HRE Plugin Implementation at Build 32

[1. PREFACE 2](#_Toc188617675)

[2. PLUGIN DEVELOPMENT 2](#_Toc188617676)

[3. PLUGIN BASICS 2](#_Toc188617677)

[4. PLUGIN DEVELOPMENT MODES 3](#_Toc188617678)

[5. PLUGIN STATES 3](#_Toc188617679)

[6. HRE PLUGIN FRAMEWORK 4](#_Toc188617680)

[7. PLUGIN FOLDERS 5](#_Toc188617681)

[8. PLUGIN PREPARATION AND USE OF PLUGINS 5](#_Toc188617682)

[9. ENABLING/DISABLING PLUGINS 6](#_Toc188617683)

[10. PLUGIN EXAMPLE CODE 6](#_Toc188617684)

[11. LIBRARIES FOR PF4J 7](#_Toc188617685)

[12. PLUGIN DUMMY CLASS Javadocs 8](#_Toc188617686)

**Revision History**

|  |  |  |
| --- | --- | --- |
| 2021-08-28 | Nils Tolleshaug | Initial draft |
| 2021-08-31 | Don Ferguson | Minor updates and re-arrangement of sections; Javadocs embedded |
| 2021-09-02 | Nils Tolleshaug | Added 8. PLUGIN PREPARATION AND USE OF PLUGINS |
| 2021-09-03 | Don Ferguson | Minor edits of Section 8 and Javadocs, added Section 9 |
| 2021-09-04 | Don Ferguson | Corrections in Enabling/Disabling Plugins; expanded Manifest detail |
| 2022-12-26 | Don Ferguson | Re-tested at Build 28 with updated pf4j library JARs |
| 2025-01-24 | Don Ferguson | Re-tested at Build 32; pf4j libraries updated to 2024 versions |

## PREFACE

A Plugin is a piece of Java code made to extend the functionality of HRE without being programmed into the HRE system. The Plugin may appear in the HRE main menu system either as an additional main menu item or as a menu point added to one of the existing HRE menu items. A typical example is the addition of new reports under the main Report menu.

A Plugin is identified by the PluginManager and linked into a running HRE system. Plugin code consists of class files that are loaded as ordinary class files to be executed by the Java Virtual Machine. The Plugin code needs to be developed according to defined rules in order to be able to interact with the HRE code. Plugins can be downloaded from a network repository and can extend the functions of HRE without being distributed with the HRE program.

The advantage of using Plugins is to allow other programmers to extend the functionality of HRE without being part of the HRE development team and without needing to know details of the internals of HRE code.

## A screenshot of a computer AI-generated content may be incorrect.PLUGIN DEVELOPMENT

An Integrated Development environment (IDE) is needed for plugin development. We recommend use of the Eclipse IDE as HRE is developed under Eclipse and the examples in this document are based on Eclipse.

To start plugin development a “Plugin” project should be created in the Eclipse IDE. The Plugin project initially shall be loaded with a pre-programmed framework package folder with source files. The plugin Java package used is the open-source PF4J Plugin Framework for Java (see <https://github.com/pf4j> with additional documentation at <https://pf4j.org/> ). The PF4J package uses a set of Java libraries which are set out later in this document. The pre-programmed plugin project also contains a Java main class that can be run to start compiling the programmed plugin. The figure at left shows the folder structure for an HRE Plugin project with a number of plugins developed and stored in hre.pf4j, one of which also has NLS files under hre.nls.

## PLUGIN BASICS

Plugin administration is controlled by a PluginManager (PM) called from HG0401HREMain in HRE. The PM searches the Plugin folder and identifies the possible plugins in the folder. The PM operates as a Java Class Loader that loads the plugin class files into the Java Virtual Machine (JVM) that runs/executes the HRE program. The program link between the plugin and HRE is programmed as a Java Interface construction that allow two classes to be interconnected without being known to each other and then imported as classes in HRE.

## PLUGIN DEVELOPMENT MODES

There are three possible levels of plugin API's.

1. Provide a Plugin API as library .jar file to the HRE Plugin project. The programmer must use the set of fixed API's provided in the jar. Methods are only available through the pointers passed over by MainMenuExtensionPoint from HRE Main at runtime and the programmer can only use methods in HRE available through the library. Typical use will be to allow the plugin to access and process data from the HRE H2 database of an open project
2. Provide API's as source code for the dummy classes. Methods are only available through the pointers passed over by MainMenuExtensionPoint from HRE Main runtime. The plugin programmer can (based on knowledge of HRE class and method program structure) define additional dummy classes and methods to use functionality in HRE. Typical use is programming of a report plugin where the plugin reuses methods in HRE, for example, name configuration, date format methods and place name configuration methods
3. A programmer of plugins has access to HRE source and can extend HRE functionality with their own MenuExtensionPoints and own plugins. To execute at this level, the base HRE code must be modified and recompiled to include the programmer’s changes to support their plugins. Only to be used by skilled HRE programmers.

