

The Secret Sucky Guide To Making Releases

Denys Duchier

**Version 1.2.3
December 1, 2001**



Abstract

This document explains how to create releases for various platforms. If you are a Mozart user interested in creating a binary distribution for your platform, you have two main options:

- if your platform supports RPMs, then see how to create binary RPMs using our source RPM
- otherwise, you should create binary tarballs.

Please, Please, Please: do contribute your binary distros back to us. This way, we can place them in our download area and everybody else can benefit from your effort. You do not need to bother with source and documentation distros, since our offerings for those are platform independent. If you are a Mozart release manager, see the last chapter which tells you all the steps you need to follow.

Credits

Mozart logo by Christian Lindig

License Agreement

This software and its documentation are copyrighted by the German Research Center for Artificial Intelligence (DFKI), the Swedish Institute of Computer Science (SICS), and other parties. The following terms apply to all files associated with the software unless explicitly disclaimed in individual files.

The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this software may be copyrighted by their authors and need not follow the licensing terms described here, provided that the new terms are clearly indicated on the first page of each file where they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE AND ITS DOCUMENTATION ARE PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Contents

1	Introduction	1
2	Tarballs	3
2.1	Source Tarball	3
2.2	Documentation Tarball	3
2.3	Standard Library Tarball	4
2.4	Binary Tarballs	4
2.4.1	Non-Linux Platforms	4
3	RPMs	7
3.1	Binary RPMs	7
3.2	Source RPM	7
4	Check-List For Release Managers	9

Introduction

First, you need to download the sources. You can either download the source tarballs from our FTP server, or you can check out the sources directly from our CVS server.

Here is how to get the sources of a particular release from the Mozart CVS archive. A release is identified by a tag, e.g. `mozart-1-2-3`. In the following, I'll write `$REL` for the release tag. Use `ssh`, i.e. set environment variable `CVS_RSH=ssh`.

```
cvs -d :pserver:anoncvs@cvs.mozart-oz.org:/services/mozart/CVS get -r $REL moza
```

Get yourself a cup of coffee, and by the time you've savoured it you have become the proud owner of a copy of the Mozart sources. I will refer to the top directory containing these sources as `$SOURCE`: it is typically called `mozart` unless you used option `-d` to the CVS `get` command (which is unrelated to the `-d` option to the `cvs` command itself). It is highly recommended to also get a copy of the Mozart Standard Library:

```
cvs -d :pserver:anoncvs@cvs.mozart-oz.org:/services/mozart/CVS get -r $REL moza
```

I will refer to the top directory containing the sources of the standard library as `$STDLIB`.

In order to build the release, you should create a separate build directory. I will call it `$BUILD` and it should not be created in `$SOURCE` tree. Simplest is to create it as a sibling of `$SOURCE`. Then you need to configure `$BUILD`: make very sure that no OZ utilities are found in your shell's search path and no OZ related variables are set in your environment. Personally, I have defined shell aliases `ozon` and `ozoff` to switch between having OZ utilities in my path and having a pristine environment blissfully unaware of things Mozartian. A minimally parametrized configuration would be:

```
cd $BUILD
$SOURCE/configure
```

By default, this will set things up to install in the default location `/usr/local/oz`. You can change this default using option `-prefix=$PREFIX`. A more complete invocation of `configure` would look like:

```
$SOURCE --prefix=$PREFIX --with-stdlib=$STDLIB
```

and that's the invocation I would recommend. Next, it's a good idea to set up dependencies in case you need to update the sources and then rebuild what needs rebuilding:

```
make depend
```

Now, you can build the system:

```
make
```

And optionally install it:

```
make install
```

or install it in a different installation directory `$DIR`:

```
make install PREFIX=$DIR
```

Note that it is not possible to skip the `make` step and directly invoke `make install`. At the top level `make` is equivalent to `make bootstrap`. If you want to do it in one command, then you need to type:

```
make bootstrap install
```

If you need to reconfigure, beware that `configure` builds a cache of configuration data in `$BUILD/config.cache`. You may need to remove that.

