기화 시에 모듈 디렉토리 하부를 검색하여 모든 옵션 스키마 파일들을 읽어 들인 후 프로그램에 자동 등록한다. 만약 옵션 스키마 파일이 문법에 맞지 않게 잘 못 작성되어 있거나, 확장자가 .opt가 아닌 경우에는 옵션 스키마 파일을 정상적으로 읽어 들이지 않고 무시하게 될 것이다.

옵션 스키마 파일은 가급적이면 Add-In 실행모듈이 포함되어 있는 디렉토리와 동일한 디렉토리에 설치하는 것을 권고한다.

**노트:** 옵션 스키마를 더 이상 사용하지 않으려면 StarUML™ 모듈 디렉토리(<install-dir>\modules) 하부에서 삭제하면 된다.

## 옵션값 접근하기

### COM 인터페이스를 사용하여 옵션값 접근하기

StarUML™의 COM 인터페이스를 이용하면 사용자가 옵션다이얼로그를 통해 변경한 옵션 항목의 값을 참조 할 수 있다. **IStarUMLApplication**의 **GetOptionValue()**는 입력 받은 SchemaID와 옵션의 Key로부터 옵션의 값을 Variant로 리턴해준다. **GetOptionValue()**의 호출 형식은

다음과 같다.

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

- **SchemaID** : 옵션 스키마 정의 파일에 명시되어 있는 스키마 아이디
- **Key** : 옵션 스키마 파일에 기술된 옵션 항목의 키

**GetOptionValue()**의 Variant 타입의 리턴값은 각 옵션 항목의 타입에 따라서 캐스팅하여 사용하면 된다. 그러나, Jscript 나 VBscript처럼 타입 검사를 하지 않는 언어에서는 별도의 타입 캐스팅 절차 없이 바로 값을 읽어올 수 있다.

다음은 StarUML™의 기본 옵션 스키마에 정의되어 있는 "UNDO\_LEVEL" 옵션 항목의 값을 얻어와서 메세지 상자로 출력하는 JScript 예제이다.

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var undoLevel = app.GetOptionValue("ENVIRONMENT", "UNDO_LEVEL");

WScript.Echo("Max. number of undo actions : " + undoLevel);
```

## 옵션값의 이벤트 변경 처리하기

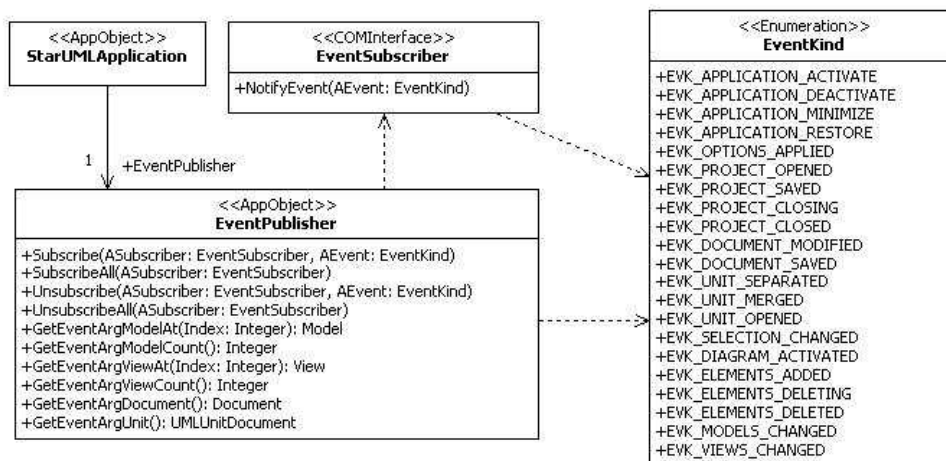
StarUML™은 **IEventSubscriber**를 구현한 Add-in들에게 프로그램의 조작 과정에서 발생하는 이벤트를 전파해준다. 사용자가 옵션 다이얼로그를 통해 옵션값을 변경하면 애플리케이션은 **IEventSubscriber**를 구현한 Add-in들의 이벤트 핸들러- **NotifyEvent()**-를 호출한다. 옵션값이 변경되었을 때 옵션 값을 즉시 Add-in에 반영하려면 **IEventSubscriber**를 구현하고 **NotifyEvent()** 메서드 내에서 **EVK\_OPTIONS\_APPLIED** 이벤트일때 **IStarUMLApplication.GetOptionValue()** 메서드를 사용하여 직접 옵션값을 읽어 오면 된다. VBScript나 JScript 등 스크립트를 사용하는 Add-in들은 **IEventSubscriber**를 구현할 수 없으므로 사용자에 의한 옵션값 변경 시에 이를 Add-in에 바로 반영할 수 없다.

이벤트 핸들링에 관한 더 자세한 내용은 다음 섹션에서 다루어 질 것이다.

## 이벤트 수신의 기본 개념

**IEventSubscriber** 인터페이스를 구현하는 Add-In 개체는 StarUML™ 애플리케이션의 다양한 내부 이벤트(Event)들을 전파 받을 수 있다. StarUML™ 애플리케이션은 내부 이벤트가 발생할 때 마다 등록된 **IEventSubscriber** 타입 개체들의 **NotifyEvent** 메소드를 호출한다.

아래의 클래스 다이어그램은 이벤트 수신과 관련된 외부 API 인터페이스들의 구조를 보여주고 있다.



## 이벤트의 종류

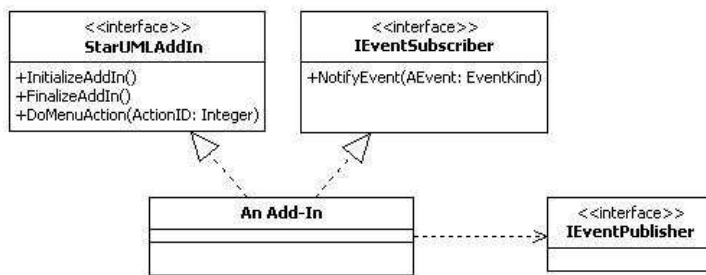
위의 그림에서 **EventKind** 열거체(Enumeration)는 **IEventSubscriber** 인터페이스를 구현하는 Add-In 개체가 수신할 수 있는 StarUML™ 애플리케이션 내부 이벤트의 종류를 정의하고 있다. **EventKind** 열거체의 각 리터럴의 의미는 다음 표와 같다.

이벤트 종류 (리터럴)	정수 값	이벤트 설명
EVK_APPLICATION_ACTIVATE	0	StarUML™ 애플리케이션의 창(Window)이 활성화(activate)될 때 발생한다.
EVK_APPLICATION_DEACTIVATE	1	StarUML™ 애플리케이션의 창이 비활성화(deactivate)될 때 발생한다.
EVK_APPLICATION_MINIMIZE	2	StarUML™ 애플리케이션의 창이 최소화될 때 발생한다.
EVK_APPLICATION_RESTORE	3	최소화된 StarUML™ 애플리케이션의 창이 이전 크기로 복구될 때 발생한다.
EVK_OPTIONS_APPLIED	4	옵션 값이 변경되었을 때 발생한다.
EVK_PROJECT_OPENED	5	프로젝트 요소가 생성되었거나 프로젝트 파일을 열었을 때 발생한다.
EVK_PROJECT_SAVED	6	프로젝트가 저장되었을 때 마다 발생한다.

EVK_PROJECT_CLOSING	7	프로젝트 닫기를 선택했을 때 발생한다.
EVK_PROJECT_CLOSED	8	프로젝트가 닫혔을 때 발생한다.
EVK_DOCUMENT_MODIFIED	9	도큐먼트(프로젝트, 유닛)가 변경되었을 때 발생한다.
EVK_DOCUMENT_SAVED	10	도큐먼트(프로젝트, 유닛)가 저장되었을 때 발생한다.
EVK_UNIT_SEPARATED	11	유닛(Unit) 요소로 분리되었을 때 발생한다.
EVK_UNIT_MERGED	12	분리된 유닛 요소가 병합되었을 때 발생한다.
EVK_UNIT_OPENED	13	유닛이 불러오기 되었을 때 발생한다.
EVK_SELECTION_CHANGED	14	모델링 요소의 선택이 변경될 때 발생한다.
EVK_DIAGRAM_ACTIVATED	15	다이아그램이 열릴 때 발생한다.
EVK_ELEMENTS_ADDED	16	새로운 모델링 요소가 생성되었을 때 마다 발생한다.
EVK_ELEMENTS_DELETING	17	모델링 요소를 삭제할 때 발생한다.
EVK_ELEMENTS_DELETED	18	모델링 요소가 삭제되었을 때 발생한다.
EVK_MODELS_CHANGED	19	모델 요소의 속성 값이 변경되었을 때 발생한다.
EVK_VIEWS_CHANGED	20	뷰 요소의 속성 값이 변경되었을 때 발생한다.

## 이벤트 수신하기

StarUML™ 애플리케이션의 이벤트를 수신하려는 Add-In 개체는 모든 StarUML™ Add-In들의 공통사항인 **IStarUMLAddIn** 인터페이스 외에 추가로 **IEventSubscriber** 인터페이스를 구현해야 한다.



다음 예제는 **IStarUMLAddIn** 인터페이스와 **IEventSubscriber** 인터페이스를 구현하는 StarUML™ Add-In 개체의 클래스 정의를 보여준다. 예제는 델파이 파스칼(Delphi Pascal) 코드로 쓰여졌다.

```

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;
  
```

## 이벤트 수신 등록 및 해지

**IEventSubscriber** 인터페이스를 구현한 Add-In 개체가 실제로 이벤트를 수신하기 위해서는 이벤트 수신을 등록하는 과정이 필요하다. 이벤트 수신 등록과 해지는 **IEventPublisher** 타입 개체를 통해 할 수 있는데, **IEventPublisher** 타입 개체는 **IStarUMLApplication** 요소를 통해 얻을 수 있다. 아래의 예제는 델파이 파스칼 코드에서 **IStarUMLApplication**과 **IEventPublisher** 타입 개체의 참조를 얻는 것을 보여주고 있다.

```

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;

```

그리고 **IEventPublisher** 인터페이스는 이벤트 수신 등록과 해지를 지원하는 다음과 같은 메소드들을 제공한다. 각 메소드에서 "ASubscriber" 인자는 **IEventSubscriber** 인터페이스를 구현한 Add-In 개체 자신이다.

메소드	설 명
Subscribe(ASubscriber: IEventSubscriber; AEvent: EventKind)	AEvent 인자가 지정하는 이벤트에 대한 수신을 등록한다.
SubscribeAll(ASubscriber: IEventSubscriber)	모든 이벤트에 대한 수신을 등록한다.
Unsubscribe(ASubscriber: IEventSubscriber; AEvent: EventKind)	AEvent 인자가 지정하는 이벤트에 대한 수신을 해지한다.
UnsubscribeAll(ASubscriber: IEventSubscriber)	모든 이벤트에 대한 수신을 해지한다.

Add-In 개체가 특정 이벤트 만을 수신하고자 할 때는 **Subscribe** 메소드를 사용한다. 예를 들어 두 가지의 특정 이벤트에 대해서만 수신을 하고자 하는 경우, 각각의 이벤트에 대해 **Subscribe** 메소드를 호출해야 한다. 모든 이벤트들을 모두 수신 하기를 원한다면 **SubscribeAll** 메소드를 사용하면 된다. 일반적으로 **Subscribe**, **SubscribeAll** 메소드는 **IStarUMLAddIn .InitializeAddIn** 메소드 구현에서 호출한다.

Add-In 개체는 수신 등록한 이벤트를 더 이상 수신할 필요가 없을 때(개체가 소멸될 때 등) 자신이 등록한 모든 이벤트들에 대해 등록을 해지해야 한다. **Subscribe** 메소드를 사용하여 수신 등록을 한 경우에는 **Unsubscribe** 메소드를 사용하고, **SubscribeAll** 메소드를 사용한 경우에는 **UnsubscribeAll** 메소드를 사용하여 등록을 해지한다. 일반적으로 **Unsubscribe**, **SubscribeAll** 메소드는 **IStarUMLAddIn.FinalizeAddIn** 메소드 구현에서 호출한다.

다음은 **EVK\_ELEMENTS\_ADDED**, **EVK\_ELEMENTS\_DELETED** 두 이벤트에 대한 수신 등록과 해지를 하는 예이다.

```

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;

```

## 이벤트 인자 얻어오기

각 이벤트가 발생했을 때 해당 이벤트와 관련된 아규먼트(**Argument**)를 얻어야 할 필요성이 있다. 예를 들어 모델링 요소의 생성과 관련된 (**EVK\_ELEMENTS\_ADDED**) 이벤트가 발생했을 때 어떤 모델링 요소가 생성되었는지를 알아야 하는 경우 등이다. **IEventPublisher** 인터페이스는 이벤트 아규먼트와 관련하여 다음과 같은 메소드들을 제공하고 있다.

메소드	설 명
GetEventArgModelCount (): Integer	이벤트와 관련된 모델 요소들의 개수를 반환한다.
GetEventArgModelAt(Index: Integer): IModel	이벤트와 관련된 모델 요소들 중에서 Index 번째 요소에 대한 참조를 반환한다.
GetEventArgViewCount: Integer	이벤트와 관련된 뷰 요소들의 개수를 반환한다.
GetEventArgViewAt(Index: Integer): IView	이벤트와 관련된 뷰 요소들 중에서 Index 번째 요소에 대한 참조를 반환한다.
GetEventArgDocument: IDocument	이벤트와 관련된 도큐먼트(Document) 요소에 대한 참조를 반환한다.
GetEventArgUnit: IUMLUnitDocument	이벤트와 관련된 유닛(Unit) 요소에 대한 참조를 반환한다.

## 이벤트 처리하기

Add-In에서 이벤트 수신을 등록했다면 등록된 이벤트가 발생했을 때 필요한 처리를 해야 할 것이다. StarUML™ 애플리케이션은 등록된 이벤트가 발생할 때 마다 해당 Add-In의 **NotifyEvent** 메소드를 호출하며, 발생할 이벤트의 종류를 인자로 전달한다. **NotifyEvent** 메소드의 인자로 이벤트의 종류를 전달하는 이유는 Add-In이 하나 이상의 이벤트에 대한 수신 등록을 했을 가능성이 있기 때문이다. 각 Add-In은 **NotifyEvent** 메소드에서 이벤트의 종류별로 필요한 처리를 하는 로직(logic)을 구현하면 된다.

다음은 **NotifyEvent** 메소드를 구현한 예제이다. 이 예제에서는 StarUML™ 애플리케이션에 연관(**UMLAssociation**) 요소나 일반화(**UMLGeneralization**) 요소가 생성되었을 때, 각 요소의 연결이 의미적으로 유효한지를 검사하고 있다. (이 예제는 위의 예제들과 연결되므로 Add-In 개체의 정의부분은 위의 예제를 참고하기 바란다.)

```

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;
    end;

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

[처음으로](#)

## 제 10 장. 노테이션 확장하기

이 장에서는 **Notation Extension**이 무엇이며 사용되는 기본 개념들은 어떤 것이 있는지 설명하고, **Notation Extension**을 위해서 사용되는 언어의 기본 구문에 대한 명세를 간략하게 소개한다. 그리고 예제를 통해서 새로운 종류의 다이어그램을 **Notation Extension**을 이용하여 어떻게 추가할 수 있는지 설명한다.

### 왜 노테이션 확장이 필요한가?

**Notation Extension**은 UML 모델에 대한 표기법을 사용자가 직접 정의하여 사용할 수 있도록 표기법을 위한 확장 개념이다. **StarUML**은 이러한 **Notation Extension**을 실행할 수 있는 플랫폼을 제공한다. 그러면 **UML**이 있는데 이러한 **Notation Extension** 개념이 왜 필요할까?

