SHAKEMAP SOFTWARE GUIDE

Table of Contents

3.1 Introduction 3
3.2 System and Software Requirements 4
3.2.1 Operating System 4
3.2.2 Perl 4
3.2.3 GMT 5
3.2.4 convert 6
3.2.5 Ghostscript 6
3.2.6 Make 6
3.2.7 Subversion 7
3.2.8 C compiler 7
3.2.9 MySQL 7
3.2.10 mp (Metadata Parser) 7
3.2.11 Zip 8
3.2.12 Ssh 8
3.3 Installing the Software 8
3.3.1 Installing and Configuring MySQL 8
3.3.2 Installation and Upgrade 10
3.4 Customizing ShakeMap 13
3.4.1 Region-Specific Files 13
3.4.2 Configuration Files 13
3.4.3 Vs30 Grids 13
3.4.4 Site Amplification Tables 14
3.4.5 MMI Site Amplification Tables 15
3.4.6 Passwords 15
3.4.7 Web Pages 15
3.4.8 Automation 16
3.4.9 Overriding the Automation (input/source.txt and input/timestamp_text.txt) 17
3.4.10 Ground-Motion Prediction Equation (GMPE), Intensity Prediction Equation (IPE), and Ground-Motion/Intensity Conversion Equation (GMICE) Modules 17
3.5 Running ShakeMap 23
3.5.1 Data Directory Structure 23
3.5.2 Creating the Maps 23
3.5.3 The Processing Sequence, *shake*, and Versioning 24
3.5.3.1 shake.conf 24
3.5.3.2 The Processing Sequence and shake.conf 24
3.5.3.3 Flags, Versions, and the MySQL Database 26
3.5.3.4 Passwords and mydb.conf 30
3.5.3.5 Backing up the MySQL database 31
3.5.4 Special Issues When Running ShakeMap 31
3.5.4.1 A Note About Program Flags 31
3.5.4.2 Using User-supplied Estimates 32
3.5.4.3 A Note About Estimates and Flagged Stations 33
3.5.4.4 A Note about Finite Faults 33
3.5.4.5 Directivity 34
3.5.4.6 Scenarios 35

3.5.4.7 Getting Help 36
3.6 Common Problems 36
3.6.1 Shake flags database causes confusion 36
3.6.2 Files in incorrect format: 36
3.7 XML Formats in ShakeMap 37
3.7.1 About XML 37
3.7.2 ShakeMap XML Files 38
3.7.3 Retrieving Data from a Database 43
3.7.4 External Data XML Files 43
3.8 Development Model 45
3.9 Tables 45
3.10 References 54

3.1 Introduction

ShakeMap is a collection of programs, largely written in the Perl programming language. These programs are run sequentially to produce ground motion maps (as PostScript and JPEG images, GIS files, etc.) as well as web pages and email notifications. In addition to Perl, a number of other software packages are used. In keeping with our development philosophy, all additional software required by ShakeMap is freely available. This chapter explains what is required to install and run ShakeMap.

This version of the Software Guide is peculiar to ShakeMap V3.5 and later.

The following conventions are used throughout this *Guide*:

`Courier Text & prompt (%)`	`User Input, commands, and screen displays`
< brackets >	User-assigned or environment-specific <Variables>
italics	ShakeMap and non-ShakeMap *programs* other *files* and *directories*
-italics	required or optional program *flag*
‘single’ or “double quotes”	“file” or “subdirectory” names
http://www.web.org	Web Page URL

3.2 System and Software Requirements

Before ShakeMap can be installed and run, a number of other software packages and Perl modules must be installed. This required software is described in the sections that follow.

If you intend to install and operate ShakeMap, please see the section Getting Help and sign up for the shake-dev mailing list. The mailing list is the main channel of communication between ShakeMap operators and developers.

3.2.1 Operating System

ShakeMap Version 3.5 was developed and tested on systems running the MacOS X and Linux. Version 3.0 and earlier were developed for the SPARC version of Solaris v2.6, v2.7 and v2.8 (i.e., SunOS 5.6, 5.7, and 5.8). Version 3.0 and higher of ShakeMap will run on FreeBSD, and v3.2 and later support Linux. ShakeMap will also run under various versions of MacOS X, though we can only effectively support the version our developer uses (currently 10.6.8). In this guide, we have tried to make note of any differences among Linux, Solaris, and FreeBSD installations of ShakeMap. We have never tested ShakeMap v3.5 with the SPARC version of Solaris, but we expect that it would work. For any other OS, you will be blazing your own trail. In particular, many of the programs would probably work under another OS, but *transfer* might be problematic. In addition, the makefiles we use are very Unix-like and use GNU-specific extensions (we get around this on FreeBSD by using *gmake*, which supports the extensions we use).

