Using imake for Multiple Projects

Paul DuBois
dubois@primate.wisc.edu

Wisconsin Regional Primate Research Center
Document revision: 1.05
Revision date: 28 August 1993

1. Introduction

imake is a tool for generating Makefiles and is used to allow software projects to be configured easily on a wide variety of target machines. It is distributed with projects such as the X Window System and the Kerberos authentication system. The tool *makedepend* is typically used in conjunction with *imake* to generate *Makefile* dependencies. Another tool is used to bootstrap a *Makefile* from an *Imakefile*. This has traditionally been *xmkmf*; a variant called *imkmf* is discussed here, too.

This document describes the use of *imake* to configure projects at the Wisconsin Regional Primate Research Center. The WRPRC configuration architecture allows multiple sets of configuration files to peacefully coexist on a single machine so that any of them can be applied to the configuration of an arbitrary number of distinct projects. It allows redundancy among project configurations to be exploited while providing for project-specific information which does not affect other projects. Information on obtaining these tools and configuration files is provided at the end of this document.

The method described here was patterned after the architecture of the configuration files used in the X Window System (version 11, release 4). This decision was made because that architecture was significantly more developed than that of either X11R3 or Kerberos (then at protocol revision 4). Since that time Kerberos V5 has been released (in beta) and its configuration too has been patterned after X11R4. X11R5 was released later yet; its architecture is essentially similar to that of X11R4.

A more-than-passing acquaintance with *imake* is assumed on the reader's part. Other documentation you might want to read includes "Configuration Management in the X Window System" by Jim Fulton (in *mit/doc/config/usenixws/paper.ms* from the X11 distribution) and the configuration and *README* files (in the *mit/config* directory). You should also be willing to look at the WRPRC configuration files firsthand.

2. Statement of Problem

imake is useful because it helps you keep system dependencies out of your source code and put them into a set of separate configuration files. The files are used to select, on a platform-specific basis, the proper configuration information for a particular system and to build Makefiles accordingly. Since *make* is assumed to be the main tool used to control a project's build process, reconfiguring the Makefiles for a different system amounts to running *imake* again, rather than having to edit all the Makefiles by hand.

Having discovered the flexibility *imake* provides as a configuration tool (once I figured out how to use it!), I began to use it on all my projects. Since these were all developed at the same site, the projects naturally had quite a bit of configuration information in common. At the same time, different projects had their own individual requirements.

The X11R4 project allows configuration files to be found either (a) in the *config* directory under the X top-level directory or (b) in a well-known system-wide location (e.g., */usr/lib/X11/config*). For my own projects I would have liked to adopt approach (b) and install a single set of system-wide configuration files and use them for all my projects. That wasn't suitable because the requirements for my projects tended to differ somewhat. Instead, I adopted approach (a) on a per-project basis: under each project's top-level directory I created a *config* directory and placed a full set of configuration files in it, modified as necessary for the project.

This was the brute force approach, of course, but it worked … sort of. It was most useful when moving a project to a *different* machine, since rebuilding the Makefiles to configure them on a per-machine basis was much easier. (That after all was the original purpose of *imake*). The approach was less useful for sharing configuration information among projects on the *same* machine, because of the multiple sets of files:

- There was a great deal of redundancy between these sets since projects were built and installed in similar ways. That was dissatisfactory because it seemed rather wasteful and silly not to take advantage of the overlap.
- The many copies of files contributed to a certain amount of maintenance difficulty. For instance, if I decided to change the way a rule worked, and the rule was used in several projects, that meant changing the rules file for each of those projects.

The last result was somewhat paradoxical as *imake* is intended to reduce maintenance chores, not contribute to them. Evidently a boundary condition on *imake*'s capabilities was being reached, which led to two questions: how can the number of configuration files to be maintained be minimized, and how can configuration files be shared among projects without sacrificing the ability to specify project-specific information as necessary?

