Introducing Eclipse Plug-ins 1

Contents

Eclipse Plug-ins

- Eclipse isn't a single monolithic program, but rather a small kernel called a plug-in loader surrounded by plug-ins.
- Each plug-in may rely on services provided by another plug-in, and each may in turn provide services on which yet other plug-ins may rely.

Eclipse Plug-ins

- The behavior of every plug-in is in code, yet the dependencies and services of a plug-in are declared in the MANIFEST.MF and plugin.xml files.
- This facilitates lazy-loading of plug-in code on an as-needed basis, thus reducing the startup time and the memory footprint.

Eclipse Plug-ins

- On startup, the plug-in loader scans the MANIFEST.MF and plugin.xml files for each plug-in and then builds a structure containing this information. But, until required, it doesn't load all the code from a plug-in.

OSGi

- Eclipse originally used a home-grown runtime model/mechanism that was designed and implemented specifically for Eclipse.
- This was good because it was highly optimized and tailored to Eclipse.
- But less than optimal because having a unique runtime mechanism prevented reusing the work done in other areas (e.g., OSGi, Avalon, JMX, etc.).

OSGi

- As of Eclipse 3.0, a new runtime layer was introduced based upon technology from the OSGi Alliance (www.osgi.org).
- It has a strong specification, a good component model, supports dynamic behaviour, and is reasonably similar to the original Eclipse runtime.
- With each new release of Eclipse, the Eclipse runtime API and implementation (e.g. "plug-ins") continues to align itself more and more closely with the OSGi runtime model (e.g. "bundles").

Plug-in State Information

- The plug-in state information located in the workspace directory hierarchy is associated only with that workspace, yet the Eclipse IDE, its plug-ins, the plug-in static resources and plug-in configuration files are shared by multiple workspaces.

Plug-in Structure

- The HelloPlugin plug-in directory (or JAR file) contains files similar to a typical plug-in, including *.jar files containing code, various images used by the plug-in, and the plug-in manifest.
- List of files: helloplugin.jar, icons, META-INF/MANIFEST.MF, plugin.xml.

Plug-in Structure

- helloplugin.jar: A file containing the actual Java classes comprising the plug-in. Typically, the JAR file is named for the last segment in the plug-in's identifier, but it could have any name, as long as that name is declared in the META-INF/MANIFEST.MF file.

Plug-in Structure

- icons: Image files are typically placed in an icons or images subdirectory and referenced in the plugin.xml and by the plug-in's various classes. Image files and other static resource files that are shipped as part of the plug-in can be accessed using methods in the plug-in class.

Plug-in Structure

- META-INF/MANIFEST.MF: A file describing the runtime aspects of the plug-in such as identifier, version, and plug-in dependencies.
- plugin.xml: A file in XML format describing extensions and extension points.

Plug-in Deployment

- The plug-in directory must be located in the plugins directory as a sibling to all the other Eclipse plug-ins.
- As of Eclipse 3.1, the plug-in can be delivered as a single JAR file containing the same files as a plug-in directory.