
Wigwam 4

Migration Guide

April 2004

042503WMG-1

Contents

Use a Wigwam 3 package in a Wigwam 4 project	
Packages installed from Wigwam 3 package archive	1
In-project packages	1
Prepare playpen for migration	
Re-bootstrap project with Wigwam 4	
Construct in-project source as a package	
Migrate source code to a package	4
Set up in-project package meta-data	5
Install your in-project package	6
Restructure your package overrides	
Move package overrides from package-overrides	7
Move a meta file overrides to package directories	8
Install packages for the project	
Share the migrated project	
If you migrated the project	11
If you need to update to the migrated project	11
Known limitations in Wigwam 4	

DRAFT

Migrate to Wigwam 4

This document provides information on how migrate either your project, package, or package files so that they conform to Wigwam 4.

1. Use a Wigwam 3 package in a Wigwam 4 project

You can easily install Wigwam 3 packages into your Wigwam 4 project. For in-project packages, however, you may need to migrate old style filenames to the Wigwam 4 conventions. The following paragraphs provide more details.

1.1 Packages installed from Wigwam 3 package archive

There are many packages available that were built in the context of Wigwam 3. Wigwam 4 has been designed so that it is fully backward compatible with respect Wigwam 3 packages. Hence, if you install a Wigwam 3 package into a Wigwam 4 project, Wigwam 4 automatically symlinks relevant filenames and as such seamlessly builds and installs that package in the project. You do minimal migration.

- For packages of type `service`: You need to be sure that the `services` file lists only one service per service package.

NOTE. *If you install a Wigwam 4 package into a Wigwam 3 project, the reverse is not necessarily true. That is, the Wigwam 4 package may not build correctly in the Wigwam 3 project.*

1.2 In-project packages

For in-project packages, you need to be sure that your package meta files have Wigwam 4-compliant naming. For example, instead of using the naming conventions `myprogram-local.type`, you should simply name the file as `type`.

To migrate an existing in-project package, you can use symlinks to make the package filenames conformant (e.g., create a symbolic link to `type` from `myprogram-local.type`).

2. Prepare playpen for migration

Before you actually begin migrating your existing project to be Wigwam 4 compliant, do the following:

1. Modify the contents of the `$PLAYPEN_ROOT/etc/package-archives` so that it has both the Wigwam 3 package archives as well as the Wigwam 4 archives. If identically named packages exist in more than one archive, Wigwam uses the first one that it finds. In case there are identically named packages that are in both Wigwam 3 and Wigwam 4 archives, you should list the Wigwam 4 archives *before* the Wigwam 3 archives.
2. You need to save your `project-packages` list. So that it won't get overwritten when you re-bootstrap Wigwam, move the `project-packages` file aside. For example:

```
% mv etc/project-packages etc/project-packages.save
```

3. The non revision-controlled portion of your playpen is called `ext/`, and it gets rebuilt each time you run the Wigwam `autogen.sh` script. For now, remove the `ext/` directory. It will get rebuilt automatically later on. For example:

```
% cd $PLAYPEN_ROOT
% rm -rf ext
```

4. Remove `setup-env` from the revision control repository. This gets re-created when you re-bootstrap wigwam. The following example shows how you'd do this using CVS:

```
% cvs remove -f setup-env
% cvs commit -m 'no longer necessary' setup-env
```

5. Remove the Wigwam 3 `wigwam-bootstrap` binary from your `$PLAYPEN_ROOT/bin` directory.

```
% rm bin/wigwam-bootstrap
```

3. Re-bootstrap project with Wigwam 4

1. Download wigwam-bootstrap for version 4 and set permissions for it.

```
% cd $PLAYPEN_ROOT
% wget -O bin/wigwam-bootstrap 'http://herbie.ddv.com/~pmineiro/wigwam/wigwam-bootstrap'
% chmod +x bin/wigwam-bootstrap
```

2. Commit wigwam-bootstrap to revision control, then run it from that location. The following example uses CVS:

```
% cvs commit -m 'upgrading to ww4' bin/wigwam-bootstrap
% bin/wigwam-bootstrap --from-cvs
```

3. Now that you've run Wigwam 4 wigwam-bootstrap, you've installed Wigwam 4 versions of files. Commit these newer versions to revision control. The following example uses CVS:

```
% cd $PLAYPEN_ROOT
% cvs commit -m 'upgrade to ww4' autogen.sh bin/wigwam-bootstrap
```

TIP FOR CVS: *Since setup-env is no longer revision-controlled, you don't commit that program.*

4. You can now move your project-packages file back. For example:

```
% cd $PLAYPEN_ROOT/etc
% mv etc/project-packages.save etc/project-packages
```

4. Construct in-project source as a package

For Wigwam 4, all project source needs to be Wigwam-packaged. If you've provided in-project source, you need to set it up as an in-project package. You can set in-project packages up with the version `local`.