3. Solution to Problem

imake possesses the ability to look through multiple directories when looking for configuration files (via the −*I* command line option), so to allow it to be used to configure multiple projects with the same set of files the simplest solution to the problem seemed to be:

- Create a set of standard configuration files and store them in a well-known public location for any project's use.
- Allow projects to define an alternate directory in which private project-specific configuration files can be placed.
- Tell *imake* to look through the private project-specific directory first when searching for configuration files, then through the shared public directory for any files not found.

In this document `PubConfigDir` and `PrivConfigDir` denote the public and private configuration directories, respectively. `PrivConfigDir` is the *config* directory under the project's top-level directory, i.e., `$(TOP)`*/config*.

4. Implementation of Solution

This section describes the X11R4 configuration file architecture and how it was modified for multiple-project use.

4.1. Imake.tmpl Architecture

The architecture of *Imake.tmpl*, the *imake* template used in X11R4 is, roughly:

vendor block section	
`#include <`*platform*`.cf>`	(platform-specific definitions)
`#include <site.def>`	(site-specific definitions)
system description, build definitions	
`#include <Project.tmpl>`	(project-specific definitions)
`#include <Imake.rules>`	(*imake* rules)
`#include "./Imakefile"`	(*Imakefile* for current directory)
default *Makefile* target entries	

The vendor block section determines the machine type and the platform-specific file *platform.cf* to be included. Information in *platform.cf* controls the selection of configuration information from the other included files.

The X11R4 architecture can be modified to accommodate multiple projects. To do this, extra `#include` files are introduced into *Imake.tmpl*. The resulting modified architecture is shown below, with the new files in boldface:

vendor block section
`#include <`*platform*`.pcf>`
`#include <`*platform*`.cf>`
`#include <site.pdef>`
`#include <site.def>`
system description, build definitions
`#include <ProjectGroup.ptmpl>`
`#include <ProjectGroup.tmpl>`
`#include <Imake.prules>`
`#include <Imake.rules>`
`#include "./Imakefile"`
default *Makefile* target entries

This architecture differs from that used in X11R4 in the following respects:

- The vendor block section now additionally determines the name of the project-specific platform file *platform.pcf*, not just *platform.cf*. For instance, the original and modified Sun vendor blocks look like this:

X11R4 vendor block

```
#ifdef sun
#define MacroIncludeFile <sun.cf>
#define MacroFile sun.cf
#undef sun
#define SunArchitecture
#endif /* sun */
```

Modified vendor block

```
#ifdef sun
#define MacroIncludeFile <sun.cf>
#define MacroFile sun.cf
#define ProjectMacroIncludeFile <sun.pcf>
#define ProjectMacroFile sun.pcf
#undef sun
#define SunArchitecture
#endif /* sun */
```

- *Project.tmpl* is renamed to *ProjectGroup.tmpl* to reflect that it applies to a group of projects, not a single project.
- The template includes a number of additional files. This modification is based on the notion of override files. Before any of the platform, site, project-group or rules files are included, another related file is included which may contain project-specific information. Default versions of these project-specific files (**.pcf*, *site.pdef*, *ProjectGroup.ptmpl* and *Imake.prules*) are present in `PubConfigDir` (to prevent *cpp* from issuing file-inclusion errors) but are all empty. When a given project has special configuration requirements, it supplies its own versions of any of these files in its private configuration directory, `PrivConfigDir`. Telling *imake* to search that directory before `PubConfigDir` causes the project-specific versions to be used, overriding the empty dummy versions.