- 프로파일에 스테레오타입으로 **TextView**, **IconicView**, **DecorationView**를 제공하지만 이것만으로는 원하는 형태의 표기법을 구사할 수 없다.
- 데이터 모델링을 위해서 **ER** 다이어그램이 대부분 사용되는데, **UML**로 매핑하면 모델의 매핑은 자연스러울 수 있지만, 표기법의 매핑은 부자연스럽고 익숙하지 않다.
- UML** 메타모델은 모든 종류의 모델링 의미를 포함하기 위해서 충분한 크기의 데이터 컨테이너이다. 따라서 **UML** 틀에서 표기법을 확장할 수 있다면 모든 종류의 모델링틀을 위한 메타-모델링틀로서의 역할 수행이 가능하다.

따라서 표기는 기존 모델링 영역의 표기는 그대로 사용하면서도, 데이터로서의 모델 정보는 **UML** 모델로 표현하도록 함으로써, 다른 모델링 영역과 **UML** 모델링 영역간의 상호 보완, 효율성 및 호환성을 사용자에게 제공할 수 있다.

## 노테이션 확장 언어(Notation Extension Language)

### 구문의 기본 형태

**Notation Extension Language**의 구문은 **Scheme** 언어(**LISP**의 **Dialect**)와 유사한 형태의 구문을 가진다. 구문의 기본은 식이며, 전체가 하나의 식으로 구성된다. 식은 값이거나, 연산식으로 구성된다. 값은 실수, 정수, 스트링, 논리값, 널값, 식별자를 말하며, 연산식은 "("로 시작해서 ")"로 끝난다. 괄호 사이에는 연산자와 인자를 표현하는 또 다른 식이 순서대로 나타나는 형태를 취한다. 연산자나 식별자는 대소문자를 구분하지 않는다. 그리고 **C++**과 자바의 커멘트 방식을 따른다. 한 라인에 대한 커멘트는 **"/"**를 이용하고 여러 라인에 걸친 커멘트는 **"/\* \*/"**를 사용한다.

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

**Notation Extension Language**의 시작 연산식은 **notation**이라는 연산자로 시작된다. 그리고 인자식은 **onarrange**와 **ondraw**라는 연산자를 갖는 식으로 구성된다. **notation**은 스테레오타입 하나에 대응하는 표기법을 정의한다. 따라서 스테레오타입이 적용된 모델 요소가 화면에 보여질 때

notation식이 수행된다. 이때 **onarrange** 식이 우선 실행되어 화면에 표기법을 그리기 전에 배치에 관련된 식들을 수행한다. **ondraw** 식들은 화면에 직접 그리는 식을 수행한다.

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

**onarrange**와 **ondraw** 연산식이 인자 식으로 가질수 있는 식은 크게 다음과 같이 6가지 종류이다.

- sequence
- if
- for
- set
- arithmetic, logical, comparison operator
- built-in function

## sequence expression

**sequence** 식은 **java**의 **block**처럼 인자로 오는 식들을 순서대로 실행한다. **sequence** 식은 인자로 개수의 제한없이 여러개의 연산식을 가지는 형태로 구성된다.

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

다음 예제는 3개의 연산식을 하나의 **sequence** 식으로 묶는 것을 보여준다.

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

## if expression

**if** 식은 조건을 처리하기 위한 구문을 식으로 표현하고 있다. 첫번째 인자식은 조건식이고, 두번째 인자식은 조건식이 참일 때 수행되는 식이고, 세번째 인자식은 조건식이 거짓일 때 수행되는 식이다. 세번째 인자식은 생략 가능하다. 조건식이 거짓이고 세번째 식이 생략된 경우에 **if** 식내에서 아무런 수행도 하지 않고 식을 빠져 나온다.

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

다음 예제는 변수 **i**의 값이 0에서 30 사이의 값이라면 **i**의 값을 증가하고 그렇지 않으면 **i**의 값을 감소하는 예제이다.

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

## for expression

**for** 식은 특정 변수값을 초기값에서 종료값까지 범위까지 증가하면서 반복적으로 식을 수행한다. 첫번째 인자는 반복에 사용될 변수의 이름이다. 두번째 인자식은 변수의 초기값이고 세번째 인자식은 변수의 종료값이다. 마지막 인자식은 반복을 하면서 수행할 식을 의미한다.

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

다음은 1에서 10까지값을 화면에 출력하는 예제이다.

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

## set expression

**set expression**은 변수의 값을 할당하는 식이다. 변수의 선언은 필요 없으며 사용하는 순간 선언되며, 한번 선언되면 전역 변수로 사용된다.

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



다음은 **c**이라는 이름의 변수에 **a**와 **b** 스트링 변수의 값을 **concatenate**하는 예제이다.

```
(set a 'My name is ')      // a = "My name is ";
(set b 'foo')              // b = "foo";
(set c (concat a b))       // c = a + b;
```

## arithmetic, logical, comparison operator

제공되는 수학 연산자는 **+**, **-**, **\***, **/** 이며, 논리 연산자는 **and**, **or**, **not** 3가지 연산자가 제공되며, 비교를 위해서 **=**, **!=**, **<**, **<=**, **>**, **>=** 연산자가 제공된다. 다음은 이러한 연산자의 다양한 사용예제이다.

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

## built-in function

Notation Extension Language 내에서 정의된 **Built-in** 함수는 다음과 같이 분류 된다.

- 수학 함수
- 문자열 처리 함수
- 리스트 처리 함수
- 모델 접근 함수
- 그래픽 관련 함수

### 수학 함수

수학 **Built-in** 함수의 목록은 다음과 같다.

함수 시그니처	설 명
(sin angle)	angle에 대한 sin 값을 반환한다.
(cos angle)	angle에 대한 cos 값을 반환한다.
(tan angle)	angle에 대한 tan 값을 반환한다.
(trunc val)	val에 대해서 소수점 버림 값을 반환한다.
(round val)	val에 대해서 반올림값을 반환한다.

### 문자열 처리 함수

문자열 처리 **Built-in** 함수의 목록은 다음과 같다.

함수 시그니처	설 명
(concat str1 str2...)	인자 스트링들을 붙여서 연결된 하나의 스트링을 반환한다.
(trim str)	str 인자 스트링의 앞뒤에 존재하는 공백 문자가 제거된 스트링을 반환한다.
(length str)	str 인자 스트링의 길이를 반환한다.
(tokenize str deli)	str 인자 스트링으로, deli 인자를 구분자로, 나누어진 스트링의 리스트를 반환한다.

### 리스트 처리 함수

리스트 처리 **Built-in** 함수의 목록은 다음과 같다.

함수 시그니처	설 명
(list val1 val2 ...)	인자들로 구성된 리스트를 반환한다.
(append lst lst)	두 개의 리스트를 합한 리스트를 반환한다.
(append lst item)	
(itemat lst index)	index 위치에 존재하는 리스트를 반환한다.
(itemcount lst)	lst 리스트안에 존재하는 아이템 개수를 반환한다.

### 모델 접근 함수

모델 접근 **Built-in** 함수의 목록은 다음과 같다.

함수 시그니처	설 명
(mofattr elem attr)	elem 인자(IElement)의 attr 인자 값을 이름으로 하는 속성의 값을 반환한다.

(mofsetattr elem attr val)	elem 인자(IElement)의 attr 인자 값을 이름으로 하는 속성을 val값을 설정한다.
(mofref elem ref)	elem 인자(IElement)의 ref 인자 값을 이름으로 하는 reference를 반환한다.
(mofcolat elem col at)	elem 인자의 col 이름으로 하는 모델링 요소 참조의 collection에서 at 위치에 있는 객체 참조를 반환한다.
(mofcolcount elem col)	elem 인자(IElement)의 col 인자 값을 이름으로 하는 collection에 포함된 요소의 개수를 반환한다
(constraintval elem name)	elem 인자(IElement)의 name 인자 값을 이름으로 하는 constraint의 값을 반환한다.
(tagval elem tagset name)	elem 인자(IElement)의 tagset 인자 값의 TagDefinitionSet의 name 인자 값을 이름으로 하는 primitive 타입의 태그값을 반환한다.
(tagref elem tagset name)	elem 인자(IElement)의 tagset 인자 값의 TagDefinitionSet의 name 인자 값을 이름으로 하는 reference 타입의 태그값을 반환한다.
(tagcolat elem tagset name at)	elem 인자(IElement)의 tagset 인자 값의 TagDefinitionSet의 name 인자 값을 이름으로 하는 reference의 collection 타입일 경우의 at 위치의 태그값을 반환한다.
(tagcolcount elem tagset name)	elem 인자(IElement)의 tagset 인자 값의 TagDefinitionSet의 name 인자 값을 이름으로 하는 reference의 collection 타입일 경우의 collection에 포함된 요소의 개수를 반환한다.

## 그래픽 관련 함수

스타일 관련 Built-in 함수의 목록은 다음과 같다.

함수 시그니처	설 명																																																				
(setpencolor color)	Pen의 색깔을 인자 color 값으로 변경한다. 색깔은 다음 값 중 하나를 가질수 있다.																																																				
	<table><tr><th>Value</th><th>Meaning</th><th>Value</th><th>Meaning</th></tr><tr><td>clNone</td><td>White</td><td>clAqua</td><td>Aqua</td></tr><tr><td>clBlack</td><td>Black</td><td>clBlue</td><td>Blue</td></tr><tr><td>clCream</td><td>Cream</td><td>clDkGray</td><td>Dark Gray</td></tr><tr><td>clFuchsia</td><td>Fuchsia</td><td>clGray</td><td>Gray</td></tr><tr><td>clGreen</td><td>Green</td><td>clLime</td><td>Lime green</td></tr><tr><td>clLtGray</td><td>Light Gray</td><td>clMaroon</td><td>Maroon</td></tr><tr><td>clMedGray</td><td>Medium Gray</td><td>clMoneyGreen</td><td>Mint green</td></tr><tr><td>clNavy</td><td>Navy blue</td><td>clOlive</td><td>Olive green</td></tr><tr><td>clPurple</td><td>Purple</td><td>clRed</td><td>Red clGrayText</td></tr><tr><td>clSilver</td><td>Silver</td><td>clSkyBlue</td><td>Sky blue</td></tr><tr><td>clTeal</td><td>Teal</td><td>clWhite</td><td>White</td></tr><tr><td>clYellow</td><td>Yellow</td><td></td><td></td></tr></table>	Value	Meaning	Value	Meaning	clNone	White	clAqua	Aqua	clBlack	Black	clBlue	Blue	clCream	Cream	clDkGray	Dark Gray	clFuchsia	Fuchsia	clGray	Gray	clGreen	Green	clLime	Lime green	clLtGray	Light Gray	clMaroon	Maroon	clMedGray	Medium Gray	clMoneyGreen	Mint green	clNavy	Navy blue	clOlive	Olive green	clPurple	Purple	clRed	Red clGrayText	clSilver	Silver	clSkyBlue	Sky blue	clTeal	Teal	clWhite	White	clYellow	Yellow		
	Value	Meaning	Value	Meaning																																																	
	clNone	White	clAqua	Aqua																																																	
	clBlack	Black	clBlue	Blue																																																	
	clCream	Cream	clDkGray	Dark Gray																																																	
	clFuchsia	Fuchsia	clGray	Gray																																																	
	clGreen	Green	clLime	Lime green																																																	
	clLtGray	Light Gray	clMaroon	Maroon																																																	
	clMedGray	Medium Gray	clMoneyGreen	Mint green																																																	
	clNavy	Navy blue	clOlive	Olive green																																																	
	clPurple	Purple	clRed	Red clGrayText																																																	
	clSilver	Silver	clSkyBlue	Sky blue																																																	
	clTeal	Teal	clWhite	White																																																	
clYellow	Yellow																																																				
(setpenstyle style)	Pen의 style을 변경한다. psSolid, psDash, psDot, psDashDot, psDashDotDot, psClear, psInsideFrame중의 하나의 값이 가능하다.																																																				
	<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>psSolid</td><td>A solid line.</td></tr><tr><td>psDash</td><td>A line made up of a series of dashes.</td></tr><tr><td>psDot</td><td>A line made up of a series of dots.</td></tr><tr><td>psDashDot</td><td>A line made up of alternating dashes and dots.</td></tr><tr><td>psDashDotDot</td><td>A line made up of a series of dash-dot-dot combinations.</td></tr><tr><td>psClear</td><td>No line is drawn (used to omit the line around shapes that draw an outline using the current pen).</td></tr><tr><td>psInsideFrame</td><td>A solid line, but one that may use a dithered color if Width is greater than 1.</td></tr></table>	Value	Meaning	psSolid	A solid line.	psDash	A line made up of a series of dashes.	psDot	A line made up of a series of dots.	psDashDot	A line made up of alternating dashes and dots.	psDashDotDot	A line made up of a series of dash-dot-dot combinations.	psClear	No line is drawn (used to omit the line around shapes that draw an outline using the current pen).	psInsideFrame	A solid line, but one that may use a dithered color if Width is greater than 1.																																				
	Value	Meaning																																																			
	psSolid	A solid line.																																																			
	psDash	A line made up of a series of dashes.																																																			
	psDot	A line made up of a series of dots.																																																			
	psDashDot	A line made up of alternating dashes and dots.																																																			
	psDashDotDot	A line made up of a series of dash-dot-dot combinations.																																																			
	psClear	No line is drawn (used to omit the line around shapes that draw an outline using the current pen).																																																			
psInsideFrame	A solid line, but one that may use a dithered color if Width is greater than 1.																																																				
(setbrushcolor color)	Brush의 색깔을 인자 color 값으로 변경한다.																																																				
(setbrushstyle style)	Brush의 style을 변경한다. 다음 중 하나의 값이 가능하다.  bsSolid, bsClear, bsHorizontal, bsVertical, bsFDiagonal, bsBDiagonal, bsCross, bsDiagCross																																																				
(setFontface font)	인자의 폰트 이름으로 폰트를 변경한다.																																																				
(setFontcolor color)	폰트의 색깔을 인자 color 값으로 변경한다.																																																				
(setfontsize size)	폰트의 크기를 인자 값으로 변경한다.																																																				
	폰트의 스타일을 인자의 값으로 변경한다. fsBold, fsItalic, fsUnderline, fsStrikeOut 값의 조합으로 구성되며, 두개 이상의 값을 지정할 경우에는   문자를 사용해서 값을 조합한다.																																																				

(setFontstyle style)	<b>Value</b>	<b>Meaning</b>
	fsBold	The font is boldfaced.
	fsItalic	he font is italicized.
	fsUnderline	The font is underlined.
	fsStrikeOut	The font is displayed with a horizontal line through it.
(setDefaultstyle)	사용자의 조작에 의해서 변경된 Pen, Brush, Font 정보를 원래 정보로 복구한다.	

텍스트 출력 관련 Built-in 함수의 목록은 다음과 같다.

함수 시그니처	설 명
(textheight str)	인자 스트링의 height 값을 반환한다.
(textwidth str)	인자 스트링의 width 값을 반환한다.
(textout x y str)	x, y 좌표에 str 스트링을 출력한다.
(textbound x1 y1 x2 y2 yspace text clipping)	(x1, y1)점과 (x2, y2)점의 영역에 text를 출력한다. yspace는 문자의 행간 공백이다. clipping값이 참일 경우에는 영역에 포함되는 문자만 출력한다.
(textrect x1 y1 x2 y2 x y str)	(x1, y1)점과 (x2, y2)점의 영역에서 x, y 위치에 str 스트링을 출력한다.

