Version 1.0

Extending iGROM

A tutorial

Mikołaj Koziarkiewicz

Purpose of this document and intended audience

This document is to be used as an informational aid to those who wish to add support to an additional language, using iGROM as the base framework. It contains all the necessary information to replicate the level of functionality that the DLV editors currently have.

It is intended for Java programmers who now the basics of Eclipse in its OSGi platform aspect. Additionally, knowledge of parser and lexer construction is required, since the core element of every language extension of iGROM is a source code scanner. In this regard, knowledge of the ANTLR framework is the most helpful due to the already-developed infrastructure, but any other lexer/parser construction scheme can be used.

Introduction

iGROM, apart from being a suite of tools for answer set programming developers, additionally allows itself to be extended in order to support additional languages and/or dialects. This mechanism is a further abstraction of the DLTK (Dynamic Languages ToolKit) library, and, although the current version does not unfortunately eliminate all the boiler code and XML that is needed to extend the language support, it nevertheless reduces the amount of work considerably (especially in the case of Java code).

Since the API for language extension may change in future releases, and since it is an abstraction of an abstraction (DLTK itself is a facade to the basic Eclipse editor services), instead of describing the functionality in a top-down fashion, this document will instead show how to extend the functionality of iGROM in the form of a tutorial. This will reduce the amount of information given to the bare minimum needed to develop an editor for an additional language.

Keep in mind that this tutorial assumes you have some knowledge of the Eclipse architecture, what plugins are, etc. This tutorial also only focuses on the creation of plugins; infrastructure code such as features, RCP templates and update sites are not covered - you can use the project's sources as an example though.

Preliminaries

It is strongly recommended that you first decide on a package prefix and a language suffix. They will be used both for Java package names, and for IDs in the Eclipse extension framework. For the purpose of this document, the package prefix is denoted as *(package)*, and the language suffix as *(language)* . In the names of classes and strings, *(Language)* is the full name of your language.

Having done that, please create 3 plugins per language/dialect, in the form:

(package).core.*(language)*
(package).ui.*(language)*
(package).ui.launching .*(language)*

The following sections describe the data that needs to be included in each of the plugins: configuration, classes, and optional graphical elements.

Core facilities - *(package)*.core.*(language)*

The core plugin should include the following dependencies in MANIFEST.MF

```
Require-Bundle: org.eclipse.ui,
org.eclipse.core.runtime,
org.eclipse.dltk.core;bundle-version="1.0.0",
org.eclipse.dltk.ui;bundle-version="1.0.0",
org.eclipse.core.resources;bundle-version>="3.5.0",
org.eclipse.ui.editors;bundle-version>="3.5.0",
org.eclipse.ui.ide;bundle-version>="3.5.0",
org.eclipse.dltk.launching;bundle-version="1.0.0",
org.eclipse.dltk.debug;bundle-version="1.0.0",
org.eclipse.dltk.debug.ui;bundle-version="1.0.0",
org.eclipse.debug.core;bundle-version>="3.5.0",
org.eclipse.debug.ui;bundle-version>="3.5.0",
org.eclipse.jface.text;bundle-version>="3.5.0",
org.deved.antlride.antlr;bundle-version>="3.2.0", (optional, for ANTLR support)
at.ac.tuwien.kr.igrom.core.asp;bundle-version="1.0.0"
```

The plugin.xml should contain the following extensions:

```
<extension
id="(package).core.(language).nature"
point="org.eclipse.core.resources.natures">
<runtime>
<run
class="at.ac.tuwien.kr.igrom.core.asp.content.ASPNature">
</run>
</runtime>
</extension>
<extension
point="org.eclipse.dltk.core.sourceElementParsers">
<parser
class="(package).core.(language).(Language)SourceElementParser"
nature="(package).core.(language).nature"
priority="0">
</parser>
</extension>
<extension
point="org.eclipse.dltk.core.sourceParsers">
<parserContribution
natureId="(package).core.(language).nature">
<parser
class="at.ac.tuwien.kr.igrom.core.parser.common.ParserFactory:(package).core.(language).nature"
description="(Language) Parser"
id="(package).(language).parser"
name="(language).parser"
priority="0">
</parser>
</parserContribution>
</extension>
<extension
point="org.eclipse.dltk.core.buildParticipant">
<buildParticipant
class="org.eclipse.dltk.core.builder.ParserBuildParticipantFactory"
id="(package).(language).buildParticipant"
name="Basic (Language) Build Participant"
nature="(package).core.(language).nature">
</buildParticipant>
</extension>
<extension
point="org.eclipse.core.contenttype.contentTypes">
<content-type
base-type="org.eclipse.core.runtime.text"
```

```
describer="at.ac.tuwien.kr.igrom.core.asp.content.EmptyContentDescriber"
file-extensions="(language)"
id="(package).core.(language).contentType"
name="(Language) Files"
priority="normal">
</content-type>
</extension>
<extension
id="(package).(language).language"
name="(Language) Language"
point="org.eclipse.dltk.core.language">
<language
class="at.ac.tuwien.kr.igrom.core.asp.content.ASPLanguageToolkit"
nature="(package).core.(language).nature"
priority="0">
</language>
</extension>
<extension
point="at.ac.tuwien.kr.igrom.core.asp.language">
<language
fullName="(Language)"
languageID="(package).core.(language)"
textTools="(package).core.(language).(Language)TextTools">
</language>
</extension>
```