4.1 Migrate source code to a package

1. Create a `$PLAYPEN_ROOT/src/$PACKAGE-local` directory, where `$PACKAGE` is the name under which you're packaging your in-project source. The following example adds a `$PACKAGE-local/` subdirectory in the playpen *and* adds that directory to revision control. This example uses CVS for revision control:

```
% cd $PLAYPEN_ROOT/src
% mkdir my_ip_source_package-local
% cvs add my_ip_source_package-local
```

2. Move your in-project source to your newly-created `$PLAYPEN_ROOT/src/$PACKAGE-local/` directory.
3. **CVS-specific.** If you are using CVS and you want to preserve revision history for your files, go to `cvs-ssh` and do some manual file copying:

```
% ssh cvs-ssh
% cd /usr/local/cvsroot/my_ip_source/src
% tar cf - java | (cd my_ip_source_package-local; tar xvf -)
% tar cf - web | (cd my_ip_source_package-local; tar xvf -)
% cp build.xml my_ip_source_package-local/.
% logout
```

4. Remove the files that are no longer necessary. The following example uses CVS:

```
% cd $PLAYPEN_ROOT
% cvs upd -dP
% cvs remove -f project-build project-clean project-cleanbuild project-install build.xml
% cvs remove -f java web
% cvs commit -m 'moved to my_ip_source_package-local' java web
% rm -rf java web
% cvsq upd -dP
```

4.2 Set up in-project package meta-data

Once you've packaged your in-project source, you need to provide package meta data files too. This procedure creates meta data files for a raw-type in-project package.

1. Create a `$PLAYPEN_ROOT/packages/$PACKAGE-local` directory. The following example shows how you would
 - a. add a `$PACKAGE-local/` subdirectory in the playpen
 - b. add that directory to revision control (this example uses CVS for revision control)
 - c. go to your newly created `$PACKAGE-local/` directory.

```
% cd $PLAYPEN_ROOT/packages
% mkdir my_ip_source_package-local
% cvs add my_ip_source_package-local
% cd $PLAYPEN_ROOT/packages/my_ip_source_package-local
```

2. Create package meta data files in the `packages/$PACKAGE-local/` directory. The following example creates the files you typically need for a raw-type package. In this example, it creates:
 - a. a `type` file, with the type `raw`.
 - b. a `build` file, containing a build script
 - c. an `install` file, containing an install script

```
% echo raw > type

% cat > build
#!/bin/sh
ant all
^D

% cat > install
#!/bin/sh
ant install
^D
```

3. Create an `etc/in-project-packages` file and add your local package to that file. This ensures that your the package gets installed whenever you rebuild your playpen. Do not include the version number (i.e., `local`). For example:

```
% cd $PLAYPEN_ROOT/etc
% echo my_ip_source_package > in-project-packages
```

TIP. If an `etc/in-project-packages` file already exists, be sure that you don't unintentionally overwrite the existing contents, if any.

4.3 Install your in-project package

Install your in-project package into the playpen. For example:

```
% packagectl install my_ip_source_package local
```

TIP. *In the command line, `local` is optional. Also, note that you don't use a hyphen between the `$PACKAGE` and the `local`.*

5. Restructure your package overrides

If you have package override files, you need to move them around a bit to conform to the Wigwam 4 structure.

5.1 Move package overrides from package-overrides

This procedure describes what you need to do if you want to migrate any package meta files that you've overridden in a Wigwam 3 project.

1. Create a `$PLAYPEN_ROOT/packages/$PACKAGE-$VERSION` directory. The following example shows how you would
 - a. add a `$PACKAGE-$VERSION/` subdirectory in the playpen
 - b. add that directory to revision control (this example uses CVS for revision control)

```
% cd $PLAYPEN_ROOT/packages
% mkdir my_pkg-1.1-2
% cvs add my_pkg-1.1-2
```

2. Move each meta file for your package from the `$PLAYPEN_ROOT/package-overrides/` directory to the new `$PACKAGE-$VERSION/` directory.

```
% cd $PLAYPEN_ROOT/packages/my_pkg-1.1-2
% mv $PLAYPEN_ROOT/package-overrides/my_pkg* .
```

3. While in your `$PLAYPEN_ROOT/packages/$PACKAGE-$VERSION` subdirectory, rename each meta file to conform to the Wigwam 4 naming convention. This example renames the build and install override scripts for the package `my_pkg_override-1.1-2`.

```
% mv my_pkg-1.1-2.build build
% mv my_pkg-1.1-2.install install
```

4. Commit the files in your `$PLAYPEN_ROOT/packages/$PACKAGE-$VERSION/` directory to revision control. This example uses CVS:

```
% cvs commit -m "migrating my_pkg overrides to wigwam 4"
```