Warning: We have had several reports of significant problems running ShakeMap on Ubuntu Linux. At this point we do not recommend Ubuntu, and cannot support it. If that is your platform of choice, you are on your own.

3.2.2 Perl

Perl should be pre-installed on any system upon which ShakeMap will run, but you may wish to upgrade it (Perl 5.8.0 is known to cause problems, so definitely upgrade if that is your system's default Perl). We are using version 5.8.5, 5.8.7, 5.8.8 and 5.10.0. While we think most versions of Perl will work, there have been reports of problems with specific sub-versions. Whether these were problems with Perl itself, or with some combination of Perl and the modules used, we do not know. If you run into problems with specific Perl statements, please let us know. If there are problems with module compatibility, try a different (usually earlier) version of the module, and let us know what the problem was and what version of the module was problematic.

Perl may be obtained for free from several sources, and is usually included by default on most of the supported systems. Visit www.perl.com to find a download point for your particular OS.

ShakeMap also uses several Perl modules that may be obtained from CPAN (see www.cpan.org for CPAN archives). For Linux users, most of these modules are available via the package manager system. For FreeBSD users, most of these modules are available for automated installation via the ports collection. We list here the versions of the modules we are using and that are therefore known to work (at least on our system), your mileage may vary. You may use

the most recent version of modules, or the easiest to get and install, but if you run into problems, start backing the suspect modules out to earlier versions. Modules:

Module Name	Version
libwww-perl	(5.69)
Bundle::LWP [1]	See footnote #1
DBD::mysql	(4.20.0)
HTML::Template	(2.900)
XML::Parser	(2.400) Requires expat be installed[2]
XML::Writer	(0.606)
enum	(1.016)
Time-modules	(2006.81.400) Installing module Time::CTime will get this package
Event	(1.18)
Mail::Sender	(0.8.16)
DBD::Oracle [3]	(1.14) See footnote #3
Config::General	(2.500)
XML::Simple	(2.180)
Time::y2038	(20100403) Has numerous dependencies.

The version numbers listed above are known to work with V3.5 of ShakeMap. Other versions may work, but we can't guarantee it.

3.2.3 GMT

ShakeMap requires GMT, The Generic Mapping Tools developed by Paul Wessel and Walter H.F. Smith. GMT is freely available from www.soest.hawaii.edu/gmt. ShakeMap V3.5 is compatible with GMT 4.3, 4.4, and 4.5 (up through 4.5.7). Other versions are not recommended. In particular, versions newer than 4.5.7, when they become available, may not work. While the GMT web site usually gives you the latest stable version, the mirror sites usually have older versions available.

GMT requires the NetCDF software as part of its install process. The GMT installation documentation will tell you how to obtain and install NetCDF.

---

[1] This is a prerequisite for libwww-perl. If you use a system like 'ports' or the CPAN module to install libwww-perl, the LWP bundle is automatically installed, too. If you do the installation manually, you need to install Bundle::LWP before libwww-perl.

[2] Expat can be downloaded from http://sourceforge.net/projects/expat/. Configuration and installation are explained in the expat README. Expat will be automatically installed if you install XML::Parser with an automatic tool like port or cpan.

[3] DBD::Oracle is needed to connect to an Oracle database. It is used by programs like db2xml, eq2xml, etc. If you are using a database other than Oracle, you will need to get a different driver (e.g., DBD::Sybase). If you are providing data to ShakeMap through some other mechanism, you won't need this module.

IMPORTANT: When installing GMT, NetCDF, and ShakeMap, it is important that they all are compiled with the same compiler and compiler options. If you are pulling pre-compiled packages from various locations, you may get incompatibilities and ShakeMap will NOT work correctly. Most of the Linux distributions have package installers that use the same compiler that is installed by default, so it usually works. But if you have problems, you may need to compile GMT and NetCDF from source.

Note that when installing GMT, you need to get and install the full-resolution coastline data. This is an option in the GMT installation scripts.