The disadvantages of this architecture are obvious:

- *Imake.tmpl* is more complex.
- `PubConfigDir` must be populated with the additional **.pcf*, *site.pdef*, *ProjectGroup.ptmpl* and *Imake.prules* files, although this is easy since they are all empty:

```
% touch site.pdef
% touch ProjectGroup.ptmpl
% touch Imake.prules
% ...
```

The advantages are:

- Configuration information is centralized. Multiple projects may use files in `PubConfigDir` cooperatively.
- Flexibility is maintained. Changes can be made to the configuration of one project without affecting other projects, by specifying those changes in the project's private files. This contributes to configuration file stability, since the public files don't change just because a single project changes.
- On the other hand, when changes (e.g., improvements, bug fixes) *do* need to be made to the public files, the changes are immediately available to any project using them via a normal configuration step, i.e.:

  ```
  % make Makefile
  % make Makefiles
  ```

  (Of course, it is prudent to exercise due caution — it becomes possible to break a lot of projects if an erroneous change is made.)
- The number of configuration files referenced by *Imake.tmpl* is greater, but when multiple projects are maintained, the overall number of configuration files involved is smaller and less space is used since only one set of public files in needed.
- The public configuration files need not be distributed with every project that uses them. They can be made into a separate distribution; only the project-specific configuration files need be distributed with projects that use the public files.

The philosophy underlying the architecture is that the public files *Imake.tmpl*, **.cf*, *site.def*, *ProjectGroup.tmpl* and *Imake.rules* describe a common ‘‘baseline’’ or standard configuration, and individual projects need to specify configuration information using private **.pcf*, *site.pdef*, *ProjectGroup.ptmpl* and *Imake.prules* files only to the extent that they *differ* from the standard. Thus, private override files minimize the need for individual projects to modify the baseline files directly.

Some of the ramifications of this architectural change include:

- At the one extreme, when projects have no special project-specific configuration requirements (no *config* directory, or an empty one), only the files in `PubConfigDir` are used to create Makefiles. Since the default project-specific files are empty, the new architecture of *Imake.tmpl* is functionally equivalent to the old. At the other extreme, projects can supply *all* the configuration files if they wish so that only files in `PrivConfigDir` are used.
- The public *site.def* file might be used by the system administrator to set the value of a macro such as `BinDir` to some public directory in which general access user programs are installed. Invocations of installation rules would then normally refer to the corresponding *make* variable `BINDIR`. If a project needs its programs installed somewhere else, this could be done by defining `BinDir` in the project-specific site file *site.pdef* to override the value in *site.def*.
- Override files may also be used to extend the standard configuration by defining symbols not appearing at all in the public files. For instance, if some of a project's programs were to be installed in the public `BinDir`, but others were to be installed in a private directory, a symbol such as `ProjBinDir` could be defined for the latter purpose in *ProjectGroup.ptmpl*.

4.2. Other Architecture-related Changes

The architecture change described above required modification of those things which cause *imake* to be executed, such as the configuration file definitions used to construct *imake* commands, and the procedure for bootstrapping a *Makefile* from an *Imakefile*.

4.2.1. IMAKE_CMD

imake is invoked when "make Makefile" is performed, through the use of the *Makefile* variable `IMAKE_CMD`. The definition of this variable in the configuration file *ProjectGroup.tmpl* was changed.

Old (from X11R4):

```
#ifdef UseInstalled
        IRULESRC = $(CONFIGDIR)
        IMAKE_CMD = $(IMAKE) -DUseInstalled \
                -I$(IRULESRC) \
                $(IMAKE_DEFINES)
#else
        IRULESRC = $(CONFIGSRC)
        IMAKE_CMD = $(NEWTOP)$(IMAKE) \
                -I$(NEWTOP)$(IRULESRC) \
                $(IMAKE_DEFINES)
#endif
```

New:

```
#ifdef UseInstalled
        IMAKE_CMD = $(IMAKE) -DUseInstalled \
                -I$(NEWTOP)$(PROJCONFIGDIR) -I$(GROUPCONFIGIR) \
                $(IMAKE_DEFINES)
#else
        IMAKE_CMD = $(NEWTOP)$(IMAKE) \
                -I$(NEWTOP)$(PROJCONFIGDIR) -I$(GROUPCONFIGIR) \
                $(IMAKE_DEFINES)
#endif
```

The old rule determines, based on whether `UseInstalled` is defined or not, whether to use a version of *imake* already installed in some public directory or build it in the source tree. That symbol also determines where to look for configuration files and defines the value of `IRULESRC` accordingly. If `UseInstalled` is undefined, searches look in `CONFIGSRC`, the project's *config* directory, otherwise in `CONFIGDIR`, the system-wide configuration directory.