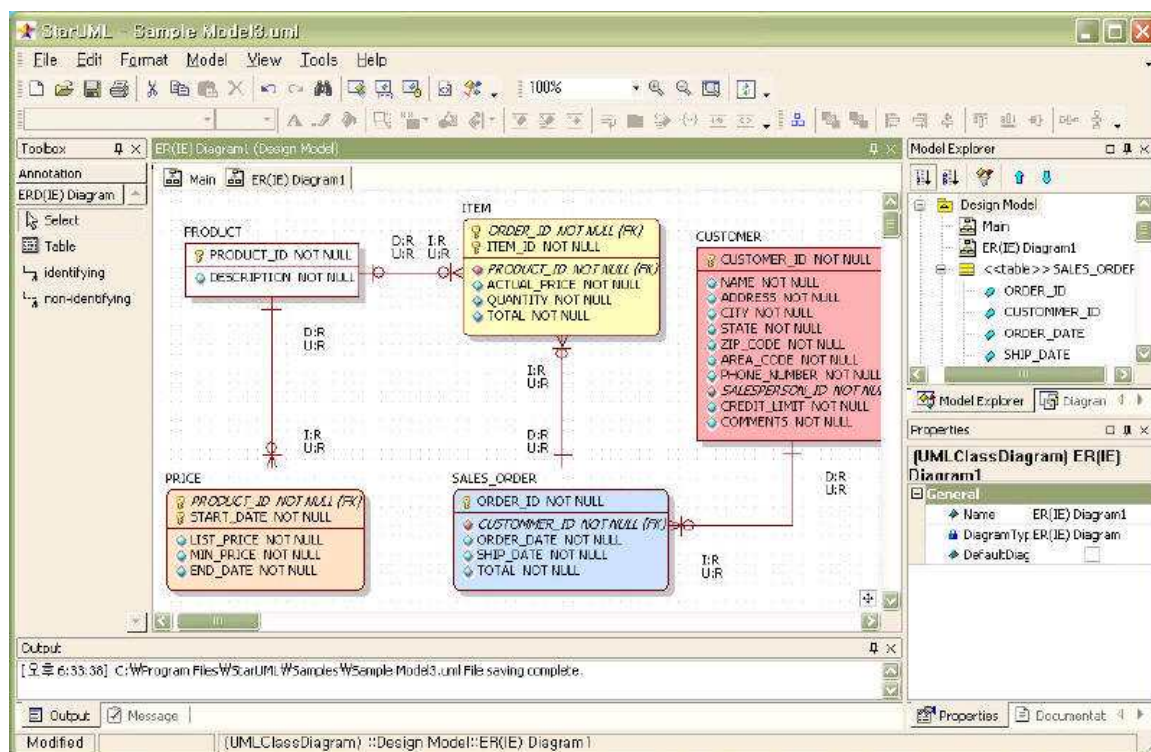
도형 관련 Built-in 함수의 목록은 다음과 같다.

함수 시그니처	설 명
(rect x1 y1 x2 y2)	(x1, y1)점과 (x2, y2)점을 영역의 끝점으로 하는 사각형을 그린다.
(filerect x1 y1 x2 y2)	(x1, y1)점과 (x2, y2)점을 영역의 끝점으로 하는 채움 사각형을 그린다.
(ellipse x1 y1 x2 y2)	(x1, y1)점과 (x2, y2)점을 영역의 끝점으로 하는 타원을 그린다.
(roundrect x1 y1 x2 y2 x3 y3)	(x1, y1)점과 (x2, y2)점을 영역의 끝점으로 하는 끝이 둥근 사각형을 그린다. x3, y3 값은 둥근 모서리 사각형의 폭과 높이를 의미한다.
(arc x1 y1 x2 y2 x3 y3 x4 y4)	draws an arc inside an ellipse bounded by the rectangle defined by (X1,Y1) and (X2,Y2). The arc starts at the intersection of the line drawn between the ellipse center ((X1+X2) / 2.0, (Y1+Y2) / 2.0) and the point (X3,Y3) and is drawn counterclockwise until it reaches the intersection of the line drawn between the ellipse center and the point (X4,Y4)
(pie x1 y1 x2 y2 x3 y3 x4 y4)	draws a pie-shaped wedge on the image. The wedge is defined by the ellipse bounded by the rectangle determined by the points (X1, Y1) and X2, Y2). The section drawn is determined by two lines radiating from the center of the ellipse through the points (X3, Y3) and (X4, Y4)
(drawbitmap x y img transparent)	x, y 위치에 img 파일명의 이미지를 출력한다. transparent 속성값이 true이면 배경색을 투명하게 출력한다.
(drawbitmap x1 y1 x2 y2 img transparent)	transparent 속성은 생략될 수 있으며, 생략될 경우에는 false로 설정된다. x1, y1, x2, y2 범위에 img 파일명의 이미지를 늘려서 출력한다.
(moveto x y)	선을 그리는 펜을 x y 위치로 이동한다.
(lineto x y)	마지막으로 펜이 이동된 위치에서 x y 위치를 연결하는 선을 그린다.
(line x1 y1 x2 y2)	(x1, y1)점과 (x2, y2)점을 연결하는 선을 그린다.
(pt x y)	인자를 x, y 좌표로 하는 포인트 객체를 반환한다.
(polygon (pt x1 y1) (pt x2 y2) ...)	인자 포인트 객체를 연결하는 polygon을 그린다.
(polyline (pt x1 y1) (pt x2 y2) ...)	인자 포인트 객체를 연결하는 polyline을 그린다.
(polybezier (pt x1 y1) (pt x2 y2) ...)	인자 포인트 객체를 연결하는 bezier 곡선을 그린다.
(ptatx index)	대상 모델의 뷰가 Edge일 경우에 대해서만 사용 가능하다. Edge의 점들중에 index 위치의 x 좌표를 반환한다.
(ptaty index)	대상 모델의 뷰가 Edge일 경우에 대해서만 사용 가능하다. Edge의 점들중에 index 위치의 y 좌표를 반환한다.
(ptcount)	대상 모델의 뷰가 Edge일 경우에 대해서만 사용 가능하다. Edge의 점 개수를 반환한다.
	It is available when current view element is edge element. it draws end of edge in argument style. Style is composed of the followings and seperator is " " character. 대상 모델의 뷰가 Edge일 경우에 대해서만 사용 가능하다. Edge의 끝 모양을 endStyle 형태로 그린다. endStyle의 값은 다음의 조합으로 구성되며, 두개 이상의 값을 지정할 경우에는   문자를 사용한다.

(drawedge headOrTail endStyle)	<b>Value</b>	<b>Shape</b>	<b>Value</b>	<b>Shape</b>
	esStickArrow	←	esSolidArrow	→
	esTriangle	▷	esDiamond	◇
	esMiniDiamond	◊	esArrowDiamond	↗
	esCrowFoot	↗	esHalfStickArrow	↖
	esBar	+	esDoubleBar	≡
	esBelowCircle	⊖	esCircle	○
	esRect	□	esFilledTriangle	◼
	esFilledDiamond	◆	esMiniFilledDiamond	◈
	esArrowFilledDiamond	◆	esFilledHalfStickArrow	↖
	esFilledCircle	●	esFilledRect	■
	esMiniHalfDiamond	◈		
(drawobject elem)	인자 elem(IView)를 StarUML에서 정의된 방식으로 화면에 출력한다.			
(arrangeobject elem)	인자 elem(IView)를 StarUML에서 정의된 방식으로 재배열한다.			

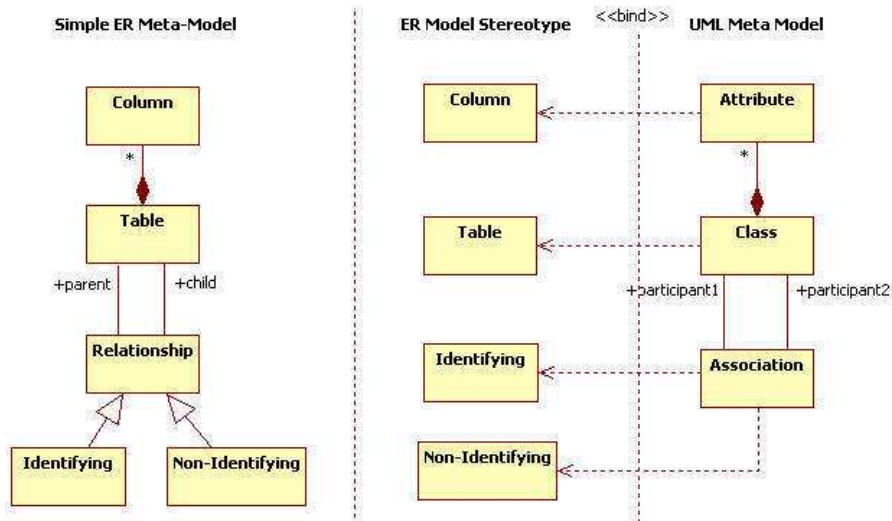
## 새로운 다이어그램 타입을 만들기

Notation Extension을 사용하기 위해서 준비해야 할 사항이 몇가지 있다. 노테이션이 적용될 요소를 기술하는 프로파일이 준비되어야 한다. 그리고 어떻게 표기법이 그려질지 정의하는 **Notation Extension** 파일(.NXT 확장자)이 필요하다. 프로파일에서 어떤 스테레오 타입과 어떤 **Notation Extension** 파일이 연결되어질지 정보를 기술해야 한다. 또한 **Notation Extension**에서 사용할 속성을 해당 스테레오 타입의 태그값에 추가한다. 그러면 이러한 과정을 ER-Diagram에 대한 표기법 확장을 통해서 간단하게 알아보자.



## 프로파일의 정의

먼저 ER-Diagram에서 사용되는 요소를 살펴보면 Table, Column, Relationship등이 있다.



이들 요소에 대해서 각각 스테레오타입을 만들고 UML 모델의 Class, Attribute, Association으로 각각 매핑하여 스테레오타입 적용시 동일한 의미를 지닐 수 있게 한다. 이러한 관계를 프로파일로 옮기면 다음과 같다. 프로파일의 <STEREOTYPELIST>에 **table**이라는 스테레오타입을 새로 추가하고, 스테레오타입이 적용될 모델이 UMLClass 타입이라는 것을 <BASECLASS>에 기술한다. 그리고 **table**이 ER 형식의 표기법으로 보여지도록 Notation Extension 파일명을 <NOTATION>에 기술한다.

**column** 스테레오타입의 경우는 PK, FK, AK, IK등인지 구분되어야 하므로 이들 정보를 저장하기 위해서 별도의 태그값이 필요하다. 따라서 이러한 태그값을 정의하고 있는 태그셋 참조명을 <RELATEDTAGDEFINITIONSET>에 기술한다.

```
<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>
  </BODY>
</PROFILE>
```

태그셋은 <TAGDEFINITIONSETLIST>의 <TAGDEFINITIONSET>에 기술되며, <TAGDEFINITION>에서 추가된 태그값의 이름, 타입, 디폴트 값을 기술한다. 현재는 PK와 FK인지를 판별하기 위한 태그값이 추가되어 있고, 태그값의 타입은 **Boolean**이다. 그리고 기본값은 **false**를 가지므로써, 모든 **column**은 기본적으로 어떠한 키도 부여되지 않고 생성된다.

```
...
</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>

```

이렇게 정의된 스테레오타입을 보여질 다이어그램을 형태를 선택해야 한다. <DIAGRAMTYPELIST>에서 새로운 다이어그램 **ER Diagram**이라는 타입을 정의하고 **ClassDiagram**을 기반으로 한다고 명시한다. 그리고 **ER Diagram**을 작성할 때 보여지는 팔레트의 참조명을 <AVAILABLEPALLETTE>에 기술한다.

```

<DIAGRAMTYPELIST>
  <DIAGRAMTYPE>
    <NAME>ER(IE) Diagram</NAME>
    <DISPLAYNAME>ER(IE) Diagram</DISPLAYNAME>
    <BASEDIAGRAM>ClassDiagram</BASEDIAGRAM>
    <ICON>DataModelDiagram.bmp</ICON>
    <AVAILABLEPALLETTELIST>
      <AVAILABLEPALLETTE>ERD(IE)</AVAILABLEPALLETTE>
    </AVAILABLEPALLETTELIST>
  </DIAGRAMTYPE>
</DIAGRAMTYPELIST>

```

위에서 참조된 팔레트 목록의 정의는 <PALLETTE>의 <PALLETTEITEMLIST>에 기술된다. 그리고 팔레트의 각 버튼에 대한 세부 정보를 기술하는 부분에 대한 참조명을 <PALLETTEITEM>에 기술하고, <ELEMENTPROTOTYPE>에서 참조된 **PALLETTEITEM**에 대한 정보를 자세히 기술한다. <ELEMENTPROTOTYPE>은 팔레트 버튼의 명칭과 버튼의 아이콘, 생성될 요소의 타입 등에 대한 정보가 기술된다. 이 중에서 생성시에 요소가 **NOTATION EXTENSION**의 적용을 받도록 하기 위해서 <SHOWEXTENSION>의 값을 **true**로 설정해야 한다.

```

<PALLETTELIST>
  <PALLETTE>
    <NAME>ERD(IE)</NAME>
    <DISPLAYNAME>ERD(IE) Diagram</DISPLAYNAME>
    <PALLETTEITEMLIST>
      <PALLETTEITEM>Table</PALLETTEITEM>
      <PALLETTEITEM>identifying</PALLETTEITEM>
      <PALLETTEITEM>non-identifying</PALLETTEITEM>
    </PALLETTEITEMLIST>
  </PALLETTE>
</PALLETTELIST>

<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>
...

```

## Notation Extension 작성

프로파일 정의만으로도 **Data Modeling**은 가능하지만 **ER** 모델링 표기법으로 보여지기 위해서는 스테레오타입 정의 부분의 <NOTATION>에서 기술된 **Notation Extension** 파일(.nxt)을 작성해야 된다.

다음은 **table** 스테레오타입에 대한 표기법 출력하는 **table.nxt** 파일이다. **notation**식은 **onarrange**에서 **drawing**하기 전에 필요한 상태의 설정들을 수행한다. **ondraw**에서는 테이블의 이름 부분, **PK** 컬럼 부분, 그외의 컬럼 부분을 보여주는 부분으로 구성된다.

```

(notation
  (onarrange ...)

  (ondraw
    // draw name part ...

    // draw PK column part ...

    // draw other column part ...
  )
)

```

테이블 이름을 보여주는 부분에서는 테이블을 그리는데 필요한 변수값을 설정하고 모델로부터 **Name** 속성값을 가져와서 **x, y** 위치에 출력한다.

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

여기서 **left**, **top** 변수는 **Notation Extension**이 수행될 때마다 매번 **StarUML** 플랫폼으로부터 넘겨받고, **Notation Extension** 종료후에 다시 값을 **StarUML** 플랫폼에 설정한다. 이러한 변수에는 다음과 같다. 넘겨**View** 요소로부터 받아오는 변수이다. 이와 같이 이것 이외에도 **view**와 **model** 변수는 현재 선택된 **View**요소와 **Model**요소에 대한 객체이다.