In addition, you need to implement the following classes:

- *(Language)*`SourceElementParser` – parser class, subclass `ASPSourceElementParser` (see http://wiki.eclipse.org/DLTK_IDE_Guide:Step_2._Towards_an_Editor for more details).
    - Alternatively, if you're using ANTLR, you can use `AbstractANTLRSourceElementParser` as your base class. In this case, the parser should implement `IANTLRParser` (and, if that's good for you, can subclass `AbstractANTLRParser` instead).
- *(Language)*`TextTools` – if you are not using ANTLR, subclass `ASPTextTools`, and implement `createScanner()` and `createDocumentPartitioner()` . The former requires you to write a subclass of `ASPScriptScanner`, which is basically a wrapper for your lexer - see the `BasicANTLRLexerScanner` for an example of how to do that. If you are using ANTLR, then just subclass `ASPANTLRTextTools` and implement `createLexer()`.

UI Main - *(package)*.ui.*(language)*

The manifest file should include the following:

```
Require-Bundle: org.eclipse.ui,
org.eclipse.core.runtime,
org.eclipse.dltk.core;bundle-version="1.0.0",
org.eclipse.dltk.ui;bundle-version="1.0.0",
org.eclipse.core.resources;bundle-version>="3.5.0",
org.eclipse.ui.editors;bundle-version>="3.5.0",
org.eclipse.ui.ide;bundle-version>="3.5.0",
org.eclipse.dltk.launching;bundle-version="1.0.0",
org.eclipse.dltk.debug;bundle-version="1.0.0",
org.eclipse.dltk.debug.ui;bundle-version="1.0.0",
org.eclipse.debug.core;bundle-version>="3.5.0",
org.eclipse.debug.ui;bundle-version>="3.5.0",
org.eclipse.jface.text;bundle-version>="3.5.0",
org.deved.antlride.antlr;bundle-version>="3.2.0",
at.ac.tuwien.kr.igrom.core.asp;bundle-version="1.0.0",
(package).core.(language);bundle-version="1.0.0",
at.ac.tuwien.kr.igrom.ui.asp;bundle-version="1.0.0"
```