This behavior is unsuitable for the purposes described here, since there are two directories which should be searched, if they exist. The new rule makes `UseInstalled` relevant only for determining where to find (and possibly build) *imake*, not where to look for configuration files. The file are always searched for in both places, which renders `IRULESRC` obsolete.

`UseInstalled` is typically relevant only for configuring the WRPRC configuration distribution itself, which must build *imake* and cannot assume it's already installed somewhere. Other projects most likely will be configured on the assumption that *imake* is publicly installed.

4.2.2. Bootstrapping Makefiles — xmkmf and imkmf

xmkmf is used to bootstrap a *Makefile* from an *Imakefile*, but will not accommodate the two-directory-search scheme outlined above. To handle this, *imkmf* was developed, essentially as a copy of *xmkmf* modified to search `PrivConfigDir` and `PubConfigDir`, not just `PubConfigDir`.

The name *imkmf* was chosen to better reflect the *imake*-related function of the program; the name *xmkmf* reflects its X-specific orientation.

4.3. Test of Implementation

After making the changes described above, a flight test was arranged. Nine projects developed under six different logins were converted to use the public files. Projects supplied override files where necessary. These projects were all initially developed using configuration files that bore a fairly strong relationship to the public set, so it was expected that conversion would not be unbearably odious. The degree of relationship broke down as follows:

- Two projects used configuration files that were identical (or nearly so) to the public files. Each of these projects, unlike the other seven, comprised only a single source directory. As expected, conversion was completely trivial, once the top-level *Makefile* was rebuilt with *imkmf*.
- Four projects required additional project-specific specification beyond the public configuration (special include file directories, link libraries, installation directories, etc.). These usually required only *site.pdef* and *ProjectGroup.ptmpl* files in their local *config* directories. One project required a *platform.pcf* file. Project owner and group names (used for install operations) were moved into *site.pdef* files from *site.def* files, which were then deleted from the sets of private configuration files.
- Three projects diverged from the public files fairly significantly, because although their configuration files were initially set up from files similar to the public ones, less effort had been exerted to keep them in sync as project development progressed. As cladists would say, they developed autapomorphies.[1] Besides the changes made to the projects described in the previous paragraph, there were additional special commands for compilation, special *Imakefile* rules, etc. Nevertheless, conversion proceeded smoothly and without incident.

One thing I had done in my original *Project.tmpl* files was to define certain *Makefile* variables directly (e.g., `INCLUDESRC=$(TOP)/h`), rather with the standard X11R4 technique of using a previously `#define`'d *cpp* symbol (e.g., `INCLUDESRC=IncludeSrc`, where `IncludeSrc` is `#define`'d within an `#ifndef/#endif` block). This was a short-cut that works in a non-shared environment but needed to be fixed to work in a shared environment. A default value of `IncludeSrc` was placed in *ProjectGroup.tmpl*, and individual projects redefine that symbol as necessary in *ProjectGroup.ptmpl*.

It was also necessary to place a number of *cpp* symbol definitions in *ProjectGroup.tmpl* within `#ifndef/#endif` blocks, since they became liable to being overridden in *ProjectGroup.ptmpl*.

Overall, conversion was reasonably simple and straightforward. I judged the ‘‘conversion odiousness coefficient’’ acceptably low.

5. Accommodating Multiple Groups of Projects

The architecture described so far allows a group of similarly-configured projects to share configuration files and selectively override or extend them as necessary. This provides a solution to the original problem (how to avoid maintaining multiple full sets of configuration files while maintaining flexibility), but leads to another. Suppose that instead of a group of projects, there exist several groups of projects, such that projects within a group are similar, but differences between groups are quite large. This can happen, for instance, if projects are installed that originated at several different sites. It is likely that developers at different sites will, at the least, use different sets of rules and test for different sets of system characteristics.