변수명	적용 뷰 요소	플랫폼으로 반환 여부	설 명
view	Node,Edge	not return	현재 표기 대상이 되는 <b>View</b> 요소
model	Node,Edge	not return	현재 표기 대상이 되는 <b>Model</b> 요소
left	Node	return	현재 표기 대상이 되는 <b>Node View</b> 요소의 <b>Left</b> 값
top	Node	return	현재 표기 대상이 되는 <b>Node View</b> 요소의 <b>Top</b> 값
right	Node	return	현재 표기 대상이 되는 <b>Node View</b> 요소의 <b>Right</b> 값
bottom	Node	return	현재 표기 대상이 되는 <b>Node View</b> 요소의 <b>Bottom</b> 값
width	Node	return	현재 표기 대상이 되는 <b>Node View</b> 요소의 <b>Width</b> 값
height	Node	return	현재 표기 대상이 되는 <b>Node View</b> 요소의 <b>Height</b> 값
minwidth	Node	not return	현재 표기 대상이 되는 <b>Node View</b> 요소의 <b>MinWidth</b> 값
minheight	Node	not return	현재 표기 대상이 되는 <b>Node View</b> 요소의 <b>MinHeight</b> 값
points	Edge	not return	현재 표기 대상이 되는 <b>Edge View</b> 요소의 <b>point</b> 집합 객체
head	Edge	not return	현재 표기 대상이 되는 <b>Edge View</b> 요소의 <b>Head</b> 로 참조하는 <b>View</b> 요소
tail	Edge	not return	현재 표기 대상이 되는 <b>Edge View</b> 요소의 <b>Tail</b> 로 참조하는 <b>View</b> 요소

다음은 현재 테이블이 다른 테이블에 대해서 종속적인 관계를 가지고 있는지 검사한다. 이를 위해서 현재 테이블(클래스)의 연관을 반복하면서 연관의 머리쪽에 테이블이 연결되어 있다면 테이블이 종속적이라고 판단한다. 이러한 판단정보를 이용하여 종속적인 테이블은 둥근 모서리의 사각형으로 테두리 사각형을 그리고, 종속적이지 않은 테이블일 경우에는 일반 사각형으로 테두리를 그린다.

```
(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))
```

컬럼의 내용을 출력할 때는 테이블에 포함된 모든 컬럼(**attribute**)을 반복하면서 **PK** 태그값이 **true**인 요소들에 대해서 다른 컬럼보다 상위에 그려지고, **PK** 컬럼 표시의 이미지를 좌측에 표시하고 컬럼 이름을 화면에 출력한다.

```
...
(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)
```

다시 한번 컬럼을 반복하면서 **PK** 태그값이 **false**인 요소들만을 찾아서 컬럼 이미지와 컬럼의 이름을 화면에 출력한다.



```

...
(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)
        ... ))))

```

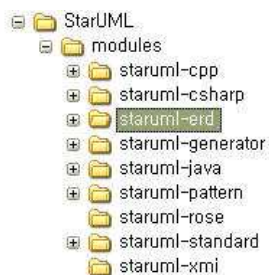
## Notation Extension의 설치와 사용

작성된 **Notation Extension** 파일을 프로파일에서 기술된 경로에 존재해야 한다. **table** 스테레오타입의 예에서는 별도의 경로가 명시되지 않고 파일명만 명시되었으므로, 프로파일과 **Noation Extension** 파일은 동일한 폴더에 위치시키도록 한다.



모든 작성이 완료되었으면 설치를 위해서는 다음의 과정을 수행한다.

1. 프로파일과 **Notation Extension** 파일, 관련 이미지 파일들을 특정이름의 모듈 폴더에 넣는다.
2. 해당 모듈 폴더를 **staruml/modules** 폴더 아래에 복사한다.
3. **StarUML**을 재시작하면 설치가 완료된다.

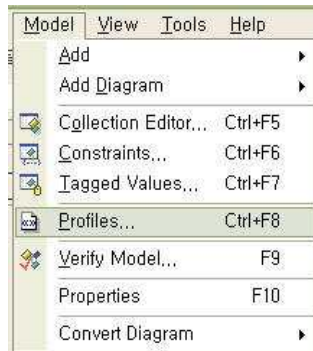


## 참고

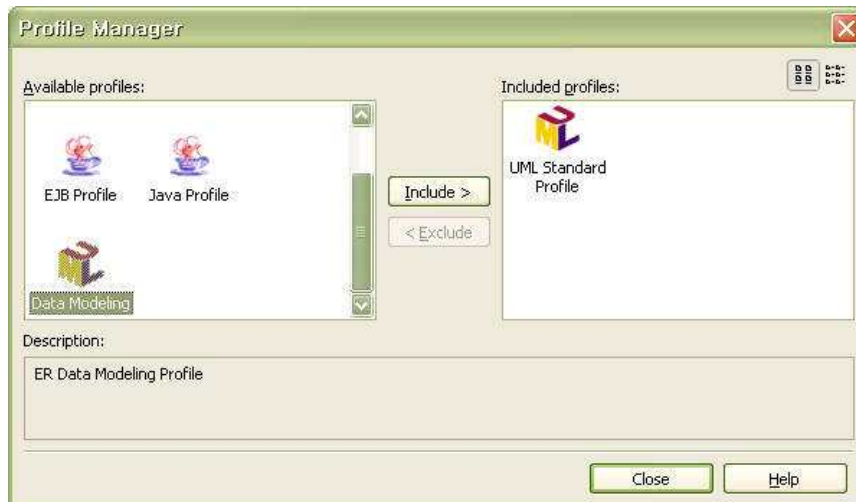
- ER-Diagram에 대한 완전한 **Notation Extension**과 프로파일을 포함하는 모듈은 **StarUML** 공식 홈페이지의 **module download**에서 받아서 설치하도록 한다.

설치된 프로파일과 **Notation Extension** 기능을 사용하는 방법은 다음과 같다.

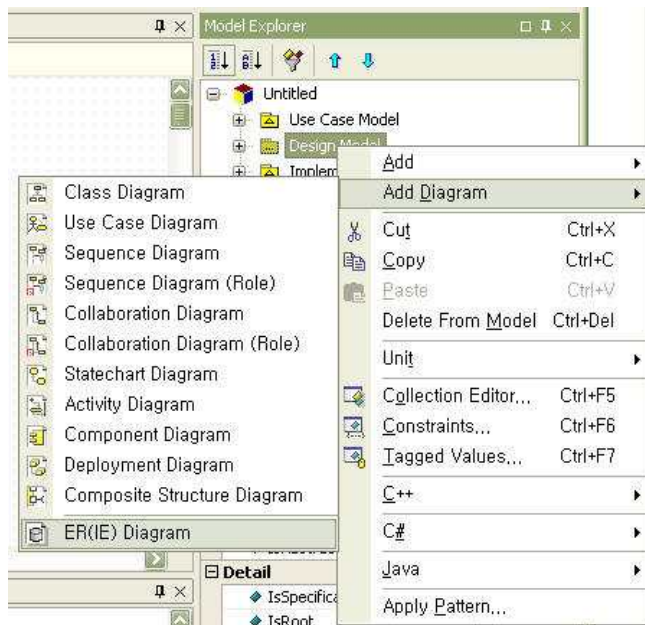
1. **StarUML**을 시작한다.
2. **[Model]** -> **[Profiles...]** 메뉴에서 클릭한다.



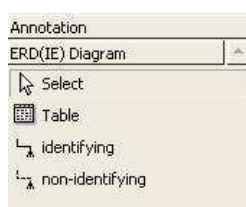
3. **[Profile Manager]** 다이얼로그가 나타나면 **[Available profiles]** 리스트창에서 Data Modeling 프로파일을 선택하고 **[Include]** 버튼을 클릭한다.



4. **[Model Explorer]**에서 ER-Diagram을 포함할 패키지를 선택하고, 팝업 메뉴 **[Add Diagram] -> [ER(IE) Diagram]** 메뉴를 클릭한다.



5. ER-Diagram이 **[Main]** 화면에 나타나고 ER 모델링용 팔레트가 화면에 나타난다.



6. 팔레트에서 노테이션을 이용하여 모델링을 하고, 속성 정보는  버튼을 클릭하여 **[Tagged Value Editor]**의 **[ERD]** 탭의 태그값들

UMLStandard ERD

Tag Definition Set:  
column

Tagged Values:

- column
  - Ref
  - ColumnName
  - PK ☒
  - FK ☐
  - AK ☐
  - IE ☐
  - MultiOption NOT NULL
    - NOT NULL
    - NULL
    - IDENTITY

Set As Default

Close Help

## 제 11 장. 템플릿 작성하기

## 템플릿 구성 요소

2010-10-13

- IF ~ ENDIF
- DISPLAY
- SCRIPT

## REPEAT 명령어

REPEAT 명령어는 인자를 만족하는 모델 요소를 반복해서 열거하는 명령어이다. 또한 REPEAT 명령어부터 ENDREPEAT 명령어까지의 영역에 존재하는 스타일을 요소의 개수만큼 반복하여 생성되는 문서에 출력한다. REPEAT 명령어는 다음과 4개의 인자를 가진다.

인 자	설 명	비 고
Pathname	반복할 요소가 존재하는 상위 경로	Optional
FilterType	현재 경로 아래에 존재하는 요소 중에 반복할 요소의 타입	Optional
CollectionName	현재 요소에서 반복할 요소를 포함하는 컬렉션 이름	Optional
Condition	반복할 요소가 만족해야 하는 조건식	Optional

첫번째 인자는 반복해서 가져올 UML 모델의 시작 기준점을 의미한다. 시작 기준점은 경로명의 형태를 취하고 있다. 경로명은 StarUML에서 해당 요소를 선택하면 하단 상태바에 "(요소타입) 경로명" 형태로 표시되므로 이 값을 참조한다. 경로는 절대경로와 상대경로형태로 표시될 수 있다. 절대경로는 경로명이 "::"로 시작한다. 예를 들어 경로명이 "::Use case model::PackageB"이라면 프로젝트말에 Use case model말에 PackageB 라는 요소를 지칭하고, "Use case model::PackageB"는 현재 경로 밑에 있는 Use case model 밑에 있는 PackageB라는 요소를 지칭한다. 또한 경로명의 앞에는 "{R}"이라는 문자가 부가적으로 첨가될 수 있다. "{R}"이 첨가된 경우에는 경로명 아래에 존재하는 모든 서브 경로에 존재하는 요소들을 반복해서 열거하고, "{R}"이 없을 경우에는 경로 바로 밑에 존재하는 요소에 대해서만 반복해서 열거한다. 첫번째 인자가 생략되면 현재 경로의 아래에 존재하는 요소를 반복한다.

두번째 인자는 첫번째 경로명 아래에 존재하는 요소중에 반복할 요소의 타입을 의미한다. 만약 "UMLClass"가 인자로 사용되면, 경로명 아래에 존재하는 모든 UMLClass 타입의 요소들을 반복하여 열거한다. 만약 인자가 생략되면 모든 타입의 요소들에 대해서 반복을 수행한다.

세번째 인자는 현재 선택된 요소 중에서 반복할 요소를 포함하는 컬렉션 이름이다. 첫번째 인자가 "::Use case Model::PackageB"이고, 두번째 인자가 "UMLClass"이고, 세번째 인자가 "OwnedElements"이면, "::Use case Model::PackageB" 경로 바로 아래에 존재하는 모든 "UMLClass" 타입 요소의 "OwnedElements" 컬렉션 내부의 요소를 반복한다. 만약 세번째 인자가 생략된다면 첫번째, 두번째 인자에 의해서만 반복을 수행한다.

네번째 인자는 반복하는 요소가 만족해야 할 조건식을 의미한다. 조건식은 JScript로 표현된다. 인자값이 current().StereotypeName == 'boundary'라면, 선택된 요소의 스테레오타입이 "boundary"인 요소에 대해서만 반복을 수행한다. 인자는 기본적으로 참의 값을 가진다. 따라서 인자값이 생략될 경우에는 선택된 모든 요소를 반복한다.

### 참고

- current()는 Generator에서 사용하는 Built-in function이다. 자세한 사항은 [템플릿 구성 요소 > Built-in Functions](#)를 참조한다.

## WORD 템플릿에서 명령어의 변형

- REPEAT 명령어에 대해서 ENDREPTR 명령어가 대응될 수 있다. REPEAT 명령어와 ENDREPTR 명령어는 특정 테이블의 행을 반복할 경우에 사용된다. 만약 클래스의 목록을 표로 작성할 경우에는 REPEAT 명령어는 반복할 행의 첫번째 셀에 위치시키고, ENDREPTR 명령어는 반드시 반복할 행의 맨 마지막 셀에 위치시킨다. 그러면 해당 요소를 열거하면 해당 요소만큼의 테이블 행이 생성된다.

## IF 명령어

IF 명령어는 인자의 조건을 만족할 경우에 대해서 ENDIF 명령어까지의 영역에 존재하는 스타일을 표시한다. IF 명령어는 다음의 인자를 가진다. 인자값은 JScript로 표현된다.

인 자	설 명	비 고
Condition	만족해야 하는 조건식	Mandatory

### 참고

- IF 명령어는 Excel과 PowerPoint 템플릿에서는 처리되지 않는다. (차후에 구현 예정)

## WORD 템플릿에서 명령어의 변형

- IF 명령어에 대해서도 비슷한 변형이 존재한다. 특정 테이블의 행이 조건이 만족할 경우에 보이고, 그렇지 않을 경우에는 보이지 않도록 하기 위해서 IF와 ENDIFTR 명령어가 존재한다. 인자 설정방식은 IF 명령어와 동일하며, 해당 행의 첫번째 셀에 IF 명령어가 위치해야 하고, 해당 행의 마지막 셀에 ENDIFTR 명령어가 위치해야 한다.

## DISPLAY 명령어

DISPLAY 명령어는 인자를 만족하는 모델 요소의 값 또는 그림을 생성되는 문서에 출력한다. DISPLAY 명령어는 다음과 같은 인자들을 가진다.

인 자	설 명	비 고
Pathname	선택할 요소의 경로	Optional

Expression	출력할 값에 대한 표현식	Optional
------------	---------------	----------

첫번째 인자는 출력할 값이 참조할 경로명이다. 경로명은 절대경로와 상대경로의 형태로 표현된다. 만약 생략된다면 현재 경로를 선택한다.

두번째 인자는 출력할 값의 표현식이다. 만약 첫번째 인자가 "::Use case Model::PackageB"이고 두번째 인자가 `current().Documentation`이라면, 생성되는 문서에 Use case Model 아래의 PackageB라는 이름의 요소의 Documentation 속성의 값을 출력한다.

## WORD 템플릿에서 명령어의 변형

- WORD 템플릿에서 **DISPLAY** 명령어 인자의 사용 방식이 약간 차이가 있다. 첫번째 인자에 의해서 선택된 경로의 요소가 **UMLDiagram** 타입이고 두번째 인자가 생략되면 생성되는 문서에 선택된 다이어그램의 이미지를 삽입한다. 그러나 첫번째 인자가 **UMLDiagram** 타입을 요소를 선택하고 있더라도, 두번째 인자의 표현식이 값을 가지고 있으면 그 값을 출력한다.
- WORD 템플릿에서는 **DISPLAY** 명령어가 세번째 인자를 가진다. 세번째 인자는 현재 출력된 값(문자열)을 색인으로 설정할지 여부를 나타낸다. WORD 문서 생성 후반부 시점에 색인 목록이 자동으로 생성되어질 수 있게 한다. 인자값을 "I"로 설정하면 색인으로 생성하고, 생략되면 색인으로 생성하지 않는다.

## POWERPOINT 템플릿에서 명령어의 변형

- POWERPOINT 템플릿에서 **DISPLAY** 명령어의 이름이 **DISPLAY-TEXT**와 **DISPLAY-IMAGE**로 구분하여 사용한다. **DISPLAY-TEXT**는 텍스트 출력을 위해서 명시적으로 사용한다. 인자는 **DISPLAY**와 동일하다. 그리고 다이어그램을 출력하기 위해서는 **DISPLAY-IMAGE** 명령어를 사용한다. 첫번째 인자에 의해서 **UMLDiagram** 타입 요소를 선택하고, 두번째 인자는 생략되어야 한다.