## PLUGIN STATES

Each plugin passes through a pre-defined set of states controlled by method calls from the PM. The class PluginState in PF4J defines all possible states. The primary plugin states are:

* CREATED - found by PM in the plugin folder; the plugin path and descriptor are known
* DISABLED - set disabled by PM or in plugin configuration; the plugin cannot be used   
  (see files ‘disabled.txt’ and ‘enabled.txt’ in Section 9)
* RESOLVED – the plugin is created, all dependencies are created; plugin can be started
* STARTED – {@link Plugin#start()} has executed
* STOPPED – {@link Plugin#stop()} has executed
* FAILED – the plugin failed to start.

## HRE PLUGIN FRAMEWORK

The standard interface methods currently implemented at HRE Build 32 are:

void setMainPane(JDesktopPane mainPane);

Transfers a pointer to the main HRE window to be able to position the plugin window in relation to the HRE window

void setNationalLanguage(String nls);

Transfers the NLS language code to the plugin to setup language translation in the plugin

String className();

Returns the Plugin extension call name to HRE

void setMainPointer(HG0401HREMain mainFrame);

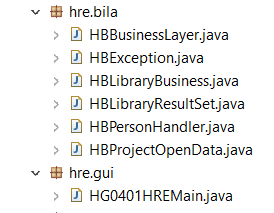
Transfers a pointer to the main HRE class to access methods for the open project.

void buildMenuBar(JMenuBar menuBar);

Transfers a pointer to the HRE menu bar so that the plugin can create new menu items in HRE

void actionPerformed(ActionEvent actionEvent)

**Dummy classes to identify the classes and methods used in HRE:** dummy classes are used by the compiler to establish runtime access from the plugin to classes and methods in HRE.

The link to HRE from Plugin is established by passing a pointer to: HBBusinessLayer[] pointBusinessLayers in HRE main. The array elements contain pointers to each Business Layer Manager class in HRE and give access to commonly used methods e.g., for data collection and presentation in HRE.

A screenshot of a computer program

AI-generated content may be incorrect.

The figure above lists the dummy classes currently defined for use by Plugin programmers.

The figure to right shows the dummy methods programmed into a plugin to access HRE HBBusinessLayer methods.

More detailed documentation can be found in the Plugin Project Javadocs embedded at the end of this document.

## PLUGIN FOLDERS

Each plugin must be loaded into the defined HRE Plugins folder as either a JAR or ZIP file with a structure like the following example (a plugin providing an extension of the HRE Report menu) - the PM unpacks this to extract the class files when executing the plugin:

**pf4j-nls-demo-menu-report.zip** (example user defined zip name)

**classes**

**hre**

**nls** (Natural Language Support files)

**pf4j** (plugin class files for execution)

**META-INF**

extensions.idx (see below)  
MANIFEST.MF (see below)

In the above example, the nls folder contained Message.java and language property files for NLS.

The pf4j folder contained:

MenuPluginReport.class – extending Plugin.class  
PersonReportMenuExtension.class – plugin code extending ReportMenuExtensionPoint  
ReportMenuExtensionPoint.class - HRE interface class extending Extension.class.

The extension.idx text file contains:

# Generated by PF4J   
hre.pf4j.PersonReportMenuExtension

The MANIFEST.MF text file must contain 2 mandatory entries and optional entries:

Plugin-Id: menu-plugin-report (ID of plugin; mandatory)  
Plugin-Version: 0.0.1 (Plugin version; mandatory)

Plugin-Provider: Nils Tolleshaug (Author; optional)  
Plugin-Class: hre.pf4j.MenuPluginReport (Plugin class name; optional)  
Plugin-Requires: n.n.n (Required app version; optional)  
Plugin-Dependencies: x, y, z (Other plugin dependencies; optional)  
Plugin-Description: simple description (Plugin description; optional)  
Plugin-License: license type (Plugin license; optional)

## PLUGIN PREPARATION AND USE OF PLUGINS

The pf4j folder of the Eclipse “Plugin” project will contain java .class files compiled and produced when the HRE plugin is developed. To distribute a Plugin, the **class** files and associated other files are zipped together as explained in Section 7 and the zipped folder is given a user selected file name that describes the Plugin.

At HRE execution time, the PluginManager (PM) called from HRE Main looks for available plugins in the Plugins folder located at C:/Users/user-name/HRE/Plugins/ (for Windows). Each plugin is stored in a separate zip file.

