|  |  |  |
| --- | --- | --- |
| <http://www.ibm.com/developerworks/kr/library/os-ecplug/>  <http://www.ibm.com/developerworks/library/os-ecplug/index.html>   |  |  | | --- | --- | | Eclipse 플러그인 개발하기  **플러그인 구현, 디버그, 설치**  http://www.ibm.com/i/c.gif | developerWorks | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://www.ibm.com/i/c.gif | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | http://www.ibm.com/i/c.gif | |  | | --- | | **문서 옵션** |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | http://www.ibm.com/i/c.gif | 수평출력으로 설정 | [**이 페이지 출력**](javascript:print())  양식의 맨 위  양식의 맨 아래 | | http://www.ibm.com/i/c.gif | 이 페이지를 이메일로 보내기 | [**이 페이지를 이메일로 보내기**](javascript:document.email.submit();) | |   ﻿   |  | | --- | | **제안 및 의견** |  |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | http://www.ibm.com/i/c.gif | [피드백](https://www-128.ibm.com/developerworks/secure/feedback.jsp?domain=dwkorea) http://www.ibm.com/i/c.gif | | |   난이도 : 초급  [David Gallardo](http://www.ibm.com/developerworks/kr/library/os-ecplug/#author), Software consultant  2002 년 10 월 01 일 2003 년 6 월 24 일 수정  David Gallardo는 Plug-in Development Environment의 코드 생성 위자드를 사용하여 Eclipse 플러그인을 만드는 방법을 설명한다. 런타임 워크벤치에서 플러그인을 실행하고 디버깅하는 방법을 비롯하여 Eclipse에 완벽한 플러그인을 설치하는 방법을 배우게 될 것이다.  **플러그인 기반 아키텍쳐**  Eclipse Platform은 IBM이 개발에 쏟아 부은 4천만 달러 때문이 아니라 그러한 지출에 합당한 결과를 보여줘야한다는 이유로 주목을 받고 있다. 성숙하고, 디자인이 잘 되어있고, 확장 가능한 아키텍쳐를 기대하고 있다. Eclipse의 가치는 무엇인가? 이것은 오픈 소스 플랫폼으로서 확장 가능한 통합 개발 환경을 제공한다. 이 플랫폼으로 누구나 환경과 다른 툴을 완벽하게 통합하는 툴을 구현할 수 있다.  Eclipse와 툴의 완벽한 통합의 열쇠는 *플러그인(* *plug-in)*이다. 작은 런타임 커널을 제외하고 Eclipse의 모든 것이 플러그인이다. 따라서 당신이 개발하는 플러그인은 다른 플러그인과 같은 방식으로 Eclipse와 통합된다. 이러한 관점에서 볼 때 모든 기능들은 동등하게 만들어졌다.  하지만 어떤 플러그인은 다른 어떤 것들보다 더 비슷하다. Workbench와 Workspace는 Eclipse 플랫폼에 있어서 없어서는 안될 플러그인이다. 그림 1 처럼 대부분의 플러그인에서 사용되는 확장 포인트를 제공한다. 플러그인이 기능을 하기 위해 플러그인 될 수 있는 확장 포인트가 필요하다.  **그림1. Eclipse Workbench와 Workspace: 필수 플러그인 지원** Eclipse Workbench  Workbench 컴포넌트에는 확장 포인트가 포함되어 있어 당신의 플러그인이 메뉴 선택과 툴바 버튼을 포함하도록 Eclipse 사용자 인터페이스를 확장시켜 다른 유형의 이벤트 공지를 요청하고 새로운 뷰를 만든다. Workspace 컴포넌트에는 확장 포인트가 포함되어 있어 프로젝트와 파일을 포함하여 리소스들과 인터랙팅 할 수 있다.  Workbench와 Workspace는 다른 플러그인에 의해 확장될 수 있는 Eclipse 컴포넌트 그 이상이다. Debug 컴포넌트로는 플러그인이 프로그램을 시작할 수 있도록 하고 실행 프로그램과 인터랙팅하며 에러를 처리한다. 이 모두는 사실상 디버거를 만드는데 필요한 것들이다. 특정한 애플리케이션에 필요하지만 디버그 컴포넌트는 대부분의 애플리케이션에는 필요하지 않다.  Team 컴포넌트는 Eclipse 리소스로 하여금 버전 콘트롤 시스템(VCS)과 인터랙팅 할 수 있도록 하지만 VCS용 Eclipse 클라이언트를 구현하지 않을 경우에만 그렇다. Team 컴포넌트는 Debug 컴포넌트와 마찬가지로 확장 기능을 갖고있지 않다.  마지막으로 Help 컴포넌트로는 애플리케이션용 온라인 문서와 컨텍스트 관련 도움말을 제공할 수 있다. help 문서화는 전문 애플리케이션의 필수 요소이지만 플러그인 기능에는 꼭 그렇지 않다.  위 컴포넌트가 제공하는 확장 포인트는 Eclipse Platform Help에 문서화 된다. API 레퍼런스의 Workbench 섹션은 처음에는 기가 죽는다. 