## SCRIPT 명령어

SCRIPT 명령어는 공통 명령어 이외에 명령어를 사용자가 수행하고자할때 사용하는 명령어이다. 인자부분을 JScript 형태의 스크립트 문장으로 기술한다. 다른 명령어 인자의 식과 달리 문장으로 구성되므로 여러개의 JScript 문장을 사용할 수 있다.

## Built-In 함수

템플릿의 명령어에서 사용될 수 있는 모든 **Built-In** 함수는 다음과 같다.

시그니처	설 명	대상 템플릿
StarUMLApp(): IStarUMLApplication	StarUML Application COM 객체를 반환한다.	WORD, EXCEL, POWERPOINT
StarUMLProject(): IUMLProject	StarUML Application에서 최상위 프로젝트의 COM 객체를 반환한다.	TEXT
MSWord(): WordApplication	Word Application의 COM 객체를 반환한다.	WORD
MSExcel(): ExcelApplication	Excel Application의 COM 객체를 반환한다.	EXCEL
MSPPT(): PowerpointApplication	Powerpoint Application의 COM 객체를 반환한다.	POWERPOINT
findByFullpath(Path): IElement	인자로 받은 전체 경로명에 존재하는 모델 요소를 찾아서 반환한다.	WORD, EXCEL, POWERPOINT, TEXT
findByLocalpath(RootElem, Path): IElement	인자로 받은 RootElem 모델 요소를 기준으로 Path 상대 경로에 존재하는 요소를 찾아서 반환한다.	WORD, EXCEL, POWERPOINT, TEXT
itemCount(RootElem, CollectionName): int	RootElem 모델 요소의 CollectionName을 만족하는 Collection내의 객체 개수를 반환하는 함수이다.	WORD, EXCEL, POWERPOINT, TEXT
item(RootElem, CollectionName, Index): IElement	RootElem 모델 요소의 CollectionName을 만족하는 Collection내에서 Index 위치에 존재하는 객체를 반환하는 함수이다.	WORD, EXCEL, POWERPOINT, TEXT
attr(Elem, AttrName): Value	Elem 모델 요소에서 AttrName과 일치하는 Attribute 또는 Reference의 값을 반환하는 함수이다.	WORD, EXCEL, POWERPOINT, TEXT
current(): IElement	명령어에 의해서 현재 선택된 모델 요소를 반환한다.	WORD, EXCEL, POWERPOINT, TEXT
pos(): int	명령어에 의해서 현재 선택된 모델 요소가 상위 요소의 몇 번째 자식 요소인지 위치 정보를 반환한다.	WORD, EXCEL, POWERPOINT
createFile(path): TextStream	인자로 준 경로명을 가지는 파일을 생성하여 TextStream 객체를 반환한다.	TEXT
deleteFile(path)	인자로 준 경로명을 가지는 파일을 삭제한다.	TEXT
createFolder(path): Folder	인자로 준 경로명을 가지는 폴더를 생성하여 Folder 객체를 반환한다.	TEXT
deleteFolder (path)	인자로 준 경로명을 가지는 폴더를 삭제한다.	TEXT
fileExists(path): Boolean	인자로 준 경로명의 파일이 존재하는지 여부를 반환한다.	TEXT
folderExists(path): Boolean	인자로 준 경로명의 폴더가 존재하는지 여부를 반환한다.	TEXT
fileBegin(path)	인자로 준 경로명의 파일을 생성하고, fileEnd 함수가 나타나기 전까지의 모든 출력을 현재 생성된 파일에 출력한다.	TEXT
fileEnd(path)	앞의 fileBegin 함수의 호출과 항상 대응하여 호출되며, 현재 파일로 출력되는 것을 멈추기 위해서 사용한다.	TEXT



getTarget(path): String	현재 StarUML Generator™ UI에서 설정한 출력 경로명을 반환한다.	TEXT
-------------------------	--	------

## Text 기반 템플릿 작성하기

텍스트 템플릿 작성을 위해서 사전에 다음과 같은 절차를 수행한다.

1. 텍스트 템플릿 생성용 샘플 문서(template-text.zip)를 StarUML 공식 사이트(www.staruml.com)의 템플릿 다운로드로부터 다운 받는다.
2. staruml-generator\templates 폴더 밑에 template-text라는 이름의 폴더를 생성해서 다운 받은 파일의 압축을 푼다.
3. StarUML을 실행한다.
4. **[Tools] -> [StarUML Generator...]** 메뉴를 클릭한다.
5. **[Select templates for generator]** 페이지에서 "Default Code Template" 템플릿을 선택한다.
6. **[Clone Template]** 버튼을 클릭하고 템플릿의 이름과 템플릿이 저장될 경로를 지정하고, **[OK]** 버튼을 클릭한다.
7. **[List of templates]** 에서 새로 생성한 템플릿을 선택하고, **[Open Template]** 버튼을 클릭하면 편집할 새로운 텍스트 템플릿이 열려진다.
8. 열려진 화면에서 다음의 규칙에 따라서 명령어를 작성하도록 한다.

템플릿 구성요소에서 설명한 명령어들은 각 명령어의 의미와 인자의 순서는 같지만 각 템플릿마다 명령어를 표현하는 방식이 다르다. 텍스트 템플릿은 명령어를 텍스트 문장내에서 <@ 와 @> 문자열 사이에 배치한다. 명령어의 이름은 <@ 문자열 바로 다음에 위치한다. 그리고 첫번째 인자는 명령어 이름 다음 한 칸의 공백 이후에 인자가 위치한다. 인자와 인자 사이의 구분은 세미콜론 ";"을 사용으로 한다. <@ ... @> 명령어 영역 바깥에 존재하는 문자열은 스타일로 취급되며, 생성시 문서에 그대로 출력된다.

"::Design Model" 경로 밑의 모든 하부 경로에 존재하는 UMLClass를 열거하는 경우를 REPEAT 명령어로 표현하면 다음과 같다.  
To iterate "UMLClass" typed element existing in all sub path under "::Design Model" path, do as following.

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

그리고 다음과 같이 REPEAT과 ENDREPEAT 명령어 사이에 DISPLAY 명령어를 위치시켜서, 반복하는 각 클래스의 이름과 설명을 출력하고, 자바 클래스 선언 형식으로 출력되도록 "class", "{", "/", "}" 등의 문자열을 <@ ... @> 명령어 영역 바깥쪽의 원하는 위치에 배치한다.

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

텍스트 템플릿에만 특수하게 DISPLAY 명령어와 비슷하지만 경로 인자를 가지지 않는 단축형 명령을 가지고 있다. 형태는 <@=출력값 표현식@> 형태이며, DISPLAY 명령어의 두번째 인자만을 사용하여 표현할 경우에 사용한다. 위의 템플릿을 <@= @>를 사용한 동등한 표현은 다음과 같다.

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

IF와 ENDIF 명령어를 사용하면, 클래스에 대한 설명이 존재할 경우에 대해서만 자바 커멘트 부분이 출력되도록 할 수 있다.

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

명령어의 인자로 사용되는 식은 JScript 표현식 구문을 이용한다. 이때 식에서 Built-In 함수들이 사용된다. 만약 Built-In 이외의 함수를 사용하려면, SCRIPT 명령어를 이용하여 JScript 구문으로 새로운 함수를 정의하고, 다른 명령어의 인자식에서 사용한다.

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

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

제공되는 명령어 이외의 특수한 처리를 위해서도 SCRIPT 명령어가 사용될 수 있다. 다음은 반복하는 클래스를 각각 다른 클래스 파일명으로 저장하기 위한 명령을 SCRIPT 명령과 Built-In 함수인 fileBegin, fileEnd로 구성한 예이다.

```

<@REPEAT {R}::Design Model:UMLClass::@>

<@SCRIPT fileBegin(getTarget()+"\\WW"+current().Name+".java"); @>

class <@DISPLAY :current().Name@> {
    // <@DISPLAY :current().Documentation@>
}

<@SCRIPT fileEnd(); @>
<@ENDREPEAT@>

```


모든 명령어에 대한 편집을 완료하고 문서를 저장하면, 직접 작성한 템플릿을 이용하여 코드를 생성할 수 있다. 문서 생성에 대한 자세한 절차는 사용자 가이드 7장. **Generating by template**을 참조한다.

## Word 템플릿 작성하기

WORD 템플릿 작성을 위해서 사전에 다음과 같은 절차를 수행한다.

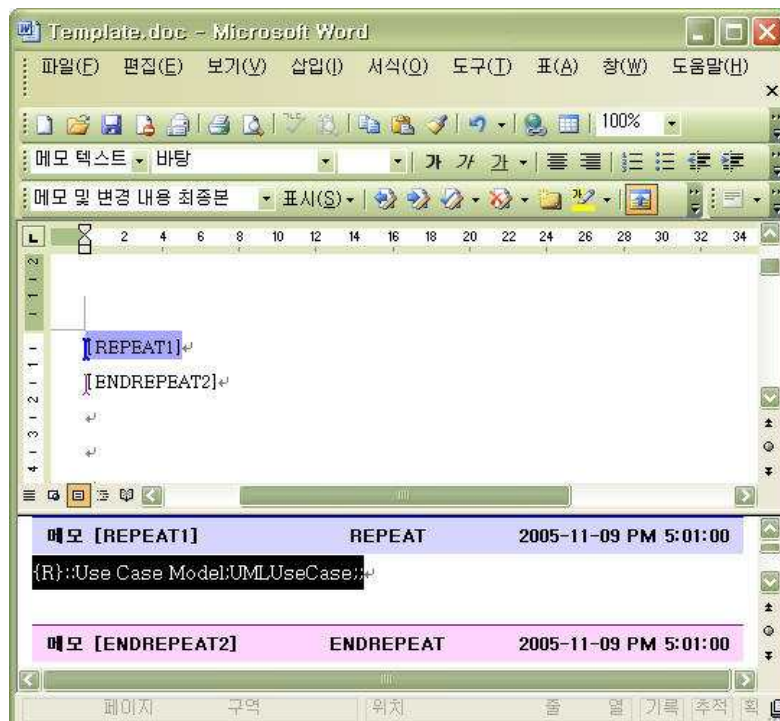
1. WORD 템플릿 생성용 샘플 문서(template-word.zip)를 StarUML 공식 사이트([www.staruml.com](http://www.staruml.com))의 템플릿 다운로드로부터 다운 받는다.
2. staruml-generator\templates 폴더 밑에 template-word라는 이름의 폴더를 생성해서 다운 받은 파일의 압축을 푼다.
3. StarUML을 실행한다.
4. **[Tools] -> [StarUML Generator...]** 메뉴를 클릭한다.
5. **[Select templates for generator]** 페이지에서 "Default Word Template" 템플릿을 선택한다.
6. **[Clone Template]** 버튼을 클릭하고 템플릿의 이름과 템플릿이 저장될 경로를 지정하고, **[OK]** 버튼을 클릭한다.
7. **[List of templates]** 에서 새로 생성한 템플릿을 선택하고, **[Open Template]** 버튼을 클릭하면 복사한 워드 템플릿이 열린다.
8. 열린 워드 화면에서 다음의 규칙에 따라서 명령어를 작성하도록 한다.

WORD 템플릿에서 명령어 영역은 메모를 사용하여 표현한다. 명령어 이름은 메모의 작성자 속성에 표시하고, 인자들은 메모의 내용 속성에 표시한다. 인자와 인자 사이의 구분은 세미콜론 ";"을 사용으로 한다. 메모 이외의 영역은 스타일 영역으로 취급되며, 생성시 문서에 그대로 출력된다.

"::Use case Model" 경로 아래의 모든 하부 경로에 존재하는 UMLUseCase를 열거하는 경우를 표현하려면, 우선 **[REPEAT]**과 **[ENDREPEAT]** 이름의 메모를 복사하여 원하는 위치에 붙인다. 그리고 **[REPEAT]** 메모를 선택하고, REPEAT 명령어 인자 설정을 위해서 워드의 검토 버튼  을 클릭한다. 검토창이 나타나면 다음 그림과 같이 **[REPEAT]** 메모의 내용 속성에 인자를 입력한다.

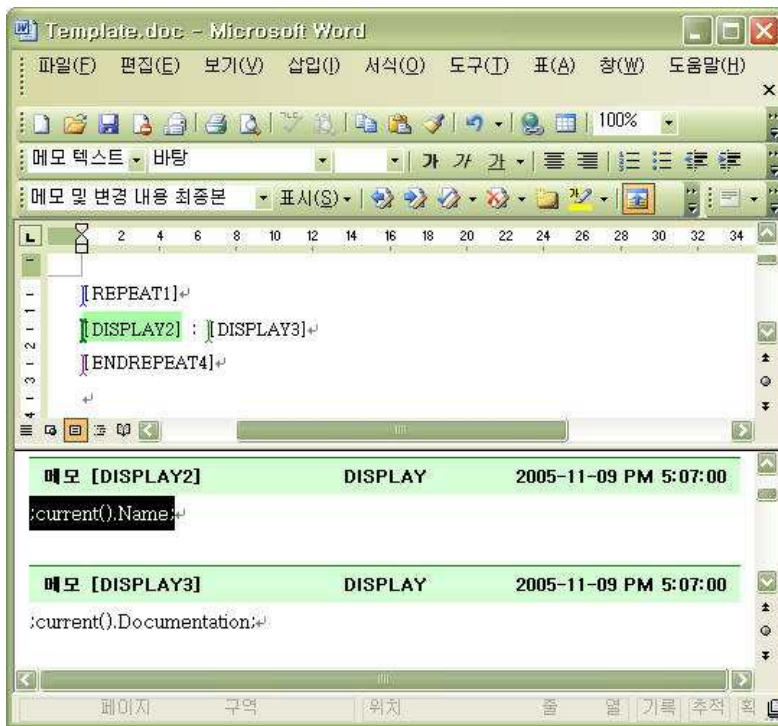
### 참고

- 메모의 작성자 속성은 사용자에 의해서 임의로 수정될 수 없다. 따라서, 현재의 템플릿내부에 이미 존재하는 메모 명령어를 원하는 위치에 복사하여 사용한다.

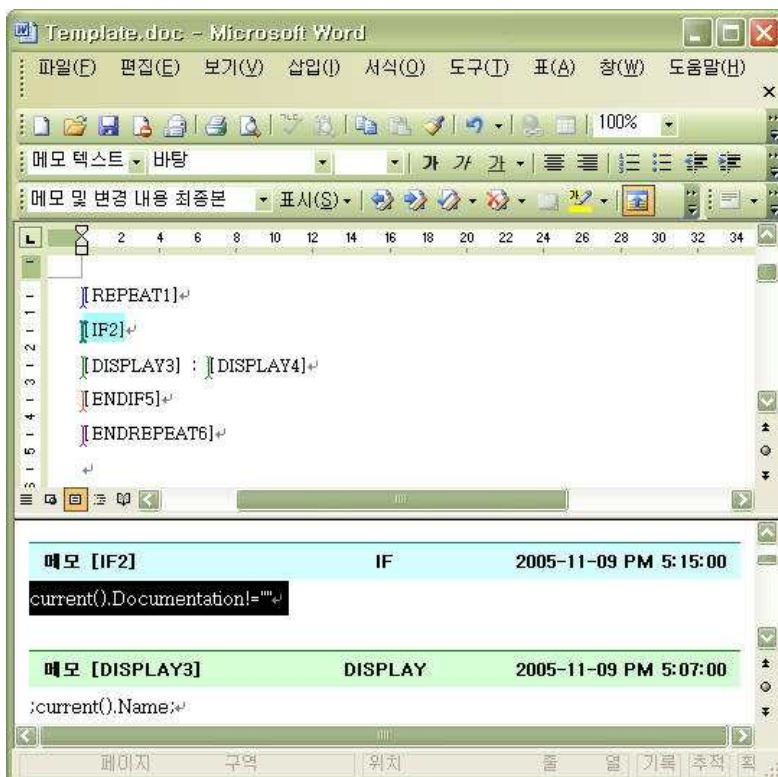


그리고 **[REPEAT]**과 **[ENDREPEAT]** 명령어 사이에 **[DISPLAY]** 명령어를 복사하고, 인자의 값을 다음과 같이 수정하면, "::Use case Model" 경로 아래의 모든 유스케이스를 반복하면서 유스케이스의 이름과 설명을 출력한다.