To use a Plugin the plugin zip file is copied to the Plugins folder of an HRE installation. Upon opening a new project in HRE, the PM looks for new plugins and if a Plugin zip file is found in the Plugins folder, the PM unzips the zip file and examines the content to see if the unzipped file structure contains a valid Plugin implementation. If a valid plugin (Extension) is found, the code is activated and the plugin’s menu items will appear in the HRE menu structure, e.g., under the Report menu item.

In future releases of HRE one can foresee that developed, tested and validated plugins could be made available for download on the HRE website along with the HRE program install packages.

High use and/or high-value plugin code could then also be merged into base HRE code if this is deemed desirable.

## ENABLING/DISABLING PLUGINS

Currently, a Setting within HRE can be used to turn off or on the use of all Plugins. This could be expanded later to allow HRE to show a list of all available plugins in the Plugins folder and then allow activation/deactivation of plugins on an individual basis.

There is a currently available method for enabling/disabling plugins on an individual basis. This requires creating a simple file ‘enabled.txt’ or ‘disabled.txt’ in the Plugins folder of an HRE installation.

Example content for ‘enabled.txt’ is:

########################################

# - load only these plugins

# - add one Plugin-Id on each line

# - put this file in Plugins folder

########################################

menu-plugin-report

Example content for ‘disabled.txt’ is:

########################################

# - ignore these plugins

# - add one Plugin-Id on each line

# - put this file in Plugins folder

########################################

menu-plugin-report

The PluginManager reads these files (if they exist) and ignores all comment lines (lines that start with # character). It then processes the files according to their contents. The plugin name in these text files must EXACTLY match the Plugin-Id field in the plugin’s MANIFEST.MF file.

**NOTE**: if a file called ‘enabled.txt’ exists then ‘disabled.txt’ is ignored.

## PLUGIN EXAMPLE CODE

The following is the code for MainMenuExtensionPoint with the basic ExtensionPoint methods to be called declared in the Extension handler and called from HRE Main:

package hre.pf4j.ext;

import javax.swing.JDesktopPane;

import javax.swing.JMenuBar;

import org.pf4j.Extension;

import org.pf4j.ExtensionPoint;

import hre.gui.HG0401HREMain;

@Extension

public interface MainMenuExtensionPoint extends ExtensionPoint {

String className();

void buildMenuBar(JMenuBar menuBar);

void setMainPane(JDesktopPane mainPane);

void setMainPointer(HG0401HREMain mainFrame);

void setNationalLanguage(String nativeLanguage);

}

The following is the code for MenuPluginReport with the basic start, stop and delete methods called from PluginManager:

package hre.pf4j;

import org.apache.commons.lang3.StringUtils;

import org.pf4j.Plugin;

import org.pf4j.PluginAlreadyLoadedException;

import org.pf4j.PluginRuntimeException;

import org.pf4j.PluginWrapper;

import org.pf4j.RuntimeMode;

public class MenuPluginReport extends Plugin {

@SuppressWarnings("deprecation")

public MenuPluginReport(PluginWrapper wrapper) {

super(wrapper);

}

@SuppressWarnings("deprecation")

@Override

public void start() throws PluginAlreadyLoadedException {

System.out.println(" -> " + getClass().getName() + ".started");

// for testing the development mode

if (RuntimeMode.DEVELOPMENT.equals(wrapper.getRuntimeMode())) {

System.out.println(StringUtils.upperCase(" " + getClass().getName() + ".Runtime"));

}

}

@Override

public void stop() throws PluginRuntimeException {

System.out.println(" -> " + getClass().getName() + ".stopped");

}

@Override

public void delete() throws PluginRuntimeException {

System.out.println(" -> " + getClass().getName() + ".deleted");

}

}

## LIBRARIES FOR PF4J

These are the basic libraries needed to support PF4J and which must be loaded to the HRE project’s Build Path (more recent versions may become available).

| **Product** | **JAR** | **Author** | **Last Version** | **Licence** |
| --- | --- | --- | --- | --- |
| Plugin for Java | pf4j-3.11.0 | PF4j project | 3.11.0 (2024-03-06) | Apache 2.0 licence |
| Simple Logging Façade for Java | slf4j-api-2.0.16 | SLF4J project | 2.0.16 (2024-08-10) | MIT Licence |
| Logback | logback-classic-1.5.7 logback-core-1.5.7 | Logback project | 1.5.7 (2024-08-15) | GNU Lesser Public Licence v2.1 or Eclipse Public Licence v1.0 |
| Java Servlet | javax.servlet-api-4.0.1 | Glassfish Community | 4.0.1 (2018-04-20) | GNU General Public Licence v2 or CDDL v1.1 |

## PLUGIN DUMMY CLASS Javadocs

The following attachment includes Javadocs documenting the Plugin Dummy classes.