5. Go to the `$PLAYPEN_ROOT/package-overrides/` directory and remove the files that are no longer necessary. The following example uses CVS for revision control:

```
% cd $PLAYPEN_ROOT/package-overrides
% cvs upd -dP
% cvs remove -f my_pkg-1.1-2.build
% cvs remove -f my_pkg-1.1-2.install
% cvs commit -m 'renamed and moved to my_pkg-1.1-2 directory in src'
% cvsq upd -dP
```

5.2 Move a meta file overrides to package directories

If a package is installed in your `$PLAYPEN_ROOT/ext/packages/` directory and you provided an override for one or more of its meta files in your `$PLAYPEN_ROOT/packages/` directory, you need to move those files into package-specific directories and rename them using Wigwam 4 naming conventions.

Instead of these files having the form `$PLAYPEN_ROOT/ext/packages/package-version.name`, they need to have the form `$PLAYPEN_ROOT/packages/$package-$version/name`

1. Create a `$PLAYPEN_ROOT/packages/$PACKAGE-$VERSION/` directory. The following example shows how you would
 - a. add a `$PACKAGE-$VERSION/` subdirectory in the playpen
 - b. add that directory to revision control (this example uses CVS for revision control)

```
% cd $PLAYPEN_ROOT/packages
% mkdir packageX-1.1-2
% cvs add packageX-1.1-2
```

2. Move and rename each meta file for your package from the `$PLAYPEN_ROOT/packages/` directory to the new `$PLAYPEN_ROOT/packages/$PACKAGE-$VERSION/` directory.

```
% cd $PLAYPEN_ROOT/packages
% mv $PLAYPEN_ROOT/packages/packageX-1.1-2.options packageX-1.1-2/options
```

3. Commit the files in your `$PLAYPEN_ROOT/packages/$PACKAGE-$VERSION/` directory to revision control. This example uses CVS:

```
% cvs commit -m "migrating packageX overrides to wigwam 4"
```

4. Go to the `$PLAYPEN_ROOT/packages/` directory and remove the files that are no longer necessary. The following example uses CVS for revision control:

```
% cd $PLAYPEN_ROOT/packages
% cvs upd -dP
% cvs remove -f packageX-1.1-2.options
% cvs commit -m 'renamed and moved to packageX-1.1-2 directory in src'
% cvsq upd -dP
```

6. Install packages for the project

Once you've prepared your playpen (see page 2); re-bootstrapped your project with Wigwam 4 (see page 3); and constructed your in-project packages (see page 4), you can install all of the other packages that the project needs.

1. First, be sure that your `$PLAYPEN_ROOT/etc/package-archives` file lists a Wigwam 4 package URL (e.g, `http://herbie.ddv.com/~pmineiro/wigwam/wigwam-bootstrap`) *before* it lists any Wigwam 3 package archive URLs.
2. Install additional Wigwam programs in your playpen. These are the programs that are not automatically installed when you run `wigwam-bootstrap`. For example:

```
% packagectl install servicectl pubtool wigwam-packaging-utils
```

When you execute this command, you install these three packages in your playpen.

3. Run `autogen.sh` in the top directory of the project to rebuild the `ext/` directory in your playpen. For example:

```
% cd $PLAYPEN_ROOT
% ./autogen.sh
```

By running this, you:

- install and build all packages listed in `etc/project-packages/` and `etc/in-project-packages/` into your `ext/packages/` and `ext/src/` directories
 - create and/or update all Wigwam-specific directories and files
 - synchronize your playpen with the referenced package archives (i.e., in `etc/package-archives`) so that your playpen has its own internal representation of the package archives' `package-list`, `dependencies`, `conflicts`, and `provides` files.
 - cause any package profile files are to be automatically copied into `ext/etc/profile` (no manual copying required).
4. Source `setup-env` from your top level project directory so that your shell environment knows about the Wigwam executables that are available as a result of the packages you installed. For example:

```
% . ./setup-env
```

7. Share the migrated project

This section outlines how all developers that are working on a "project X" can now migrate their respective playpens to Wigwam 4.

7.1 If you migrated the project

Now that your `$PLAYPEN_ROOT` has been re-bootstrapped with Wigwam 4, and you've set up and/or installed all the packages in the `in-project-packages` and `project-packages`, you can simply do the following:

1. Commit the revision-controlled directories to revision control so that the other project users know about them.
2. Inform other project users that "project X" has been migrated to Wigwam 4.

7.2 If you need to update to the migrated project

If you've been informed that a project you are using has been migrated from Wigwam 3 to Wigwam 4, you can migrate your playpen as follows:

If `$PLAYPEN_ROOT` has no uncommitted novel source or overrides

If your local system's `$PLAYPEN_ROOT` has no novel files (i.e., files that have not been communicated to other users), and it has no playpen-specific overrides, you can use this procedure.