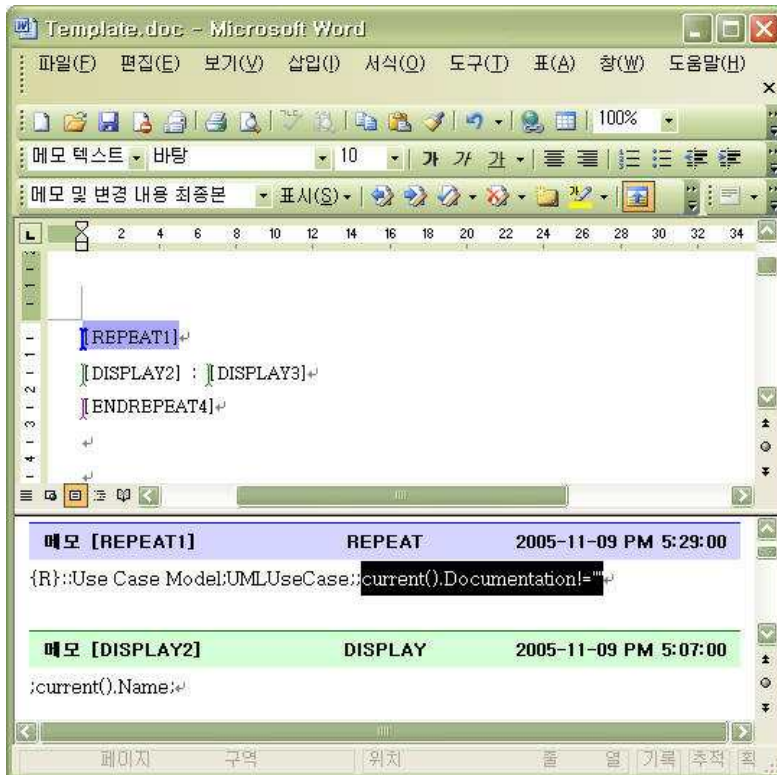




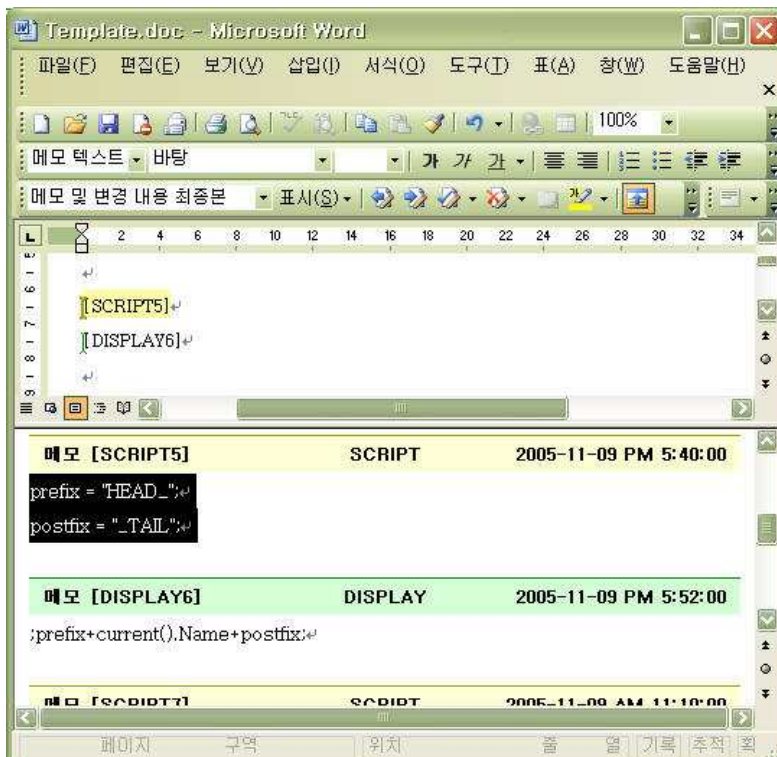
다음과 같이 [IF], [ENDIF] 명령어를 작성하면, 특정 조건을 만족하는 경우에 대해서 명령어가 수행되거나 출력되도록 할 수 있다.



[REPEAT]과 [IF] 명령어의 조합은 [REPEAT] 명령어 하나로 대체 가능하다. [IF] 명령어의 조건 인자를 [REPEAT] 명령어의 조건 인자로 옮기고, [IF]와 [ENDIF] 명령어를 삭제하면 동일한 의미가 된다.



워드 템플릿내도 다른 템플릿과 마찬가지로 **SCRIPT** 명령어를 이용하여 원하는 JScript 구문을 워드 문서 생성중에 실행할 수 있다. JScript 구문을 실행한 후 결과 값을 화면에 출력하기를 원한다면, **[SCRIPT]** 명령어의 내용 속성에 JScript 문장을 기술하고 특정 변수를 설정하도록 하며, **[DISPLAY]** 명령어의 인자에 설정한 변수 이름을 위치하면 된다.



워드 템플릿에는 **[REPEAT]** 명령어를 테이블의 특정행을 반복하기 위해서 사용할 수 있다. **[REPEAT]** 명령어에 인자를 설정하는 방식은 동일하지만 대응되는 명령어가 **[ENDREPTR]** 명령어이다. 그리고 **[REPEAT]**과 **[ENDREPTR]** 명령어의 위치는 테이블에서 반복해서 생성되기를 원하는 행의 첫번째 셀에 항상 **[REPEAT]** 명령어가 위치해야하며, 반복 생성을 원하는 행의 마지막 셀에 반드시 **[ENDREPTR]** 명령어가 위치해야 한다. 다음은 Usecase의 이름과 설명을 테이블로 생성하는 예이다.



모든 명령어에 대한 편집을 완료하고 문서를 저장하면, 직접 작성한 워드 템플릿을 이용하여 문서 생성이 가능하다. 문서 생성에 대한 자세한 절차는 사용자 가이드 7장. Generating by template을 참조한다.

## Excel 템플릿 작성하기

EXCEL 템플릿 작성을 위해서 사전에 다음과 같은 절차를 수행한다.

1. EXCEL 템플릿 생성용 샘플 문서(template-excel.zip)를 StarUML 공식 사이트([www.staruml.com](http://www.staruml.com))의 템플릿 다운로드로부터 다운 받는다.
2. staruml-generator\templates 폴더 밑에 template-excel라는 이름의 폴더를 생성해서 다운 받은 파일의 압축을 푼다.
3. StarUML을 실행한다.
4. **[Tools] -> [StarUML Generator...]** 메뉴를 클릭한다.
5. **[Select templates for generator]** 페이지에서 "Default Excel Template" 템플릿을 선택한다.
6. **[Clone Template]** 버튼을 클릭하고 템플릿의 이름과 템플릿이 저장될 경로를 지정하고, **[OK]** 버튼을 클릭한다.
7. **[List of templates]** 에서 새로 생성한 템플릿을 선택하고, **[Open Template]** 버튼을 클릭하면 엑셀에서 복사한 엑셀 템플릿이 열린다.
8. 열린 엑셀 화면에서 다음의 규칙에 따라서 명령어를 작성하도록 한다.

엑셀 템플릿에서 명령어 영역은 메모를 사용하여 표현한다. 명령어 이름과 인자들은 메모의 내용 속성에 표시한다. 메모의 내용은 명령어 이름, 그리고 인자들이 오는 순서로 구성된다. 그리고 명령어 이름과 인자, 인자와 인자 사이의 구분은 세미콜론 ";"을 사용으로 한다. 메모 이외의 영역은 스타일 영역으로 취급되며, 생성시 문서에 그대로 출력된다.

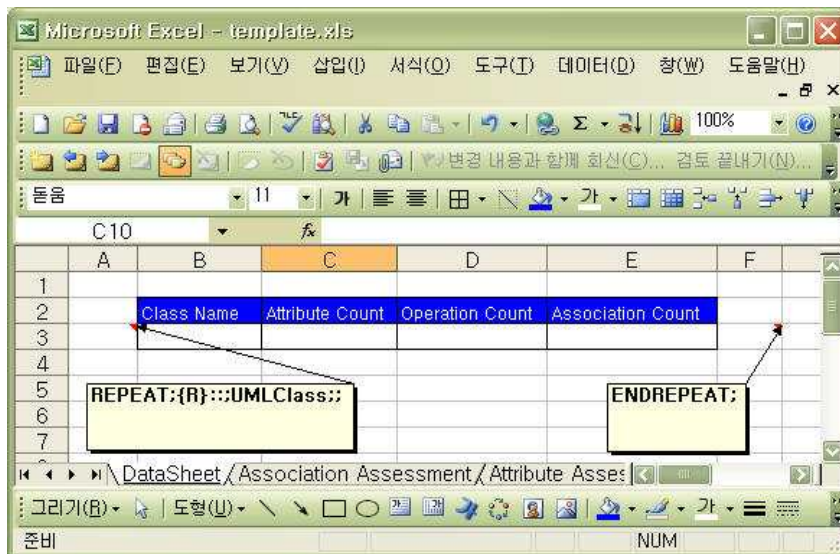
엑셀 템플릿은 엑셀의 장점을 충분히 이용하여, 모델 정보로부터 통계나 수치 데이터를 추출하여 분석하고 판단할 수 있다. 이 절에서는 현재 모델링된 정보나 역공학을 통해서 읽어들이 모델 정보로부터 클래스 관련 정보에 대한 수치 데이터를 추출하여 그래프화 하는 방법을 설명한다.

모델로부터 클래스 관련 정보를 추출하기 위해서 우선 모델에 포함된 모든 클래스를 나열해야 한다. REPEAT 명령어를 이용하여 클래스가 출력될 위치와 반복할 클래스에 대한 관련 인자를 다음과 같이 설정한다.

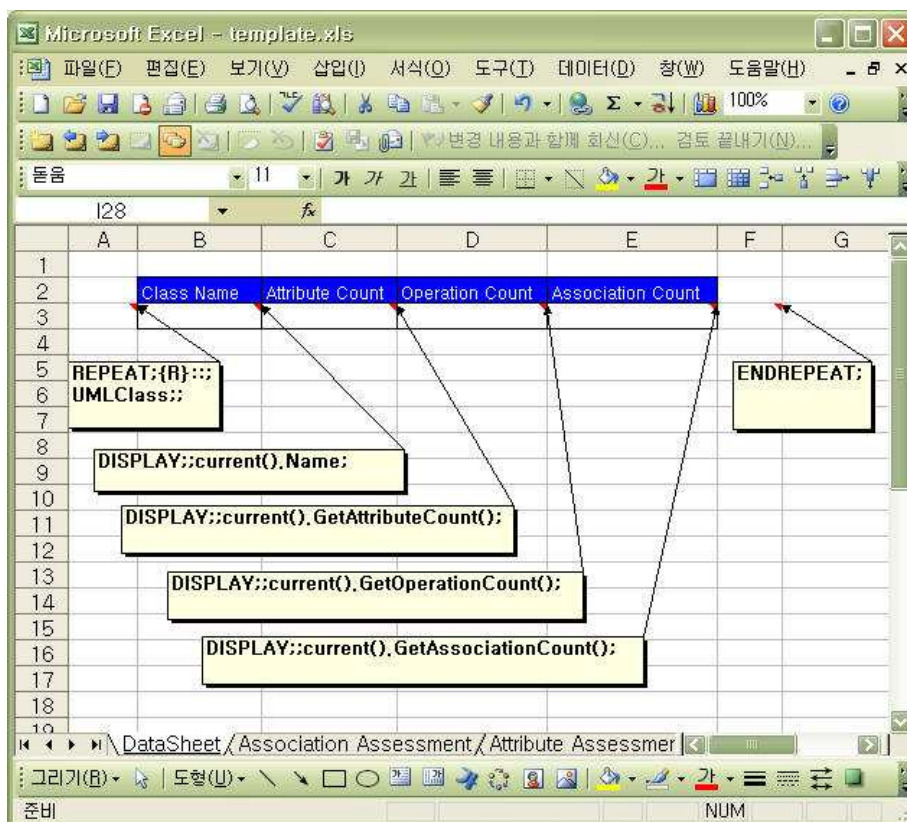
### 주의

- EXCEL 템플릿에서 REPEAT 명령어는 행 단위로 증가하는 반복만을 제공한다. 행내의 열 단위로 증가하는 반복은 제공하지 않는다.