사용할 수 있는 많은 확장 포인트의 세부사항에 접근하기 보다는 간단한 플러그인과 컴포넌트 부터 접근하도록 하겠다.   |  | | --- | | http://www.ibm.com/i/v14/rules/blue_rule.gif |  |  |  |  | | --- | --- | --- | | http://www.ibm.com/i/c.gif   |  |  | | --- | --- | | http://www.ibm.com/i/v14/icons/u_bold.gif | [**위로**](http://www.ibm.com/developerworks/kr/library/os-ecplug/#main) | |   **플러그인 입문**  플러그인을 만드는 가장 쉬운 방법은 Plug-in Development Environment (PDE)를 사용하는 것이다. PDE는 플러그인을 만들 수 있도록 도와주는 위자드를 제공한다.  **그림 2. "Hello, World" 코드 생성 위자드 선택하기** New plug-in  다음 스크린에서는 추가 정보를 요청한다. 플러그인 이름, 버전 숫자, 공급자 이름, 클래스 이름 등이다. 이들은 플러그인에 대한 중요한 정보들이다. 위자드가 제공한 디폴트를 수락한다. Next를 누른다. 다음 스크린에서 패키지 이름, 클래스 이름, 메시지 텍스트용 디폴트를 수락한다. 체크박스에 "Add the action set to the resource perspective"라고 마킹된 채로 둔다. Finish를 누른다.  위자드가 완료되기 위해서는 다른 특정한 플러그인을 실행해야 한다고 나온다면 OK를 누른다.  **그림 3. PDE 모습: Welcome to Hello Plug-in** PDE perspective  Package Explorer의 왼쪽 워크벤치에 위자드가 만든 것들이 보인다. 프로젝트 클래스 경로에 포함된 많은 .jar 파일들, 툴바 버튼용 그래픽을 포함하고 있는 아이콘 폴더, 자동 구현 스크립트에 의해 사용되는 변수가 포함된 build.properties 파일 등이 있다.  이중 가장 흥미로운 src 폴더에는 플러그인을 위한 소스 코드와 plugin.xml 파일이 들어있다. plugin.xml 부터 살펴보자.   |  | | --- | | http://www.ibm.com/i/v14/rules/blue_rule.gif |  |  |  |  | | --- | --- | --- | | http://www.ibm.com/i/c.gif   |  |  | | --- | --- | | http://www.ibm.com/i/v14/icons/u_bold.gif | [**위로**](http://www.ibm.com/developerworks/kr/library/os-ecplug/#main) | |   **플러그인 적하목록 파일(plug-in manifest file)**  플러그인 manifest 파일인 plugin.xml에는 Eclipse가 플러그인을 프레임웍으로 통합할 때 사용할 정보들이 포함되어 있다. 기본적으로, plugin.xml은 플러그인이 처음 만들어질 때, 적하목록 편집 영역에서 열린다. 에디터의 밑에 있는 탭에서 플러그인에 대한 다양한 정보를 선택할 수 있다. Welcome 탭은 "Welcome to Hello Plug-In" 메시지를 디스플레이하고 사용된 템플릿을 간략히 소개하며 플러그인 구현 팁을 보여준다. "Source" 탭을 선택하면 plugin.xml 파일의 전체 소스를 볼 수 있다.  **Listing 1. 플러그인 적하목록 파일-- 일반 정보**   |  | | --- | | <?xmlversion="1.0" encoding="UTF-8"?>  <plugin  id="com.example.hello"  name="Hello Plug-in"  version="1.0.0"  provider-name="EXAMPLE"  class="com.example.hello.HelloPlugin">  <runtime>  <library name="hello.jar"/>  </runtime> |   다음은 우리의 플러그인이 요청한 플러그인들이 나열된다:  **Listing 2. 플러그인 적하목록 파일 -- required plug-ins**   |  | | --- | | <requires>  <import plugin="org.eclipse.core.resources"/>  <import plugin="org.eclipse.ui"/>  </requires> |   첫째 줄의 org.eclipse.core.resources는 워크스페이스 플러그인이지만 실제로 우리의 플러그인에는 필요하지 않다. 두 번째 플러그인, org.eclipse.ui는 워크벤치이다. 두 개의 확장 포인트를 확장할 것이기 때문에 워크벤치 플러그인이 필요하다.  **그림 4. Resource Perspective 커스터마이징** Resource Perspective  액션 세트에는 두 개의 태그가 포함되어 있다. *menu tag*는 우리의 아이템이 workbench 메뉴 어디에, 어떻게 나타나야 하는지를 설명한다. *action tag*는 무엇을 해야하는지를 설명한다. 특히 이 액션을 수행하는 클래스를 정의한다.  **Listing 3. 