If you upgrade your version of GMT, make sure to edit <shake_home>/include/macros to reflect your changes, and then go to <shake_home> and run 'make veryclean' and 'make' to update all of your code. Test thoroughly.

3.2.4 convert

There are two recommended ways within the ShakeMap program *genex* to convert the PostScript output from the GMT programs to JPEG and PNG:

(1) Use *convert* from ImageMagick. The program can be obtained from www.imagemagick.org. It is free. Ghostscript (see below) is required for *convert* to process PostScript. We are using versions 5.5.7 and 6.2.8 of convert. Make sure that your installation of *convert* can convert PostScript to JPEG and PNG before proceeding with the ShakeMap installation. If you use *convert*, set the genex.conf parameter "convert" to the path to your *convert* executable.

(2) Use *gm* from GraphicsMagic, available from http://www.graphicsmagick.org. It, too, is free. Click on the "Installation" link for a list of sites from which to obtain necessary third-party software. You will need, at a minimum, Ghostscript, ZLIB, LIBPNG, and the Independent JPEG Group's JPEG software. Version 1.3.12 of *gm* seems to work. Other versions will probably work as well, but have not been extensively tested. If you use *gm*, set the genex.conf parameter "convert" to the path to *gm* followed by the string "convert" (e.g., "convert : /usr/local/bin/gm convert").

gm is recommended on Linux, where it makes better-looking text.

3.2.5 Ghostscript

Ghostscript is used by *convert* and *gm* for conversion of PostScript to JPEG and PNG. We use various versions of Aladdin Ghostscript (7.07, 8.00). Use whatever version is recommended for your version of convert. The software is free and can be downloaded through sourceforge: http://sourceforge.net/projects/ghostscript.

3.2.6 Make

On Linux, GNU make will likely be your default. On Solaris, use Sun's make or GNU make (http://directory.fsf.org/GNU/make.html).

On FreeBSD, you will want to get *gmake*, the GNU make from http://directory.fsf.org/GNU/make.html. This is easily installed (as are many of the Perl modules) through the ports collection.

The 'install' program on Linux is defined as */usr/bin/install* and on Solaris */usr/ucb/install*. Please double check your 'macros' file in <shake_home>/include after you run *make* in <shake_home>/install to be sure that the correct version of 'install' is selected.

3.2.7 Subversion

You will need the Subversion (SVN) client code in order to check out a version of the ShakeMap code base. The code is freely available for many, many operating systems from http://subversion.tigris.org/ and elsewhere. Many systems come with the SVN client already installed or available with the OS distribution. You do not need the server code, only the client. Using SVN, you will be able to receive code updates much more easily than in the past.

3.2.8 C compiler

You will need a C compiler. On Solaris, we use Sun's, and on Linux and FreeBSD, we use GNU's (again, use the ports collection to install *gcc*). If you will use *gcc* on Solaris, you can get it from (http://directory.fsf.org/GNU/gcc.html). In either case, you will specify the compiler and compiler flags in '<shake_home>/include/macros.'

It is important that the compiler you select is the same one used to compile GMT and the NetCDF package, so be careful if you get pre-compiled versions of these packages. In a perfect world, this restriction would not exist, but this world is far from perfect.

3.2.9 MySQL

Please follow the instructions in the section Installing and Configuring MySQL, below, for configuring MySQL, and for converting existing pre-V3.0 ShakeMap databases to MySQL.

3.2.10 mp (Metadata Parser)

ShakeMap produces FGDC-compliant metadata and provides it as text, HTML and XML on the downloads page. Producing the HTML and XML requires the program 'mp' (which may be obtained from http://geology.usgs.gov/tools/metadata/tools/doc/mp.html). Once you have installed ShakeMap (see installation instructions, below), download, gunzip, and untar the MP software. Pre-compiled binary is available for Linux. For Solaris, cd to the tools/src directory and do the following:

```
% mkdir ../bin
% make –f Makefile.sun all
```

On FreeBSD, copy the file <shake_home>/util/Makefile.bsd to <metadata_home>/tools/src, then do:

```
% make –f Makefile.bsd all
```

In previous versions of ShakeMap, mp was installed (or linked to) in <shake_home>/bin, however in the current version, the path to mp is configured as a parameter in *genex.conf*.

3.2.11 Zip

Zip is not required, but almost all installations will want to install it. It is required if you wish to make GIS shape files (using the '-shape' flag to 'genex'), or if you use the '-zip' option with 'genex' to compress your PostScript output (recommended to save space and transfer time to your web servers).