The difficulty here is that there are multiple sets of public configuration files, and they cannot all be installed in the same well-known location. This problem can be overcome quite simply. Instead of using `PubConfigDir` as the single well-known location in which the public files are installed, define that directory as `PubConfigPath` and use it as the root of a set of subdirectories, each of which corresponds to a set of configuration files. Each set of files is installed in the appropriate directory, and `PubConfigDir` for that set is defined in *ProjectGroup.tmpl* as that directory (e.g., `PubConfigDir` for group *abc* is `PubConfigPath`*/abc*).

To allow *imkmf* to be used for bootstrapping, it was modified slightly again to allow selection of any of the configuration group subdirectories with a −*C* option. For flexibility, the argument following −*C* may be absolute or relative. If absolute, it is used unchanged. If relative, `PubConfigPath` is prepended.

imkmf as described has some significant differences from the *xmkmf* program distributed with X11R4. However, *imkmf* can be used in a way compatible with *xmkmf* and thus used to configure X projects. If the X configuration files are installed in */usr/lib/X11/config* and the value of `PubConfigPath` is */usr/lib/config*, the X11R4 configuration files can be linked into the directory structure used by *imkmf*:

[1] Although most cladists might decline to attribute these to a designer.

```
% cd /usr/lib/config
% ln -s /usr/lib/X11/config X11
```

For sites without symbolic links, */usr/lib/config/X11* can be created as a regular directory into which the files from */usr/lib/X11/config* are copied:

```
% cd /usr/lib/config
% mkdir X11
% cp /usr/lib/X11/config/* X11
```

Ugly, but workable. (Also known as "crude, but effective.")

With these conventions, projects within groups can share configuration files (as before), project groups can coexist peacefully, and WRPRC *imkmf* is compatible with the behavior of X11R4 *xmkmf*.

The nine projects previously converted were tested against these additional modifications. They required only the following to properly reset the configuration information in the Makefiles:

```
% imkmf -C WRPRC
% make Makefiles
```

(*−C WRPRC* selects the standard Primate Center configuration files.)

6. Developing New Projects

6.1. Describing the Configuration

If a new project has any special configuration requirements, create a *config* directory and specify those requirements in private configuration files. Also, create a *README* file explaining what they are.

6.2. Distributing Your Project

The mechanism described in this document solves the problem of reusing configuration files while allowing for extension and override within individual projects. However, when a project is packaged for distribution, it's very little use to include only those configuration files which are unique to the project if the receiving site doesn't have the public files. It seems simplest to provide the public files as a separate distribution, along with the sources to the configuration tools. This is perhaps inconvenient for the recipient, as it is another distribution that needs to be retrieved. On the other hand, it only needs to be done once, and then it's not necessary to distribute all the configuration files with each project that uses them.

7. Distribution and Update Availability

The WRPRC configuration distribution may be freely circulated and is available for anonymous FTP access in the file *~ftp/pub/imake-stuff/config-WRPRC.tar.Z* on host *ftp.primate.wisc.edu*. Updates appear there as they become available.

The WRPRC distribution illustrates the concepts discussed in this document. It also includes (modified) parts of the X11R4 *mit/util* source tree (where *makedepend* is found) as well as *msub*, a locally-developed utility for yanking the values of *make* variables out of Makefiles and using them to process templates to produce target files such as shell or *perl* scripts, help files that reflect local convention, etc.

The same directory contains the X11R4 configuration files, in *config-X11R4.tar.Z*. It might be instructive to compare them against the WRPRC files. There is also a paper describing the X11R4 configuration architecture in some detail.

If you do not have FTP access, send requests to *software@primate.wisc.edu*. Bug reports, questions, suggestions, and comments may be sent to this address as well.