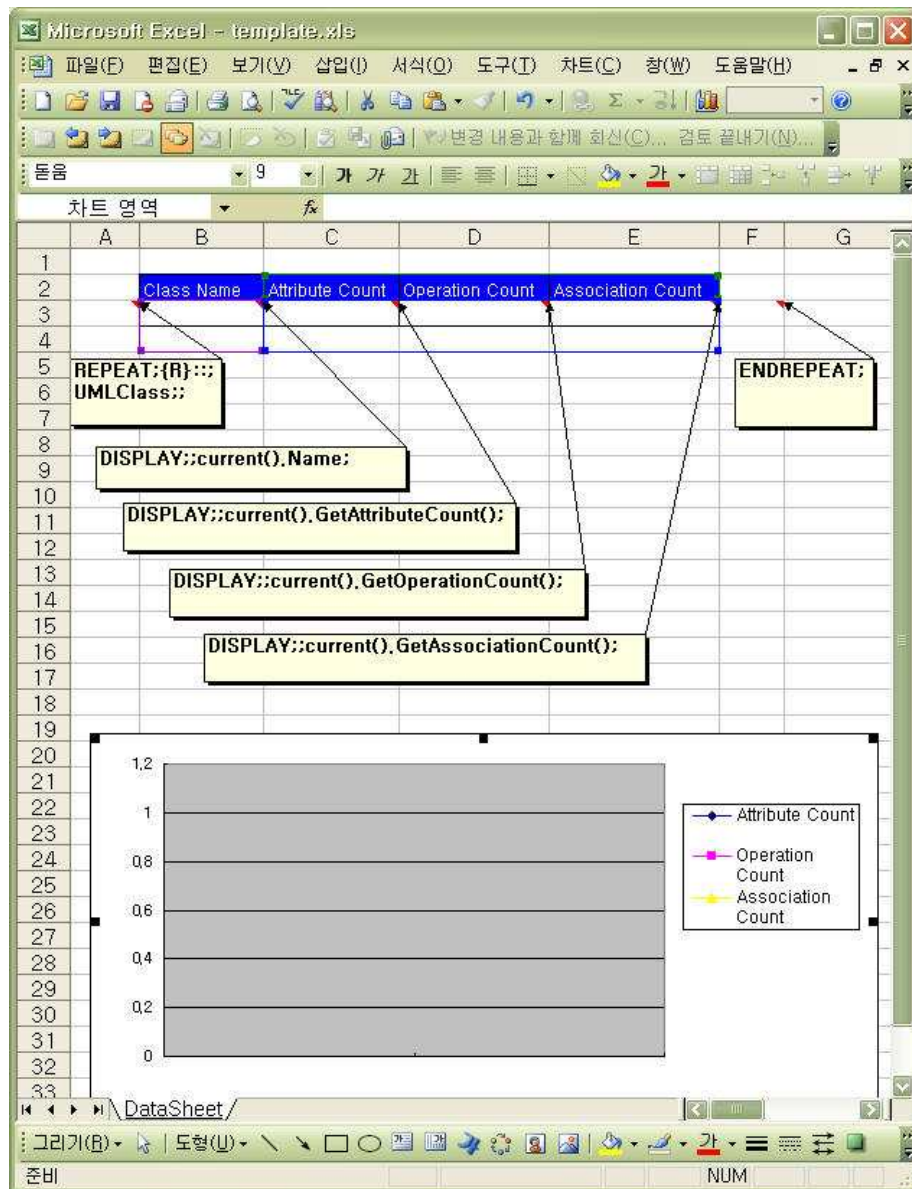




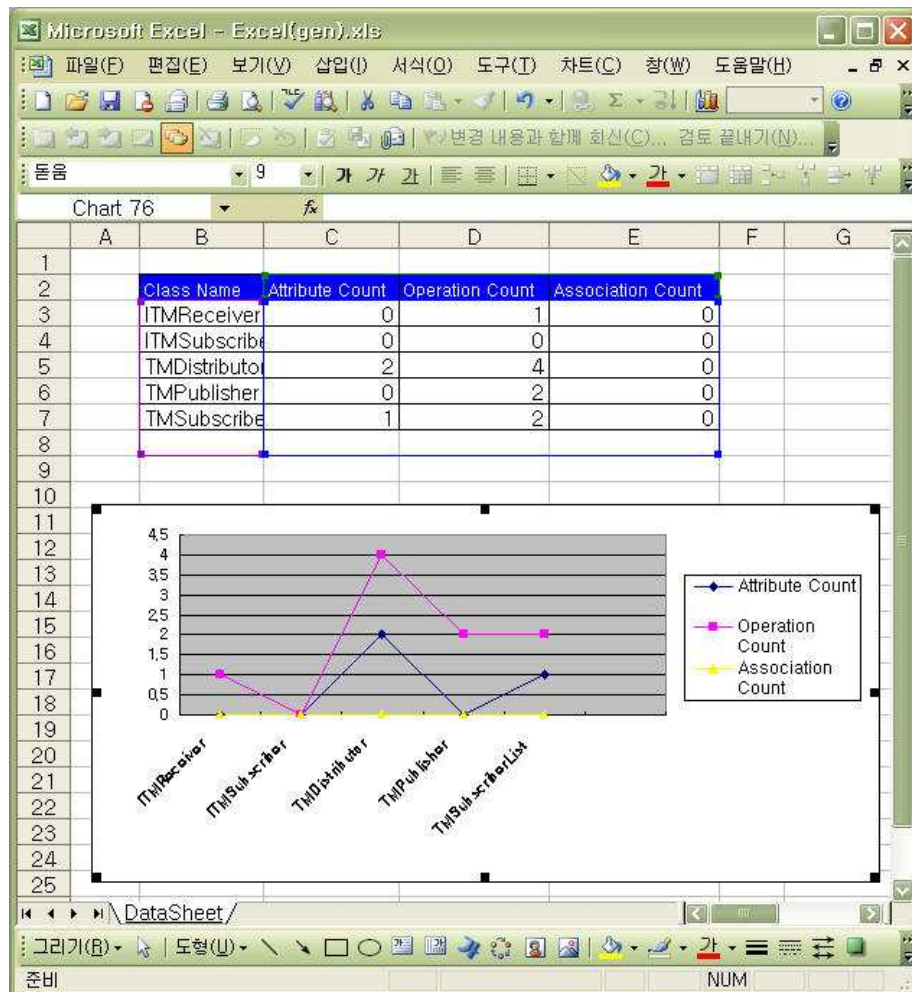
그리고 REPEAT과 ENDREPEAT 명령어 사이에 DISPLAY 명령어(메모)를 삽입하고, 각각 클래스의 이름, 속성 개수, 연산 개수, 연관 개수를 가져와서 출력할 수 있도록 인자의 값을 다음과 같이 수정한다.



여기에 다시 엑셀 차트를 삽입하고 원본 데이터를 Attribute, Operation, Association Count로 지정하면 모든 설정이 완료된다.



모든 명령어에 대한 편집을 완료하고 문서를 저장하면, 직접 작성한 워드 템플릿을 이용하여 문서 생성이 가능하다(문서 생성에 대한 자세한 절차는 사용자 가이드 7장 **Generating by template**을 참조한다). 다음 그림은 클래스와 관련된 수치 데이터를 자동 생성한 결과이다. 엑셀 명령어와 엑셀의 다양한 차트를 혼합하여 활용함으로써, Q/A 팀이나 프로젝트 팀 등에서 필요한 설계 관련 수치 정보를 쉽게 분석하고 판단할 수 있다.



## PowerPoint 템플릿 작성하기

POWERPOINT 템플릿 작성을 위해서 사전에 다음과 같은 절차를 수행한다.

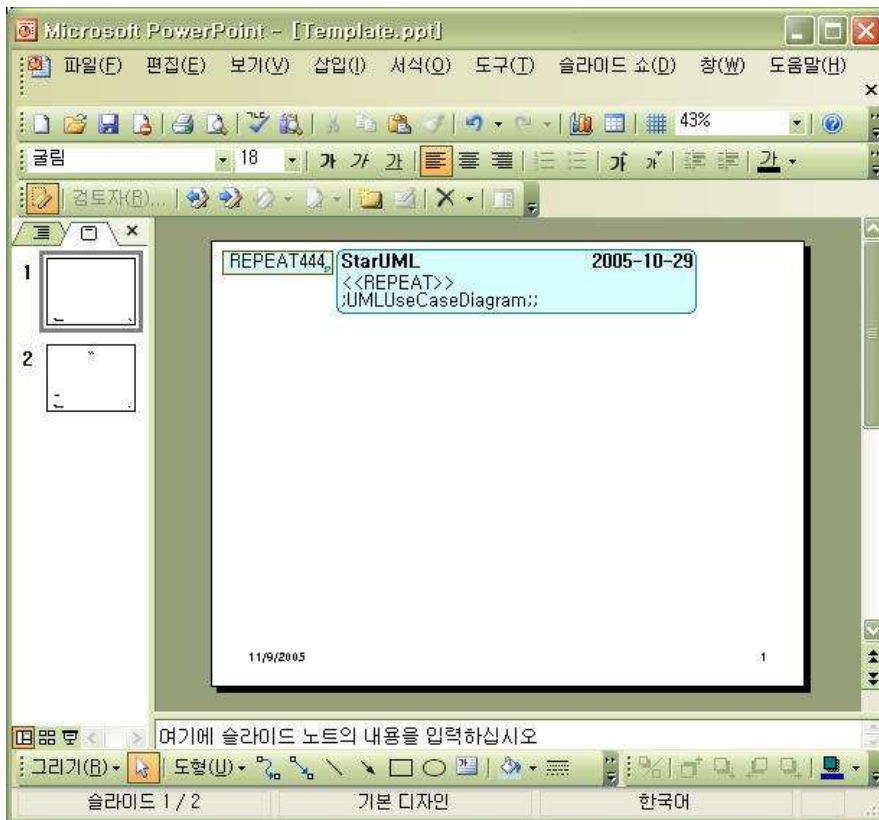
1. POWERPOINT 템플릿 생성용 샘플 문서(template-powerpoint.zip)를 StarUML 공식 사이트(www.staruml.com)의 템플릿 다운로드로부터 다운 받는다.
2. staruml-generator\templates 폴더 밑에 template-powerpoint라는 이름의 폴더를 생성해서 다운 받은 파일의 압축을 푼다.
3. StarUML을 실행한다.
4. **[Tools] -> [StarUML Generator...]** 메뉴를 클릭한다.
5. **[Select templates for generator]** 페이지에서 "Default Powerpoint Template" 템플릿을 선택한다.
6. **[Clone Template]** 버튼을 클릭하고 템플릿의 이름과 템플릿이 저장될 경로를 지정하고, **[OK]** 버튼을 클릭한다.
7. **[List of templates]** 에서 새로 생성한 템플릿을 선택하고, **[Open Template]** 버튼을 클릭하면 파워포인트에서 복사한 POWERPOINT 템플릿이 열린다.
8. 열린 POWERPOINT 화면에서 다음의 규칙에 따라서 명령어를 작성하도록 한다.

POWERPOINT 템플릿에서 명령어 영역은 메모를 사용하여 표현한다. 명령어 이름은 메모의 내용 속성 첫째 줄에 << 와 >> 문자열 사이에 표시하고, 인자들은 메모의 내용 속성 두번째 줄에 표시한다. 인자와 인자 사이의 구분은 세미콜론 ";"을 사용으로 한다. 메모 이외의 영역은 스타일 영역으로 취급되며, 생성시 문서에 그대로 출력된다.

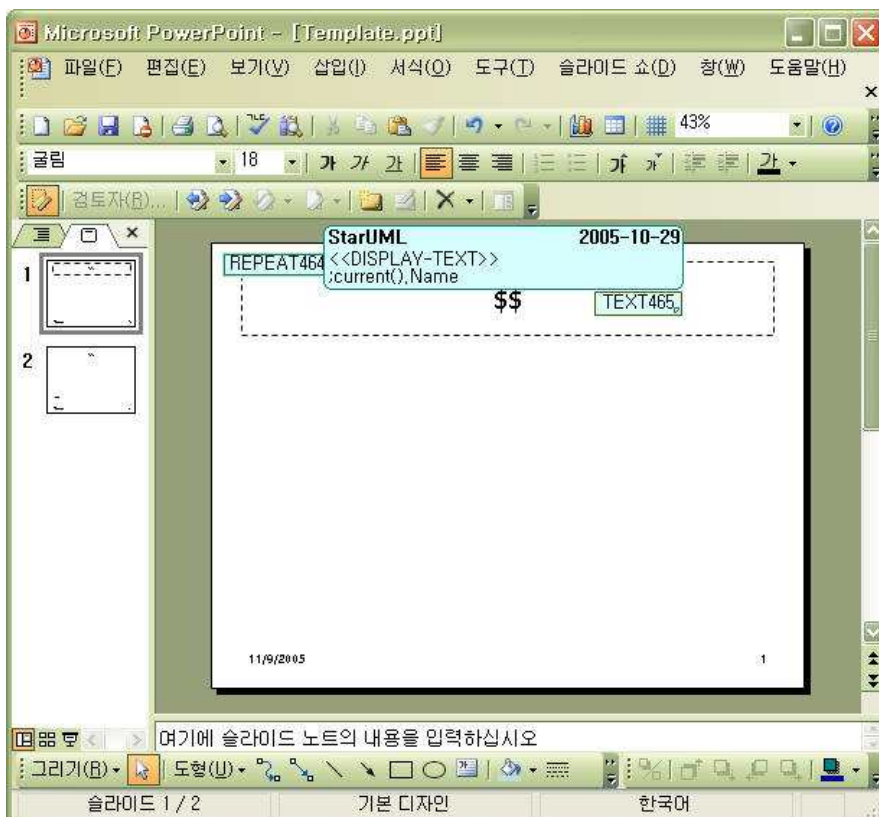
프로젝트 내부의 유스케이스 다이어그램과 다이어그램의 설명을 슬라이드로 생성하는 POWERPOINT 템플릿 작성 경우를 예를 들어 알아보자. 우선 하나의 슬라이드에 하나의 다이어그램이 열거되도록 템플릿을 구성하기 위해서 해당 슬라이드 왼쪽 상단 메모를 추가하고 다음과 같이 REPEAT 명령어의 인자를 설정한다. 이때 ENDREPEAT을 바로 설정하지 않도록 한다. 이유는 잠시 후에 설명한다.

### 주의

- POWERPOINT 템플릿에서 REPEAT 명령어는 슬라이드 단위의 반복만을 제공한다. 또한 슬라이드내에 존재하는 개별 요소의 반복은 제공하지 않는다.

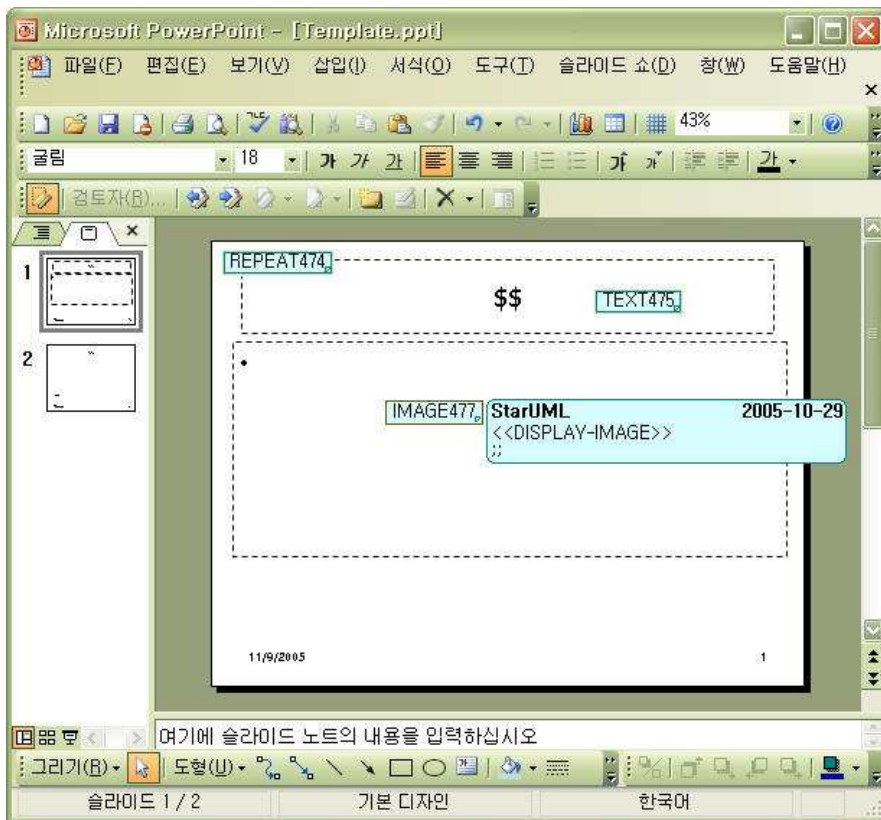


그 다음에 UseCaseDiagram의 이름이 슬라이드의 제목으로 나올수 있도록 텍스트 박스와 DISPLAY-TEXT 명령어를 추가하고 다음과 같이 인자를 설정한다. 그리고 텍스트 박스에는 \$\$라는 문자열을 삽입하여 DISPLAY-TEXT 명령어가 어느 위치에 텍스트를 출력할지 지정한다. 이때 주의할 점은 DISPLAY-TEXT와 다음에 설명할 DISPLAY-IMAGE 명령어는 명령어 영역을 포함하는 텍스트 박스속에 DISPLAY-... 명령어의 수행 결과를 출력한다. 따라서 반드시 텍스트 박스의 테두리 안에 DISPLAY-... 명령어가 위치해야 한다.