액션 세트**   |  | | --- | | <extension  point="org.eclipse.ui.actionSets">  <actionSet  label="Sample Action Set"  visible="true"  id="com.example.hello.actionSet">  <menu  label="Sample &Menu"  id="sampleMenu">  <separator  name="sampleGroup">  </separator>  </menu>  <action  label="&Sample Action"  icon="icons/sample.gif"  class="com.example.hello.actions.SampleAction"  tooltip="Hello, Eclipse world"  menubarPath="sampleMenu/sampleGroup"  toolbarPath="sampleGroup"  id="com.example.hello.actions.SampleAction">  </action>  </actionSet>  </extension> |   두 번째 확장 태그는 Resource Perspective에 플러그인을 추가하기위해 선택된 결과로 생겨났다. 이 태그는 Eclipse가 플러그인을 시작하고 로딩할 때 우리의 플러그인이 Resource Perspective에 추가되도록 한다:  **Listing 4. 확장 태그**   |  | | --- | | <extension  point="org.eclipse.ui.perspectiveExtensions">  <perspectiveExtension  targetID="org.eclipse.ui.resourcePerspective">  <actionSet  id="com.example.hello.actionSet">  </actionSet>  </perspectiveExtension>  </extension>  </plugin> |   이 마지막 확장이 생략되었다면 사용자는 **Window=>Customize Perspective**을 사용하여 Resource (다른) Perspective에 플러그인을 추가시켜야 했을 것이다.   |  | | --- | | http://www.ibm.com/i/v14/rules/blue_rule.gif |  |  |  |  | | --- | --- | --- | | http://www.ibm.com/i/c.gif   |  |  | | --- | --- | | http://www.ibm.com/i/v14/icons/u_bold.gif | [**위로**](http://www.ibm.com/developerworks/kr/library/os-ecplug/#main) | |   **플러그인 소스 코드**  코드 생성 위자드는 두 개의 자바 소스 파일을 만들었다. PDE 패키지 익스플로러의 src 폴더를 열어 볼 수 있다. 첫 번째, HelloPlugin.java는 플러그인 클래스이고 AbstractUIPlugin 추상 클래스를 확장한다. HelloPlugin은 플러그인의 수명을 관리한다. 좀더 확장된 애플리케이션에서는 다이얼로그 박스 세팅과 유저 선호도 같은 것을 관리한다. HelloPlugin은 많은 것을 수행하지 않는다:  **Listing 5. HelloPlugin**   |  | | --- | | packagecom.example.hello.actions;  import org.eclipse.ui.plugin.\*;  import org.eclipse.core.runtime.\*;  import org.eclipse.core.resources.\*;  import java.util.\*;  /\*\*  \* The main plugin class to be used in the desktop.  \*/  public class HelloPlugin extends AbstractUIPlugin {  //The shared instance.  private static HelloPlugin plugin;  //Resource bundle.  private ResourceBundle resourceBundle;    /\*\*  \* The constructor.  \*/  public HelloPlugin(IPluginDescriptor descriptor) {  super(descriptor);  plugin = this;  try {  resourceBundle= ResourceBundle.getBundle(  "com.example.hello.HelloPluginResources");  } catch (MissingResourceException x) {  resourceBundle = null;  }  }  /\*\*  \* Returns the shared instance.  \*/  public static HelloPlugin getDefault() {  return plugin;  }  /\*\*  \* Returns the workspace instance.  \*/  public static IWorkspace getWorkspace() {  return ResourcesPlugin.getWorkspace();  }  /\*\*  \* Returns the string from the plugin's resource bundle,  \* or 'key' if not found.  \*/  public static String getResourceString(String key) {  ResourceBundle bundle= HelloPlugin.getDefault().getResourceBundle();  try {  return bundle.getString(key);  } catch (MissingResourceException e) {  return key;  }  }  /\*\*  \* Returns the plugin's resource bundle,  \*/  public ResourceBundle getResourceBundle() {  return resourceBundle;  }  } |   두 번째 소스 파일인 SampleAction.java에는 적하목록 파일의 액션 세트에 지정된 액션을 수행하는 클래스가 포함되어있다. SampleAction는 IWorkbenchWindowActionDelegate 인터페이스를 구현하는데, 이로서 Eclipse가 플러그인 프록시를 사용하여 정말로 필요하기 전에는 로딩할 필요가 없다. IWorkbenchWindowActionDelegate 인터페이스 메소드는 플러그인이 그 프록시와 인터랙팅 할 수 있도록 한다:  **Listing 6. IWorkbenchWindowActionDelegate 인터페이스 메소드**   |  | | --- | | package com.example.hello.actions;  import org.eclipse.jface.action.IAction;  import org.eclipse.jface.viewers.ISelection;  import org.eclipse.ui.IWorkbenchWindow;  import org.eclipse.ui.IWorkbenchWindowActionDelegate;  import org.eclipse.jface.dialogs.MessageDialog;  /\*\*  \* Our sample action implements workbench action delegate.  \* The action proxy will be created by the workbench and  \* shown in the UI. When the user tries to use the action,  \* this delegate will be created and execution will be  \* delegated to it.  \* @see IWorkbenchWindowActionDelegate  \*/  public class SampleAction implements IWorkbenchWindowActionDelegate {  private IWorkbenchWindow window;  /\*\*  \* The constructor.  \*/  public SampleAction() {  }  /\*\*  \* The action has been activated. The argument of the  \* method represents the 'real' action sitting  \* in the workbench UI.  \* @see IWorkbenchWindowActionDelegate#run  \*/  public void run(IAction action) {  MessageDialog.openInformation(  window.getShell(),  "Hello Plug-in",  "Hello, Eclipse world");  }  /\*\*  \* Selection in the workbench has been changed. We  \* can change the state of the 'real' action here  \* if we want, but this can only happen after  \* the delegate has been created.  \* @see IWorkbenchWindowActionDelegate#selectionChanged  \*/  public void selectionChanged(IAction action, ISelection selection) {  }  /\*\*  \* We can use this method to dispose of any system  \* resources we previously allocated.  \* @see IWorkbenchWindowActionDelegate#dispose  \*/  public void dispose() {  }  /\*\*  \* We will cache window object in order to  \* be able to provide parent shell for the message dialog.  \* @see IWorkbenchWindowActionDelegate#init  \*/  public void init(IWorkbenchWindow window) {  this.window = window;  }  } |  |  | | --- | | http://www.ibm.com/i/v14/rules/blue_rule.gif |  |  |  |  | | --- | --- | --- | | http://www.ibm.com/i/c.gif   |  |  | | --- | --- | | http://www.ibm.com/i/v14/icons/u_bold.gif | [**위로**](http://www.ibm.com/developerworks/kr/library/os-ecplug/#main) | |   **플러그인 실행과 디버그**  Eclipse용 플러그인을 개발할 때 Eclipse를 끄고 테스트와 디버그를 위해 새로운 플러그인으로 재시작하는 것은 이상하다. Eclipse의 PDE는 다행히도 자가 호스팅 개발 환경을 제공하고 있기 때문에 워크벤치의 개별 인스턴스에 설치하지 않고 플러그인을 실행할 수 있다.  Hello 플러그인을 실행하려면, **Run=>Run As=>Run-time Workbench**를 선택하고 추가된 플러그인 메뉴 선택과 툴바가 있는 Workbench의 또 다른 인스턴스를 시작한다. (그림 5)  **그림 5. 런타임 워크벤치에서 실행되는 Hello 플러그인** Hello plug-in  툴바 버튼을 클릭하거나 "Sample Menu" 메뉴에서 플러그인을 활성화시킬 수 있다. 그렇게 하면 "Hello Plug-in" 타이틀과 "Hello, Eclipse world" 내용이 나타나면 OK 버튼을 누른다.  **Run=>Debug As=>Run-time Workbench**를 선택하여 비슷한 방법으로 플러그인을 디버그할 수 있다.  플러그인 테스트가 완료되고 배포 준비를 갖췄다면 패키징 과정에 돌입한다.   |  | | --- | | http://www.ibm.com/i/v14/rules/blue_rule.gif |  |  |  |  | | --- | --- | --- | | http://www.ibm.com/i/c.gif   |  |  | | --- | --- | | http://www.ibm.com/i/v14/icons/u_bold.gif | [**위로**](http://www.ibm.com/developerworks/kr/library/os-ecplug/#main) | |   **플러그인 패키징**  Eclipse는 어떤 플러그인이 로딩 될 것인지를 시작할 때 플러그인 디렉토리를 보면서 결정한다. 플러그인을 설치하기 위해서는 플러그인 디렉토리에 하위 디렉토리를 만들어서 프로그램 파일과 적하목록 파일에 복사한다. 이것은 필수적인 일은 아니지만 디렉토리 이름이 플러그인 ID를 나타내게 하고 그 다음에 언더바를 하고 버전 숫자를 추가한다. Eclipse가 C:\eclipse에 설치되면 디렉토리를 만든다:   |  | | --- | | <runtime>  <library name="hello.jar"/>  </runtime> |   지금까지가 플러그인을 설치하는데 필요한 전부이다. 하지만 새로운 플러그인이 인식되도록 하려면 Eclipse를 정지하고 재시작해야한다.   |  | | --- | | http://www.ibm.com/i/v14/rules/blue_rule.gif |  |  |  |  | | --- | --- | --- | | http://www.ibm.com/i/c.gif   |  |  | | --- | --- | | http://www.ibm.com/i/v14/icons/u_bold.gif | [**위로**](http://www.ibm.com/developerworks/kr/library/os-ecplug/#main) | |   **플러그인 버전 업데이트**  디렉토리 이름이 버전 넘버를 추가하는 목적은 여러 버전의 플러그인이 같은 머신에 공존하도록 하기 위함이다. Hello 플러그인의 업데이트 버전을 만들어서 어떻게 작동하는지를 볼 수 있다.   |  | | --- | | http://www.ibm.com/i/v14/rules/blue_rule.gif |  |  |  |  | | --- | --- | --- | | http://www.ibm.com/i/c.gif   |  |  | | --- | --- | | http://www.ibm.com/i/v14/icons/u_bold.gif | [**위로**](http://www.ibm.com/developerworks/kr/library/os-ecplug/#main) | |   **플러그인 조각(fragment)과 특성(feature)**  Eclipse는 플러그인들로 구성되어있지만 Eclipse용 플러그인을 개발할 때 고려해야하는 두 가지 다른 레벨의 컴포넌트가 있다.  플러그인 *조각(fragment)*은 완벽한 플러그인(타겟 플러그인)의 부분을 형성한다. 조각들이 제공하는 기능들은 타겟 플러그인의 기능에 합병된다. 조각은 다른 언어를 위해 플러그인을 로컬라이징하는데 사용될 수 있다. 완전히 새로운 배포판을 만들 필요 없이 기존 플러그인에 기능을 추가할 수 있다. 조각은 플러그인과 동일하다. 주요 차이점은 조각은 플러그인 클래스를 갖지 않는다는 점이다. 조각의 수명 주기는 타겟 플러그인에 의해 관리된다. 또한 조각의 적하목록 파일은 타겟 플러그인의 ID와 버전 넘버, 조각의 ID와 버전 넘버를 리스팅한다.  플러그인 *특징(feature)*은 코딩을 전혀 포함하지 않는다. Eclipse 아키텍쳐 용어에서 특징(feature)은 전체 제품으로 관련 플러그인들의 그룹을 패키징하는 것이다. 예를 들어, JDT는 자바 에디터, 디버거, 콘솔로 구성되었다. 적하목록 파일(feature.xml)은 특징 아카이브를 설명한다. |