Zip allows the creation of compressed archives. It may be downloaded from http://www.info-zip.org (though, again, FreeBSD users can find it in the ports collection). Once you have installed *zip* on your system, there is a configuration parameter *zip* in 'genex.conf' that should be given the full path to the *zip* executable. Again *zip* is only required if *genex* is run with either the *–shape* option or the *–zip* option, but almost everyone will want to use both of those options.

3.2.12 Ssh

The secure shell, *ssh*, should be installed if you intend to transfer ShakeMap files via the 'scp' protocol. This is currently required, for example, if you will be transferring your web pages to the USGS servers. If *ssh* is not available on your system, please see your system administrator – he or she will want to make sure the installation is done correctly and in accordance with your network security policy.

3.3 Installing the Software

3.3.1 Installing and Configuring MySQL

Download MySQL from www.mysql.com. Binary distributions are available for Linux, Solaris 8, 9, and 10. If you are using an earlier version of Solaris, you may have to get the source and do a compile or you can get a pre-compiled, though older, version of MySQL from www.sunfreeware.com. If you are using FreeBSD, MySQL is, as usual, found in the ports collection and installation is almost trivial. We are using versions 5.0.18 and 4.0.21, though newer versions will probably work, as well.

We will not describe the MySQL installation process. Extensive documentation is available both online and in the distribution. You will need to get the MySQL server (mysqld) running, and set up an init script to start the server when the machine boots. Be especially careful to follow the instructions for setting a root user password and making sure your MySQL server is secure. You will be asked to do something like:

```
% cd /usr/local/mysql
% ./bin/mysqladmin -u root password 'your_root_password'
```

or:

```
% ./bin/mysql -p
Password:
(give an empty password)
```

```
...
mysql> set password for
    -> root@your_machine=PASSWORD('your_root_password');
```

(The following instructions assume that your MySQL server is running on the same machine that you run ShakeMap. This configuration is not required; you may run MySQL on another machine, but you will have to modify some of the commands given below to include a host name. See the MySQL documentation for more information. Also, keep in mind that your ShakeMap system will only be as reliable as the combined reliability of these two machines (i.e., consider providing backup power for both machines, their LAN router, etc.).)

The first step is to create a database and a user. Connect to the MySQL server as root. To connect and be prompted for a password:

```
% mysql -u root –p
Password:
(type your password and hit 'return')
…
mysql>
```

Now establish the shake database (we call it 'shakemap,' but you can call it anything you want as long as that is the name you use throughout the installation and configuration process):

```
mysql> create database shakemap;
```

Now give the users permission to modify the table. Here we give the user 'shake' (mysql password 'shake_password') the needed permissions:

```
mysql> grant select,insert,update,delete,create,drop,alter
    -> on shakemap.*
    -> to shake@localhost
    -> identified by 'shake_password';
Query OK, 0 rows affected (0.00 sec)
```

Below we have listed the above lines in a format that makes them easy to copy-and-paste into MySQL:

LINES TO CUT-AND-PASTE:
```
grant select,insert,update,delete,create,drop,alter
on shakemap.* to shake@localhost identified by 'shake_password';
```
END LINES TO CUT-AND-PASTE (don't forget to change the password...)

If you are going to be backing up and restoring the database (see Section 3.4.3.5), you will also want to grant the 'shakemap' user the 'lock tables' privilege.

Also create a user 'admin' to do backups:

```
mysql> grant select on shakemap.* to admin@localhost;
Query OK, 0 rows affected (0.00 sec)
```

LINES TO CUT-AND-PASTE:
```
grant select on shakemap.* to admin@localhost;
```
END LINES TO CUT-AND-PASTE

You may wish to create databases for other users, as well. Simply create a separate database for them, and then modify the above command to use the new username and database. For example:

```
mysql> create database jims_database;
...
mysql> grant select,insert,update,delete,create,drop,alter
    -> on jims_database.*
    -> to jim@localhost
    -> identified by 'jims_password';
```

LINES TO CUT-AND-PASTE:
```
grant select,insert,update,delete,create,drop,alter
on jims_database.* to jim@localhost identified by
'jims_password';
```
END LINES TO CUT-AND-PASTE (don't forget to change the username and password...)