The extension file should have the following form:

```
<plugin>
<extension
point="org.eclipse.dltk.ui.language">
<language
class="at.ac.tuwien.kr.igrom.ui.asp.ASPUILanguageToolkit"
nature="(package).core.(language).nature"
priority="0">
</language>
</extension>
<extension
point="org.eclipse.ui.newWizards">
<wizard
category="(package).(language).wizardCategory"
class="at.ac.tuwien.kr.igrom.ui.asp.wizards.NewProjectWizard:(package).core.(language).nature"
finalPerspective="(package).perspective"
icon="icons/project_new.png"
id="(package).(language).newprojectwizard"
name="New (Language) Project"
project="true">
</wizard>
<wizard
category="(package).(language).wizardCategory"
class="at.ac.tuwien.kr.igrom.ui.asp.wizards.NewProgramWizard:(package).core.(language).nature"
finalPerspective="(package).perspective"
id="(package).(language).newProgramWizard"
icon="icons/file_new.png"
name="New (Language) Program"
project="false">
</wizard>
<category
id="(package).(language).wizardCategory"
name="(Language)"
parentCategory="(package).category">
</category>
</extension>
<extension
point="org.eclipse.ui.preferencePages">
<page
category="at.ac.tuwien.kr.igrom.ui.asp.preferences.main"
class="at.ac.tuwien.kr.igrom.ui.asp.preferences.PreferenceCategoryPage"
id="(package).core.(language).nature.preferences"
```

```
name="(Language)">
</page>
<page
category="(package).core.(language).nature.preferences"
class="at.ac.tuwien.kr.igrom.ui.asp.preferences.interpreter.ASPInterpreterPreferencePage:
(package).core.(language).nature"
id="(package).core.(language).nature.preferences.interpreter"
name="Interpreters">
</page>
<page
category="(package).core.(language).nature.preferences"
class="(package).ui.(language).preferences.(Language)SyntaxHighlightPreferencePage"
id="(package).(language).syntaxHighlight"
name="Syntax Highlighting">
</page>
</extension>
<extension
point="org.eclipse.ui.editors">
<editor
class="(package).ui.(language).(Language)Editor"
default="false"
icon="icons/file.png"
id="(package).(language).editor"
name="(Language) Editor">
<contentTypeBinding
contentTypeId="(package).core.(language).contentType">
</contentTypeBinding>
</editor>
</extension>
<extension
point="org.eclipse.dltk.launching.interpreterInstallTypes">
<interpreterInstallType
class="(package).ui.(language).preferences.(Language)InterpreterInstallType:(package).core.
(language).nature"
id="(package).(language)InterpreterInstallType">
</interpreterInstallType>
</extension>
<extension point="org.eclipse.ui.actionSetPartAssociations">
<actionSetPartAssociation targetID="org.eclipse.debug.ui.launchActionSet">
<part id="(package).(language).editor" />
</actionSetPartAssociation>
</extension>
<extension
point="org.eclipse.ui.decorators">
<decorator
adaptable="true"
location="TOP_RIGHT"
lightweight="true"
label="Project Overlay"
state="true"
class="org.eclipse.dltk.ui.DeclarativeLightweightLabelDecorator:/icons/project.png"
id="(package).core.(language).decorators.projectdecorator">
<enablement>
<and>
<objectClass name="org.eclipse.core.resources.IProject"/>
<objectState
name="nature"
value="(package).core.(language).nature">
</objectState>
</and>
</enablement>
</decorator>
</extension>
```

The following classes should be implemented:

- *(package)*`.ui.`*(language)*`.`*(Language)*`Editor` – extend `ASPEditor` and implement `getNatureId()`.
- *(package)*`.ui.`*(language)*`.preferences.`*(Language)*`InterpreterInstallType` – use `ASPInterpreterInstallType` as a base class. See `DLVInterpreterInstallType` for an example implementation.

- *(package)*.ui.*(language)*.preferences.*(Language)*SyntaxHighlightPreferencePage - subclass ASPSyntaxHighlightPreferencePage. Currently all syntax highlighting are in the same storage (you can customise what a given token "means" via your lexer implementation, see AbstractANTLRLexer for an example).

Additionally, you can add several icons to improve user experience by specifying the “icon” attribute in several extensions. Additionally, there is a requirement to add two icons for the wizards. Firstly, here are the icons settable via plugin.xml :

- icon="icons/project_new.png" - icon for the “New *(Language)* Project” menu option. The icons is 16x16.
- icon="icons/file_new.png" - icon for the “New *(Language)* Program” menu option. The icon is 16x16.
- icon="icons/file.png" - the editor icon, which controls the icon in the editor tab, and the icons for the program files in the Explorer. Unlike the other plugin.xml paths, this one is **mandatory** – your plugin won't start without it (a quirk in the Eclipse platform). Note that this path does not have to point to an actual file, however. The icon should be 16x16.
- /icons/project.png - the decorator for the project icons. Decorators, as the name says, are additional icons overlaid on top of the “folder” graphic denoting a project. For example, for DLV, it is the “D”, for diagnosis, it is the “component symbol” etc. This icon should have a size **no greater than** 16x16 due to it's overlay nature.

The two icons that are **mandatory files** should have the following paths:
- icons/newFileWizard.png,
- icons/newProjectWizard.png.

They both should be the same size (75x66) and represent the icons in the upper-right-hand corner, of, respectively, the new program wizard, and the new project wizard.