IMPORTANT. *If you have any in-project packages or overrides that you haven't already committed to revision control (before the project got migrated), you need to **skip** this procedure and go to *If `$PLAYPEN_ROOT` has uncommitted novel source and/or overrides on page 12:**

1. Move the entire "old" `$PLAYPEN_ROOT` out of the way on your local system.
2. Initialize a playpen by following the procedures in *Initialize by using an existing Wigwam project* on page 23.
3. If First, be sure that your `$PLAYPEN_ROOT/etc/package-archives` file lists a Wigwam 4 package URL (e.g., `http://herbie.ddv.com/~pmineiro/wigwam/wigwam-bootstrap`) *before* it lists any Wigwam 3 package archive URLs.

4. Run **autogen.sh** in the top directory of the project to build the `ext/` directory in your playpen. For example:

```
% cd $PLAYPEN_ROOT
% ./autogen.sh
```

By running this, you:

- install and build all packages listed in `etc/project-packages/` and `etc/in-project-packages/` into your `ext/packages/` and `ext/src/` directories
 - create and/or update all Wigwam-specific directories and files
5. Source **setup-env** from your top level project directory so that your shell environment knows about the Wigwam executables that are available as a result of the packages you installed. For example:

```
% . ./setup-env
```

If \$PLAYPEN_ROOT has uncommitted novel source and/or overrides

If your local system's `$PLAYPEN_ROOT` has some novel files or some package overrides that you have not yet committed to revision control, you'll need to move some information around before you re-initialize your playpen with Wigwam

1. If you have in-project source that hasn't yet been committed to revision control, use the procedures in *Construct in-project source as a package* on page 4; otherwise, skip to step 4.
2. Move the `etc/in-project-packages` file aside for now. For example:

```
% cd $PLAYPEN_ROOT/etc
% mv in-project-packages in-project-packages.save
```

3. Move the `etc/in-project-packages` file aside for now. For example:

```
% cd $PLAYPEN_ROOT/etc
% mv in-project-packages in-project-packages.save
```

4. If you have package or package meta file overrides that you haven't yet committed to version control, use the procedures in *Restructure your package overrides* on page 7; otherwise, skip to step 8.
5. Remove your `ext/` directory.
6. Initialize a playpen by following the procedures in *Initialize by using an existing Wigwam project* on page 23.

7. Move the `in-project-packages` file back into the project:

- If there is already an `$PLAYPEN_ROOT/etc/in-project-packages` initialized from revision control, add the contents of your `in-project-packages.save` to that file, and delete your `in-project-packages.save` file. For example

```
% cd $PLAYPEN_ROOT/etc
% echo your_in_project_package_name >> in-project-packages
% rm in-project-packages.save
```

- If there is not already an `$PLAYPEN_ROOT/etc/in-project-packages` initialized from revision control, rename your `in-project-packages.save` as `in-project-packages` file.

```
% cd $PLAYPEN_ROOT/etc
% mv in-project-packages.save in-project-packages
```

8. Run `autogen.sh` in the top directory of the project to build the `ext/` directory in your `playpen`. For example:

```
% cd $PLAYPEN_ROOT
% ./autogen.sh
```

By running this, you:

- install and build all packages listed in `etc/project-packages/` and `etc/in-project-packages/` into your `ext/packages/` and `ext/src/` directories
 - create and/or update all Wigwam-specific directories and files
9. Source `setup-env` from your top level project directory so that your shell environment knows about the Wigwam executables that are available as a result of the packages you installed. For example:

```
% . ./setup-env
```

8. Known limitations in Wigwam 4

This section describes features that were provided in Wigwam 3, but are not yet provided in Wigwam 4.

servicectl check-config

The **servicectl check-config** functionality does not currently exist as an API in Wigwam 4.

Workaround: The user may attempt to start services and, based on the error message, determine that parameters have not been set. The user can look at `ext/$PACKAGE/$VERSION/config` to determine required environment variables for that service, then set those variables appropriately in the relevant project configuration files.

make-package-md5sums

The **make-package-md5sums** API is currently incompatible with Wigwam 4, as it looks for the Wigwam 3 `package.files` naming format as one of its arguments.

Workaround: Use one of the generic md5sum generating tools to produce md5sums for each file listed in `$PACKAGE/$VERSION/files`. Some of public domain, command line utilities include:

- Linux md5sum utility.
- MD5SUM utility, licensed under GNU GPL, and available from:
<http://www.iris.net/gloss/md5sum.shtml>
- md5sum, as part of the GNU Textutils, and available from:
<http://www.gnu.org/directory/GNU/textutils.html>