

TracNav

- [JPFWiki](#) - Welcome Page

Introduction...

Installing JPF...

User Guide...

Developer Guide

- [Design](#)
- [Choice Generator](#)
- [Partial Order Reduction](#)
- [Attributes](#)
- [Listener](#)

MJI...

- [Bytecode Factory](#)
- [Logging](#)
- [Report](#)
- [Embedded](#)
- [JPF tests](#)
- [JPF project layout](#)
- [Create a JPF project](#)
- [Coding Conventions](#)
- [Hosting update site](#)

Projects...

- [Summer Projects](#)
- [External Projects](#)
- [Change\(B\)log](#)

About...

- [Events](#)
- [Presentations](#)
- [Papers](#)
- [FAQ](#)
- [History?](#)
- [Support](#)
- [People?](#)
- [Playground](#)
- [Table of Context](#)

Creating a New JPF Project

So what do you have to do to create a new JPF project? For a quick shortcut to setting *most* things up, see the [jpf-template project](#).

However, jpf-template cannot do everything for you, so see below for more information on how to finish setting up your new project.

Several steps are involved:

1. get familiar with JPF configuration

You need to understand how your project will be looked up and initialized during JPF startup, and the place to learn that is the [JPF configuration](#) page. Once you know what [site properties](#) and [project properties](#) are, you can proceed.

2. get familiar with the standard JPF project layout

Although this is mostly convention, and you can deviate if you really need to, please try hard not to.

You can get the details from the [JPF Runtime Modules](#) page, but the essence is that each project has two (possible) major build artifacts:

- a `jpff-<project>.jar`, which is executed by the host (platform) VM (contains main classes and peers)
- a `jpff-<project>-classes.jar`, which is executed by JPF (contains modeled classes)

Consequently, your sources are kept in `src/main`, `src/peers`, `src/classes`, `src/annotations`, `src/tests` and `src/examples`. You might only have some of these, but please provide regression tests so that people can check if your project works as expected.

All 3rd party code you require at runtime goes into a `lib` directory.

We keep potential annotations separate (and provide additional `jpff-<project>-annotations.jar`) so that external projects (systems under test) can use them without relying on the whole of JPF to be in their classpath. The idea is that this jar does not contain any code which could alter the system-under-test behavior if not run by JPF.

The `tools` directory contains 3rd party libraries and tools that are used at build-time. For convenience reasons, we usually copy the small `RunJPF.jar` and `RunAnt.jar` command jars from the `jpff-core` in here too, so that you can easily build and run from the command line without the need for platform specific scripts, but that is completely optional.

3. create a `jpff.properties` file

Within the root directory of each JPF project, there needs to be a `project properties` file that is named `jpff.properties`. It contains the path settings the host VM and JPF need to know about at runtime. It looks like this:

```
# standard header
<project-name> = ${config_path}

# classpath elements for the host VM (java)
<project-name>.native_classpath = build/<project-name>.jar;lib/...

# classpath elements for JPF
<project-name>.classpath = build/<project-name>-classes.jar;...

# sources JPF should know about when creating traces etc.
<project-name>.sourcepath = src/classes;...
```

You can add other JPF properties, but be aware of that this is always processed during JPF startup if you add your project to the `site.properties`, i.e. it might conflict with other JPF projects. It is better to set global properties (like bytecode factories) from your application property files (`*.jpf`).

4. create your `build.xml`

Our build process is [Ant](#) based, hence we need a `build.xml` file. The standard targets are

- `clean`
- `compile`
- `build` (the default, creates the jars and hence depends on `compile`)
- `test` (run JUnit regression tests, depends on `build`)
- `dist` (creates a binary-only distribution)

If you stick to the general layout, you can use a template like the one attached to this page (of course you need to replace `<your-project-name>`!).

Please note how `site.properties` and `jpff.properties` can be used from within the `build.xml` (Ant understands a subset of the JPF property syntax), which means you don't have to explicitly add the jars of other JPF components you depend on (at least `jpff-core`). You can reference them symbolically like this:

```

<property file="${user.home}/.jpf/site.properties"/>
<property file="${jpf-core}/jpf.properties"/>
..
<!-- generic classpath settings -->
<path id="lib.path">

  <!-- our own classes and libs come first -->
  <pathelement location="build/main"/>
  ...
  <fileset dir=".">
    <include name="lib/*.jar"/>
  </fileset>

  <!-- add in what we need from the core -->
  <pathelement path="${jpf-core.native_classpath}"/>
</path>
...

```

5. add your project to your site.properties

This is optional, you only need to do this if you want to be able to run your JPF extension outside its own directory. If you do, add an entry to your `~/.jpf/site.properties` that looks like this:

```

...
<project-name> = <path to your project>
extensions+=,${<project-name>}

```

6. publish your repository

You can publish this wherever you want (<http://bitbucket.org> is a suitable free site), but we would like to learn about it. If you want to publish on our server <http://babelfish.arc.nasa.gov/hg/jpf>, you have to follow this procedure:

1. get an [account](#) on babelfish
2. send email to Peter.C.Mehlitz at nasa.gov requesting the new repository (the name should start with "jpf-..")
3. wait for confirmation that repo has been created
4. do a `hg clone http://babelfish.arc.nasa.gov/hg/jpf/<your-repo-name>` of the empty repo
5. copy your files into the cloned (empty) repo, then do a `hg addremove` and `hg commit`
6. push your local repo back to babelfish with `hg push https://babelfish.arc.nasa.gov/hg/jpf/<your-repo-name>` (NOTE: use https://)
7. create a page under [projects/start](#) describing purpose, status and repository location of your project, add your repo to the list on the [projects](#) page