In order to also build the documentation, you need to invoke `configure` with option:

```
--with-documentation=all
```

Tarballs

2.1 Source Tarball

This assumes that you have checked out the sources and configured a build directory (see Chapter 1). The top level `Makefile` provides support for creating a source tarball:

```
cd $BUILD
make src
```

This will result in the creation of file `mozart- $\text{\$VERSION}$. $\text{\$DATE}$ -src.tar.gz` where $\text{\$VERSION}$ is the release number and $\text{\$DATE}$ is the build date (the current date, or rather, the date when you invoked `configure`). That's it, you're done.

2.2 Documentation Tarball

For this, you need to build and install the documentation. As mentioned earlier, this requires that you configured using option `-with-documents=all`. Note that building the documentation takes very long and you need additional software on your machine (such as LaTeX, nsgmls, Ghostscript, the netpbm package, also java if you are going to build the postscript and pdf documentation, and probably other things I forget).

Let's assume that you configured, built and installed the system (I will not explain here how to additionally create the Postscript and PDF documentation). The top level `Makefile` contains support for creating a documentation tarball:

```
cd $BUILD
make doc
```

This will suck in all the installed documentation and create file `mozart- $\text{\$VERSION}$. $\text{\$DATE}$ -doc.tar.gz`. If you have installed the standard library, this tarball will also contain its documentation.

2.3 Standard Library Tarball

If you have configured using `-with-stdlib=$STDLIB`, then you can create the standard library tarball with:

```
make std
```

This will only work after `make install` because it uses the installed `ozengine` and core libraries. As result the following tarball is created:

- `mozart-$VERSION.$DATE-$PLATFORM-std.tar.gz`

2.4 Binary Tarballs

Here again, we assume that you have configured, built and installed the system. The top level `Makefile` contains support for creating binary tarballs:

```
cd $BUILD
make bin
```

This will suck in from the installation directory all necessary files to include in the binary tarballs. As a result, three tarballs are created:

- `mozart-$VERSION.$DATE-$PLATFORM.tar.gz`
- `mozart-$VERSION.$DATE-$PLATFORM-contrib.tar.gz`
- `mozart-$VERSION.$DATE-$PLATFORM-internal.tar.gz`

where `$PLATFORM` stands for your platform, i.e. the value returned by shell script:

```
$SOURCE/share/bin/ozplatform
```

You can ignore the last tarball. If you have installed the standard library, it will be included in the first tarball.

2.4.1 Non-Linux Platforms

Mozart requires a number of libraries which are not necessarily available by default on Non-Linux platforms. You will need to provide these libraries with the binary release. There are 2 ways: using static linking or using shared object libraries. I will describe the second way. Note, that the situation may be further complicated by the question of whether your platform supports dynamic linking or not.

2.4.1.1 Using Shared Object Libraries

The aim here is to provide both the compiled system together with the dynamic libraries that it requires. Here is a little known fact: the scripts that invoke OZ utilities are designed to modify the search path for dynamic libraries so that the linker also looks first in:

```
$HOME/.oz/platform/$PLATFORM/lib
```

and then in

```
$PREFIX/platform/$PLATFORM/lib
```

where `$PREFIX` stands for the installation directory. Thus, the trick is to plunk the necessary shared libraries into this last directory before building the binary tarballs.

Of course, the assumption is that you will be able to fetch the missing packages and build or otherwise obtain shared object libraries for them. This is usually the case.

2.4.1.2 Tcl/Tk

This is frequently a pain in the rear. Check other documentation as well as `$SOURCE/platform/wish/conf`

RPMs

3.1 Binary RPMs

If your platform uses RPMs, then you are in luck. Download the source RPM from our download page, become somebody really important (I mean `root`) and execute:

```
rpm --rebuild mozart-$VERSION.$DATE-src.rpm
```

This will create the binary RPMs in `/usr/src/redhat/RPMS/i386`.