The other users will have to configure their 'mydb.conf' and 'password' files accordingly, and can then use the included programs to create the tables and convert their old 'shake_flags' and 'earthquake' databases. Note there does not have to be direct correspondence between system usernames and MySQL usernames. Multiple users can share the same MySQL database either through a shared MySQL username, or individual MySQL usernames that all have permission to access the database.

For an explanation of the way ShakeMap uses the database and tables, see section 0, "Running ShakeMap," below.

3.3.2 Installation and Upgrade

Because the ShakeMap code is frequently updated, and the update method is through a Subversion 'svn update', many operators have found it worthwhile to establish an installation directory (or system) separate from their production system. The installation directory can be used to obtain the updates, make any necessary changes to configuration files, add any region-specific code, and test, while not disturbing the production system(s). Once the operator is satisfied that the update is working, (s)he can use 'rsync' (or a similar method) to push the ShakeMap directory tree to the production system. (And then, of course, test it again.) If you use multiple machines, it will save a lot of time and headaches if they are set up and configured exactly the same.

To begin, install the software packages and modules described in the section "

System and Software Requirements" above. If you are upgrading, there may be some new modules, and some of the existing modules may need to be updated.

For the installation of ShakeMap you will be working in an installation directory of your choice, which we will refer to as <shake_home>. You will then need to check out the latest version of the ShakeMap code:

```
% cd <shake_home>
% svn checkout
https://vault.gps.caltech.edu/repos/products/shakemap/tags/release-3.5/
local_directory_name
```

(Note that the "% svn checkout" line above is wrapped around to three lines in this document. When you enter that command on your computer all three lines above should be on the same line.) No password should be necessary. Subversion will check out all of the files in the ShakeMap directory tree. Table 3.2A provides a description of each of the top-level directories and Table 3.2B lists some of the more important subdirectories.

Now you will create a version of ShakeMap that is customized for your computing system. To do this (on Linux and Solaris):

```
% cd <shake_home>/install
% make
```

On FreeBSD, do:

```
% gmake INSTALL=/usr/bin/install
```

When you do this step, you may get some warnings about not being able to 'cd' to certain directories. These are harmless and should be ignored.

(In the instructions that follow we will use *make*, for which the FreeBSD users should substitute *gmake* unless their GNU make is installed as, or aliased to, 'make.')

Edit the file '<shake_home>/include/macros.'

The file *<shake_home>/include/macros* sets the paths to many of the required software packages as well as flags for running various programs. It tries to find the correct path, and guesses if it cannot, but you should review and verify its choices because it often guesses badly.

Note: When configuring your macros files, please be aware that the default paths for some parameters (especially paths to GMT and NetCDF libraries and include files) may not be correct for your system. Make sure you double check all the assignments in the macros file.

Next, issue the following commands:

```
% cd <shake_home>
```

```
% make all
```

Make outputs to the screen any errors and any configuration files that must be edited. Table 3.2C describes some additional top-level directories that will exist following this last step.

The next step in installing ShakeMap is to customize for your specific geographic region. To do this you will need to install a number of data files, and modify the configuration files in the directory '<shake_home>/config.' More information about the customization process can be found in section 3.4, Customizing ShakeMap; complete the customization process described there before proceeding with this section.

If this is a new install or first time upgrade to ShakeMap V3.0+, it will be necessary to create tables in the MySQL database. Once you have set up your configuration files, making the tables is easily accomplished:

```
% cd <shake_home>/bin
% ./mktables
```

This process will not destroy the tables if they already exist; to do that, connect to MySQL and issue the proper "drop table" commands. Errors in this program are not usually fatal: if one or more tables already exist, the program will complain, but will continue and make any tables that do not yet exist.

Once the config files have been edited, the final step for installation is to create the web products and put them on the web server, assuming you wish to use the ShakeMap generated web pages. To do this:

```
% cd <shake_home>/lib
% make web
% cd <shake_home>/bin
% ./transfer -permweb
```

Check that the transfer was successful. You will probably need to run and transfer an event before the web pages will work properly.

As noted above, it is probably best to do all of the setup, configuration, and testing on a non-production system, and then use something like 'rsync' to push the configured software to your production directory (or system).

