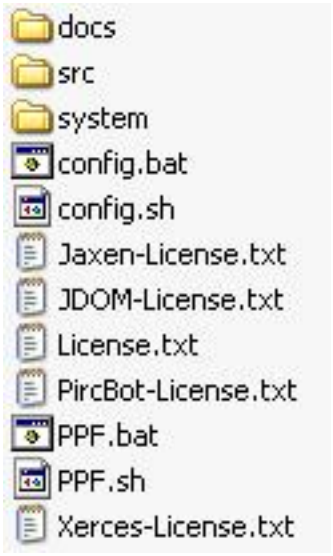
Developers

This information is meant to help developers get started with making a plugin for PPF. This is explained by taking the release package and showing what is needed from that point onwards. It doesn't cover the details needed for editing or compilation within any specific environment. Note: Development can also be done by checking out the sources from CVS and then using your favourite editor/IDE. The majority of the PPF devs use Eclipse.

1. PPF Folder Structure - top level



PPF top level directories

This is where the startup scripts are. It also contains the license files for the libraries used by the PPF code, offline documentation, source files, and the program files.

1.1. docs

HTML and PDF versions of the documentation available for offline browsing.

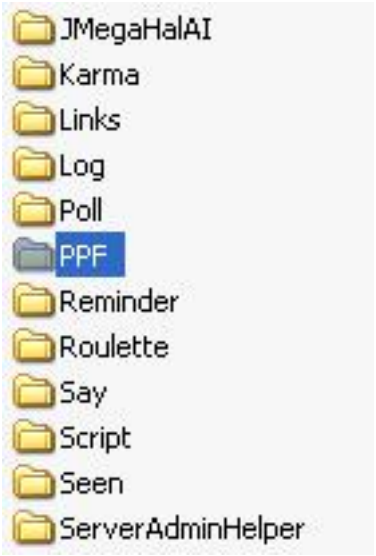
1.2. src

Archives of the sources of each plugin and the PPF core itself.

1.3. system

All of the plugins are here, each in its own directory. The PPF core is also within this directory in the one named PPF.

2. PPF Folder Structure - system



PPF system directories

This (the system directory) contains all of the plugins, each in its own directory with own config file(s) and any libraries needed just for that plugin. PPF is built using plugins, and even the development can benefit from this. A developer can create their plugin and just reload it into the running system rather than stopping and starting each time a change is made. It also forces using the plugin as the user would be, which reduces unexpected errors at deploy time.

2.1. PPF

This is the core part of the system. The main config file (PPFConfig.xml) is located here.

2.2. PPF/lib

The libraries used by the core are kept here, as well as any libraries that can be shared by all plugins.

3. Plugin

3.1. Libraries

If you need to use any 3rd party libraries for your plugins that are not already in the main PPF lib directory, then you can put the JAR files into the plugin directory. They will be automatically picked up for compilation and when creating the release package. They will also be loaded automatically when the plugin is loaded into PPF.

3.2. Config

Configuration of a plugin (where it is configurable) is currently done using XML. The config file is named the same as the plugin name, with Config as a suffix. So a config file for *MyPlugin* would be MyPluginConfig.xml

If a plugin uses a config file, eg: MyPluginConfig.xml, this file is placed to .cvsignore and a file named MyPluginConfig_template.xml is kept with the default settings in. This file is later renamed and copied to the correct name when creating the release package. The reason for this is so that developers can freely use their own configs without them ending up in CVS.

3.3. Build file

Each plugin has its own ANT build.xml. This file can be run on its own and is also called from the main ANT build file when making a release. During the development phase of a plugin, the *deploy* target of this build file is used. This will compile the

plugin source code (failing if there are any errors), and create a JAR from the classes. The plugin is then ready for reloading into a running PPF.

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

<project name="BFMatch" basedir="." default="deploy">

  <property name="plugin.name" value="BFMatch"/>

  <path id="plugin.classpath">
    <fileset dir=".">
      <include name="*.jar"/>
    </fileset>
  </path>

  <!-- Call the PPF build script to perform the build.  Can set the plugin specific
        details here and keep the PPF classpath in one place -->
  <target name="compile">
    <ant antfile="../PPF/build.xml" target="compile.plugin" inheritRefs="true"/>
  </target>

  <target name="deploy" depends="compile">
    <jar destfile="./${plugin.name}.jar" basedir="bin"/>
  </target>

  <target name="release">
    <!-- Copy plugin JAR and required libs -->
    <copy todir="${build.system.dir}/${plugin}">
      <fileset dir="../${plugin}">
        <include name="*.jar"/>
      </fileset>
    </copy>
    <!-- Copy config from template to used location and name -->
    <copy file="../${plugin}/${plugin}Config_template.xml"
      tofile="${build.system.dir}/${plugin}/${plugin}Config.xml"/>
  </target>
</project>
```

3.4. Plugin Lifecyle

When a plugin is loaded, any *.jar*, *.zip*, *.class*, and *.properties* files in the plugin directory (not in any sub directories in the plugin directory) are loaded, ready for use. The core contructs the plugin and the calls the **init()** method of the plugin. While running, all of the PircBot **onXXX()** methods that are fired are called in each plugin. One addition to the PircBot onXXX() methods is an **onAuth()** method called from the PPF core when someone auths with the bot. When a plugin is unloaded the **unload()** method in the plugin is called. This is a place to do any cleanup work. When a plugin is reloaded, it is first unloaded and then loaded.