3.2 Source RPM

In order to create the source RPM, you need to have already created or downloaded the source, documentation, and standard library tarballs. Assuming that you have placed them in `$BUILD`, become somebody important and execute:

```
$SOURCE/misc/create-rpm $BUILD $BUILD
```

Note that this will create both a source RPM and binary RPMs. They will be placed in `$BUILD` or whatever directory you supplied as the second argument to `create-rpm`:

- `mozart-$VERSION.$DATE-$REVISION.src.rpm`
- `mozart-$VERSION.$DATE-$REVISION.i386.rpm`
- `mozart-contrib-$VERSION.$DATE-$REVISION.i386.rpm`
- `mozart-doc-$VERSION.$DATE-$REVISION.i386.rpm`

Check-List For Release Managers

This chapter is intended for release managers of the Mozart consortium. Below is the very abbreviated list of steps to follow for creating a new release:

- bump up the version in file `OZVERSION`
- update version specific changes in `doc/changes/main.shtml`
- cut a new tag in the CVS. For example, if you wish to cut a tag for version 1.2.3 on the current release branch. You should check it out, `cd` to the directory of the checked out version and issue the following command:

```
cvsv tag -r mozart-1-2-3
```

- check out the standard library and also cut the same tag in it. Let's call `$STDLIB` the directory into which you checked it out.
- then you should configure, build and install the release:

```
$SOURCE/configure --with-stdlib=$STDLIB --with-documents=all
```

You can also supply `-prefix=$PREFIX` if you want to install elsewhere than the default; this has no ill-effect.

- As stated above, you should then `make` followed by `make install`. If you did not forget the `-with-stdlib` option, this will also build and install the standard library.
- Now invoke `make src doc bin std` in order to create all the necessary tar files. `make bin` creates a file `README-$VERSION.$DATE-$PLATFORM` which describes the important points about linking: i.e. what libraries `emulator.exe` and `tk.exe` are dynamically linked against. Note that, depending on whether you used `-enable-static-linking` or not, you should rename the tar and README files accordingly so that multiple offerings for the same platform but with different linking policies can be distinguished. My own preference is to add `-dynfull` when dynamic linking is used and `-dynless` when `-enable-static-linking` (and possibly `-enable-modules-static` - I normally specify both of them when I am interested in a more statically linked distribution) was used instead. For example, here are the names of the tar and README files for release 1.2.3 for the dynamically linked version of the mozart release for freebsd on the intel architecture:

```

- README-1.2.3.20011121-freebsdelf-i486-dynfull
- mozart-1.2.3.20011121-freebsdelf-i486-dynfull-contrib.tar.gz
- mozart-1.2.3.20011121-freebsdelf-i486-dynfull.tar.gz

```

- now become root and invoke `$SOURCE/misc/create-rpm $PWD $PWD`. This creates the source and binary rpms.
- Now you need to create and populate the ftp directory for the release. For this, you need to be logged into the consortium's ftp machine. Currently this is `grizzly` in Saarbrücken. Become `root`:

```
perl $SOURCE/misc/mkftp -release=$VERSION -create
```

this creates the directory structure for the release. Use option `-n` to see what `mkftp` would do without actually doing it.

```
perl $SOURCE/misc/mkftp -release=$VERSION -tar *.tar.gz README-*
```

this will install the tar and README files appropriately.

```
perl $SOURCE/misc/mkftp -release=$VERSION -rpm=redhat *.rpm
```

install the source and binary RPMs.

- Now you need to make the postscript and pdf documentation. `cd` to the build doc directory and invoke `make ps` followed by `make pdf`. Finally do `make installps` which installs the documentation in the global `print` directory of your installation. Now propagate this documentation to the ftp directory:

```
perl $SOURCE/misc/mkftp --release=$VERSION --doc=$PREFIX
```