Note that within the ShakeMap <shake_home> directory the subdirectory 'data' will contain all the event data and intermediate files as well as the final products to be transferred. Depending on the number of events, and the resolution of your grid and topography files, this directory can grow to be quite large. If disk space is limited on the install partition, the 'data' directory should be placed on a larger partition and a link to it (called 'data') should be made from the install directory. E.g.:

```
% cd $SM_HOME
```

```
% rmdir data
% ln -s /bigdisk/shake_data data
```

3.4 Customizing ShakeMap

3.4.1 Region-Specific Files

There are a number of region-specific files that you will need to create (see Table 3.3A and Table 3.3B). You should give these files names different from those in the distribution and in the configuration files or they will be overwritten every time you upgrade. Most of these files are part of the configuration defined in 'mapping.conf' and 'grind.conf.' See the configuration files themselves for more documentation.

3.4.2 Configuration Files

In the directory <shake_home>/config you will find a number of configuration files. It is important to read the documentation within these files as they provide most of the information necessary to customize ShakeMap to your particular environment. Table 3.3C lists the ShakeMap programs and the configuration files upon which they depend. All of the programs also depend on 'mydb.conf' to access the MySQL database. More discussion of shake.conf and mysql.conf can be found in the section "Running ShakeMap."

When editing configuration files, please note that the default values (as described in the documentation for some parameters) may not be the same as the value assigned to the parameter by default within the configuration file itself. The assigned value is the recommended value, the documented default is only used if no assignment is made, and may no longer be the recommended value (but may have been retained for reasons of backward compatibility).

When upgrading please note: From time to time we make changes to programs that require changes to config files. These changes must be merged with the config files that the user may have modified in customizing his/her version of ShakeMap. This is a non-trivial problem, and our solution is a bit simplistic. The merging consists of inserting the user's potentially changed config statements as comments into the new config file. The user may then go through the file and select which config statements are appropriate. This process takes a few minutes, but is fairly easy. Except in the case of 'transfer.conf,' which turns into a mess when it is changed. In this case it is often easier to clean out the destinations and file lists in the new config, then go to the backup file 'transfer.conf.BAK' (always made to keep a safe copy of the user-modified config files around) and just cut and paste your old destinations and file lists back into the new config file.)

3.4.3 Vs30 Grids

If you will require ShakeMap to apply site amplification factors to your ground motion maps, as most regional operators do, you will need to generate a grid of $V_{S_{30}}$ (hereafter, Vs30) values to cover your region of interest. Ideally, this grid will be at the same resolution as your output maps, as configured in *grind.conf*. You will specify this file with the *qtm_file* macro in *grind.conf*.

Obtaining the requisite Vs30 file is the responsibility of the network operator. If this proves difficult, one can approximate Vs30 from topography using the Allen and Wald (2008) approach, which is built into the program topo2grd. If *grind* is run with the *–qtm* flag, but no Vs30 grid is specified with the *qtm_file* macro in *grind.conf*, then *grind* will attempt to build the Vs30 file on the fly using Digital Elevation Model (DEM) files. To use this approach, one needs to set the DEMDIR macro in *include/macros* to point to a DEM directory. You can get DEM tiles from http://www.dgadv.com/srtm30/ or other places by Googling "SRTM30."

The topo2grd program requires the DEM files to be in a fairly specific format. We use a (Perl) command like:

```
my $command = "$gmtdir/xyz2grd $file -G$section.grd -R$lon/$lat
-I30c -Ddeg/deg/m/1/0 -L -F -ZTLhw -N-9999 -V";
```

to convert the DEM file to a GMT grd. The program topo2grd can then cut a chunk out of this file and convert it to Vs30.

None of this is really intended to be part of the normal ShakeMap distribution. topo2grd exists for the use of the global ShakeMap group at NEIC, so if you are going to try to make it work, be prepared to do some hacking on your own. You're much better off making a static Vs30 GMT grid file for your region and configuring its use in *grind.conf*.

3.4.4 Site Amplification Tables

Running *grind* with the *–qtm* flag will cause the program to apply site amplification to the computed ground motions. Many GMPEs have built-in site amplification functions. Running *grind* with the *–gmpesc* flag will apply the selected GMPE's native amplifications using the Vs30 grid discussed above. If the user does not specify *–gmpesc*, *grind* will use one of two more general approaches to site amplification, both based upon Borcherdt (1994):