UI Launching - *(package)*.ui.launching .*(language)*

The manifest file should include the following:

```
Require-Bundle: org.eclipse.ui,
org.eclipse.core.runtime,
org.eclipse.dltk.core;bundle-version="1.0.0",
org.eclipse.dltk.ui;bundle-version="1.0.0",
org.eclipse.core.resources;bundle-version>="3.5.0",
org.eclipse.ui.editors;bundle-version>="3.5.0",
org.eclipse.ui.ide;bundle-version>="3.5.0",
org.eclipse.dltk.launching;bundle-version="1.0.0",
org.eclipse.dltk.debug;bundle-version="1.0.0",
org.eclipse.dltk.debug.ui;bundle-version="1.0.0",
org.eclipse.debug.core;bundle-version>="3.5.0",
org.eclipse.debug.ui;bundle-version>="3.5.0",
org.eclipse.jface.text;bundle-version>="3.5.0",
org.deved.antlride.antlr;bundle-version>="3.2.0",
at.ac.tuwien.kr.igrom.core.asp;bundle-version="1.0.0",
(package).core.(language);bundle-version="1.0.0",
at.ac.tuwien.kr.igrom.ui.asp;bundle-version="1.0.0",
(package).ui.(language);bundle-version="1.0.0",
at.ac.tuwien.kr.igrom.ui.launching.asp;bundle-version="1.0.0"
```

The extension file should have the following form:

```
<extension
point="org.eclipse.debug.core.sourcePathComputers">
<sourcePathComputer
class="org.eclipse.dltk.launching.sourcelookup.ScriptSourcePathComputer"
id="(package).(language).sourcePathComputer">
</sourcePathComputer>
</extension>
<extension
point="org.eclipse.debug.core.sourceLocators">
<sourceLocator
class="org.eclipse.dltk.launching.sourcelookup.ScriptSourceLookupDirector"
id="at.ac.tuwien.kr.igrom.(language).sourceLocator"
name="(Language) Source Locator">
</sourceLocator>
</extension>
<extension
point="org.eclipse.debug.core.launchConfigurationTypes">
<launchConfigurationType
delegate="(package).ui.launching.(language).(Language)ScriptLaunchConfigurationDelegate:
(package).core.(language).nature"
id="(package).core.(language).launchConfigurationType"
modes="run"
name="(Language) Program"
public="true"
sourceLocatorId="at.ac.tuwien.kr.igrom.(language).sourceLocator"
sourcePathComputerId="(package).(language).sourcePathComputer">
</launchConfigurationType>
</extension>
<extension
point="org.eclipse.debug.ui.launchConfigurationTabGroups">
<launchConfigurationTabGroup
class="at.ac.tuwien.kr.igrom.ui.launching.asp.tabs.ASPLaunchConfigurationTabGroup:(package).core.
(language).nature"
id="(package).(language).launchConfigurationTabGroup"
type="(package).core.(language).launchConfigurationType">
</launchConfigurationTabGroup>
</extension>
<extension
```

```
point="org.eclipse.debug.ui.launchShortcuts">
<shortcut
class="at.ac.tuwien.kr.igrom.ui.launching.asp.LaunchShortcut:(package).core.(language).nature"
description="Run as a (Language) Program"
id="(package).(language).launchShortcut"
label="(Language) Program"
icon="icons/program.png"
modes="run">
<contextualLaunch>
<enablement>
<with
variable="selection">
<iterate
ifEmpty="false">
<or>
<test
args="(package).core.(language).nature"
forcePluginActivation="false"
property="org.eclipse.dltk.launching.hasProjectNature">
</test>
</or>
</iterate>
</with>
</enablement>
</contextualLaunch>
</shortcut>
</extension>
<extension point="org.eclipse.debug.ui.launchConfigurationTypeImages">
<launchConfigurationTypeImage
icon="icons/program.png"
configTypeID="(package).core.(language).launchConfigurationType"
id="(package).core.(language).launchConfigurationTypeImage">
</launchConfigurationTypeImage>
</extension>
```

You only need to implement one class in this case, namely:

- *(package)*`.ui.launching.`*(language)*`.`*(Language)*`ScriptLaunchConfigurationDelegate` – override `createInterpreterConfig()` to set any custom parameters you have here – but don't forget to call `super()` first! See e.g. `DLVKScriptLaunchConfigurationDelegate` for an example how this could be done.

Optionally, you can add a tab to you launch configuration, the data of which will be used by the delegate . See `DLVKLaunchConfigurationTabGroup` and `PlanningParametersTab` for an example. If you also want any launch configuration custom code for your launch shortcut (the ones that appear in the context menu), you need to subclass `LaunchShortcut` as well.

This plugin may also be enhanced by icons, or, more specifically, by a single icon. This icon appears both in the the "Run as..." context menu, an in the Launch Configurations window. It is per default identified as `icon="icons/program.png"` (note that this appears in **two** extensions). The icon size should be 16x16.