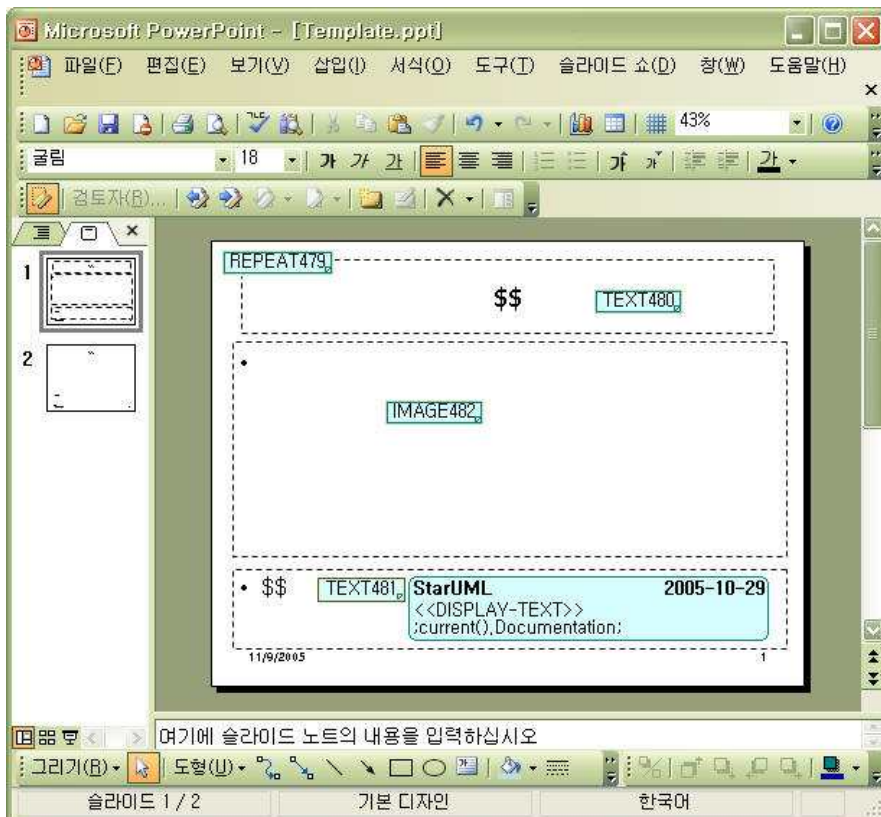


이제 다이어그램이 슬라이드 중앙에 출력하기 위해서, 텍스트 박스를 이용하여 출력할 위치와 크기를 잡는다. 그리고 DISPLAY-IMAGE 명령어를 추가하고 다음과 같이 인자를 설정한다.

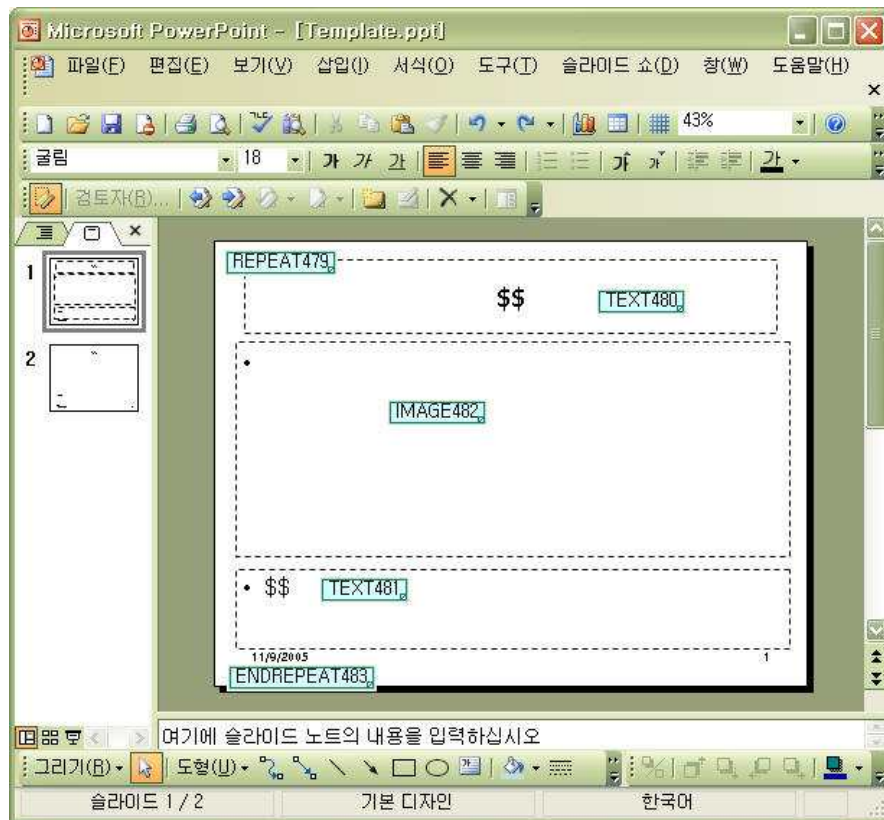




그리고 다이어그램의 설명 내용이 슬라이드 하단에 출력될 수 있도록 **DISPLAY-TEXT** 명령어와 텍스트 박스를 추가하고, 인자를 다음과 같이 설정한다.




마지막으로 반복이 되는 범위가 되는 슬라이드의 하단에 **ENDREPEAT** 명령어를 추가한다. **ENDREPEAT** 명령어 추가를 미룬 이유는 **Generator**의 메모 해석의 순서가 메모의 위치가 아니라 생성 순서에 의존하기 때문이다. 즉 슬라이드 내에서 명령어가 위에 있다고 먼저 실행되는 것이 아니라, 명령어에 부여된 인덱스 번호가 앞선 것이 먼저 실행된다. 따라서 **REPEAT**, **ENDREPEAT**, **DISPLAY-TEXT** 순서로 명령어를 추가하면, **REPEAT**과 **ENDREPEAT** 사이에는 어떠한 명령도 없는 것으로 해석한다. 따라서 **REPEAT** 명령어에 의해서 또다른 명령어를 반복하려한다면, 반드시 **REPEAT** 명령어 보다 인덱스 번호가 느리면서 **ENDREPEAT** 명령어보다 인덱스가 앞서게 명령어를 추가, 삭제해야 한다.



모든 명령어에 대한 편집을 완료했다면 문서를 현재 열려진 파일명 그대로 저장한다. 이제부터 직접 작성한 POWERPOINT 템플릿을 이용하여 문서 생성이 가능하다. 문서 생성에 대한 자세한 절차는 사용자 가이드 7장. **Generating by template**을 참조한다.

## 템플릿 등록하기

사용자가 직접 작성한 자신만의 템플릿 문서를 Generator에 등록할 수 있습니다.

1. **[Select templates for generation]** 페이지에서 **[Register Template]** 버튼을 클릭합니다.
2. **[Register Template]** 다이얼로그가 나타나면, 
3. **[Properties:]** 입력창에서 등록할 템플릿의 정보들을 입력하고 **[OK]** 버튼을 클릭하여 템플릿을 등록합니다.

### 기본 정보

템플릿 등록시에 템플릿명과 그룹, 분류, 설명에 대한 기본정보를 설정합니다.

항 목	설 명
Template Name	등록될 템플릿명을 지정합니다.
Group	템플릿이 속하는 그룹을 지정합니다. 기존에 있는 그룹 외에 새로운 그룹을 정의할 수 있습니다.
Category	프로젝트 관리, 요구사항 등의 분류를 지정합니다.
Description	템플릿의 개략적인 설명을 기술한다.




### 상세 정보

템플릿 등록시에 필요한 상세 정보들을 설정합니다.

항 목	설 명
Document Type	템플릿의 문서 형태를 지정합니다. DOCUMENT(문서), REPORT(보고서), CODE(소스코드) 중 하나를 선택합니다.
Format	문서 생성의 결과로 생성될 문서의 포맷을 지정합니다.
Version	템플릿의 버전 정보를 "1.0"과 같은 형식을 기록합니다.
Related Profile	템플릿을 적용할 모델에 포함할 프로파일들을 선택합니다.
Related Approach	템플릿을 적용할 모델의 기본 접근법을 선택합니다.
	문서 생성기의 종류를 명시합니다. WORD(워드 문서 생성기), EXCEL(엑셀 문서 생성기), POWERPOINT(파워포인트 문서 생성기), TEXT(텍스트 문서 생성기), COM(COM 컴포넌트 기반의 사용자 정의 생성기), SCRIPT(스크립트 기반의 사용자 정의 생성기), EXE(실행파일 기반의 사용자 정의 생성기) 중 하나를 선택합니다.

Translator Type	Value	Meaning
	WORD	word document generator
	EXCEL	excel document generator
	POWERPOINT	powerpoint document generator
	TEXT	code generator
	COM	COM-based generator defined by user
	SCRIPT	Script-based generator defined by user
	EXE	Executable file-typed generator made by user
Translator	문서 생성기의 이름을 명시합니다. 빌트인 문서 생성기(WORD, EXCEL, POWERPOINT, TEXT)는 지정하지 않습니다. 사용자 정의 생성기는 생성기의 파일명을 적습니다.	
Example	문서 생성에 맞는 예제 모델 파일을 지정합니다.	
Parameters	문서 생성에 필요한 인자를 정의합니다.	
Related files	문서 생성에 필요한 부가적인 파일들을 지정합니다.	

## 파라미터 정보

1. Parameters 항목에서  버튼을 클릭합니다.
2. **[Parameters]** 다이얼로그가 나타나면, 새로운 파라미터를 삽입하기 위해서  버튼을 클릭하며, 파라미터를 삭제하기 위해서는  버튼을 클릭한다.
3. **[New Parameter]** 다이얼로그가 나타나면 파라미터의 이름과 타입, 그리고 디폴트값을 입력하고 **[OK]** 버튼을 클릭한다.

Translator Type에 따른 기본 파라미터 목록을 다음과 같이 설정합니다.

항 목	타 입	대상 템플릿	설 명
TemplateFile	FILENAME or STRING	WORD, EXCEL, POWERPOINT	문서 생성시에 사용될 템플릿 문서 파일명을 지정합니다.
OutputFile	FILENAME or STRING	WORD, EXCEL, POWERPOINT, TEXT	문서 생성의 결과로 생성될 파일명을 지정합니다.
Keep Comment	BOOLEAN	WORD, EXCEL, POWERPOINT	생성된 문서내에 커맨드 정보를 남길지 여부를 지정합니다.
ShowGenerationProcess	BOOLEAN	WORD, EXCEL, POWERPOINT	MS Office에 일어나는 생성 과정을 보여줄지 여부를 지정합니다. 이 항목이 true값으로 설정되면 생성시 성능이 떨어질 수 있습니다.
Normal Generation	BOOLEAN	WORD	커맨드의 상대 경로를 최상위 프로젝트로 지정합니다. 만약 false로 설정되면 현재 모델에서 선택된 요소에서부터 상대 경로가 지정됩니다.
Generate Index	BOOLEAN	WORD	생성된 문서내에서 색인 요소를 검색하여, 문서 색인을 생성합니다.
intermediate	STRING	TEXT	코드 생성에 필요한 중간 코드를 저장할 파일명을 지정합니다.
target	STRING	TEXT	생성되는 코드 파일이 여러개일 경우에 코드 파일들이 존재할 최상위 경로명을 지정합니다.

## 참고

- 파일명과 관련된 파라미터 값을 설정하는 경우에는 **StarUML Generator**에 명시된 상수를 사용할 수 있습니다. 상수에는 다음과 같은 것들이 있습니다.

이 름	설 명
\$PATH\$	현재 템플릿 정보와 템플릿이 존재하는 디렉토리 경로를 의미합니다. (예) \$PATH\$\BusinessActorReport.doc
\$GROUP\$	그룹명을 의미합니다.
\$CATEGORY\$	분류명을 의미합니다.
\$NAME\$	템플릿명을 의미합니다.
\$TARGET\$	<b>[Generator]</b> 다이얼로그에서 사용자가 지정한 문서 생성 경로를 의미합니다.

등록된 템플릿을 다루는 방법에 대해서는 사용자 가이드 7장. Generating Codes and Templates의 Generating by Template을 참조한다.

## 템플릿 배포하기

템플릿은 staruml-generator 하단에 설치된다. staruml-generator 하단에 templates 폴더에 StarUML Generator에서 사용하는 모든 템플릿들이 존재하고, 또한 배치 작업 목록들도 존재한다. 일반적으로 한개의 템플릿은 한개의 폴더 안에 관련된 모든 리소스 파일을 포함하고 있으며, 모든 템플릿 폴더들은 templates 폴더의 바로 아래에 위치한다. 하나의 템플릿은 템플릿 정의 기술 파일(.tdf)과 템플릿 문서(.doc, .ppt, .xls, .cot) 두

가지 종류의 파일로 구성된다. 템플릿 정의 기술 파일에서는 사용자 가이드 7장. **Generating Codes and Documents** > 템플릿 등록하기 > 기본/상세/파라미터 정보에서 설정한 내용들이 저장되어 있으며, 템플릿 문서는 모델을 가져와서 보여주기 위한 명령어와 스타일을 포함하고 있습니다. 배치 목록은 **staruml-generator** 폴더 아래의 **batches** 폴더에 배치 작업 파일(.btf) 형태로 존재합니다.

파일 확장자명	설 명
BTF	배치 작업 목록과 생성 인자 정보를 포함하고 있는 XML 파일
TDF	템플릿에 대한 정보(이름, 종류, 참조 템플릿 문서명, 생성 인자등)를 기술한 XML 파일
DOC, DOT	워드 문서 생성시 사용되는 명령어와 스타일 정보를 포함하고 있는 템플릿 파일
XLS, XLT	엑셀 문서 생성시 사용되는 명령어와 스타일 정보를 포함하고 있는 템플릿 파일
PPT, POT	파워 포인트 문서 생성시 사용되는 명령어와 스타일 정보를 포함하고 있는 템플릿 파일
COT	코드 생성에 사용되는 명령어와 스타일 정보를 포함하고 있는 템플릿 파일

## Generator의 폴더 구성

staruml-generator 모듈의 폴더 구성은 다음과 같다.

```
staruml-generatorW
  templatesW
    template1W
      template1.tdf
      template1.doc
    template2W
      ...
  batchesW
    batch1.btf
    ...
```

## 템플릿 설치와 제거

템플릿을 설치하는 방법은 매우 간단하다. 만약 다른 사용자에게 자신이 작성한 템플릿 배포하여 설치하고 싶다면 **staruml-generator\templates** 폴더 아래에서 배포할 템플릿을 포함하는 폴더를 복사하고, 설치될 컴퓨터의 **staruml-generator\templates** 폴더 아래의 경로에 붙여 넣으면 설치가 종료된다.

템플릿을 제거하는 방법 또한 매우 간단하다. **staruml-generator\templates** 폴더 아래에서 삭제하고자 하는 해당 템플릿 폴더를 삭제하면 설치가 제거한다.

## 템플릿의 패키징

**staruml-generator\templates** 폴더 아래는 일반 파일처럼 단지 여러개의 템플릿 디렉토리를 또 다른 상위 폴더 아래로 위치하는 것이 가능하다. 물론 이러한 폴더의 위치 변화에 의해서 **Generator**의 목록 변화는 전혀 없다. 따라서 파일 형태의 관리나 여러 개의 템플릿을 한번에 묶어서 배포하는 것이 용이하다. 예를 들어 그룹과 카테고리가 비슷한 여러개의 템플릿 폴더들을 그룹별로 하나의 폴더 밑에 위치시켜서 사용하고, 배포시에는 해당 그룹의 폴더 또는 **templates** 폴더 전체를 압축하여 다른 컴퓨터에 배포하고, 배포된 압축 파일을 다시 통째로 **staruml-generator\templates** 아래에 풀면 쉽게 템플릿 패키지를 설치할 수 있다.

## 배치 작업의 설치와 제거

배치 작업을 설치하고 제거하는 방법도 매우 간단하다. 우선 배치 작업 목록에서 사용되는 템플릿 폴더를 설치하고, **staruml-generator\batches** 폴더 아래에서 원하는 배치작업(.btf) 파일을 복사하여, 설치될 컴퓨터의 **staruml-generator\batches** 폴더 아래로 복사하면 배치의 설치가 완료된다.

배치 작업의 설치 제거는 **staruml-generator\batches** 폴더 아래의 .btf 파일을 삭제하면 것으로써 완료됩니다.

[처음으로](#)