1) The default approach uses the table <shake_home>/lib/sitecorr/Borcherdt94.dat (or a similar file created by the user) to specify the site amplification factors. In the case of Borcherdt94.dat, the values were taken directly from Table 2 of Borcherdt (1994). The table provides m_a and m_v (the short- and mid-period factors) for the equations (Eqns. 7a and 7b of Borcherdt (1994)):

   $$F_a = (v_0 / v)^{m_a}$$
   $$F_v = (v_0 / v)^{m_v}$$

   The table provides the factors for various input amplitudes. Amplitudes below the lowest specified receive the factor for the lowest amplitude, amplitudes greater than the highest receive the factor for the highest, and the factors for amplitudes in between are linearly interpolated between the factors for the bracketing amplitudes. v_0 is the reference velocity – for ShakeMap this is *smVs30default* as set in *grind.conf*, and v is the site specific Vs30. The format of the table is discussed more in *<shake_home>/src/lib/Borcherdt94.pm* and

<shake_home>/lib/sitecorr/Borcherdt94.dat serves as an example.

2) If the operator specifies *–oldsc* along with *–qtm*, the system will attempt to use an old-style Borcherdt table, as used by ShakeMap prior to September 2011. The structure of this table is described in the source file *<shake_home>/src/lib/SiteCorrGrd.pm* and an example may be found in *<shake_home>/lib/sitecorr/site_corr_cdmg.dat*. We note that this earlier approach used hard cutoffs in both Vs30 and input amplitude, unlike the smooth interpolation of approach #1, above. Note also that the amplification factors specified are the actual F_a and F_v, not m_a and m_v, as in #1, above.

3.4.5 MMI Site Amplification Tables

Earlier versions of ShakeMap required the operator to make MMI site correction files with the program makeMMIsiteTable. It is no longer necessary to do so – the MMI site corrections are computed on the fly using the Borcherdt table and the selected GMICE.

3.4.6 Passwords

You will need passwords to access a database through db.conf or mydb.conf (or for *transfer* using ssh or ftp). To set up a password file:

```
% cd <shake_home>
% mkdir pw
% chmod og-rx pw
% cd pw
```

Create or copy your passwords file to 'passwords' and make this file readable by the ShakeMap user only. For an explanation of the format of this file, see '<shake_home>/src/lib/Password.pm.' Also see the section "Running ShakeMap," below for more on 'mydb.conf.' In general, the format for *ssh* and *FTP* passwords is:

```
<machine> <username> <password>
```

And for database access the format is:

```
<dbname> <username> <password>
```

where the substitutions for "dbname" and "username" above should exactly match the strings in the database configuration file.

Be aware that this approach leaves unencrypted passwords in a plain text file. This approach is not particularly secure and we welcome the contribution of anyone wishing to improve upon it.

3.4.7 Web Pages

You may also wish to make changes to the web pages. We have tried to include much of the region-specific data in the web.conf file, but there may be additional customizations needed. Please keep track of your changes and let us know so that we can add common items to the configuration file. The web pages and templates can be found in <shake_home>/lib/genex/web/.

We have more or less stopped development of the web page code within ShakeMap. The USGS Earthquake Program's ShakeMap site is now much more sophisticated and modern-looking. See http://earthquake.usgs.gov/eqcenter/shakemap/list.php?y=2009&n=sc for an example of the new page layout.

There are plans underway to create web templates that will allows regional networks to obtain similar look and behavior to the USGS Earthquake Program's ShakeMap site yet simplify local customization. Such a template will also entail providing a server-side PHP indexer to sort, update and search real-time, archived, and Scenario ShakeMaps.

3.4.8 Automation

Because each regional network is different, automation is left to you. Currently code exists to automate generating ShakeMaps from two types of systems: 1) a database running the NCEDC/SCEDC schema (as in southern California and Berkeley), and 2) *earthworm* running with the Oracle database. If you are using either of these systems you will be able to adapt current code.

If you do not use one of the above data acquisition systems, you will need to first generate code that will process data in near-real-time. The output of this processing should include peak horizontal acceleration, peak horizontal velocity, and 5%-damped pseudo-spectral peak horizontal acceleration (0.3, 1.0 and 3.0 second periods) for all horizontal component data. This information along with station information must be written into ShakeMap compatible XML files with filenames that end in "_dat.xml." The event information – latitude, longitude, depth, and magnitude – should be written to a second ShakeMap compatible XML file – "event.xml". See the section on "ShakeMap XML Input," below, for a discussion of these file formats. Examples of data and event XML files can be found in the distribution in the directory <shake_home>/data/9583161/input.

Next, you need a program to watch when these files are made, then copy them to the ShakeMap input directory and start ShakeMap. This could, of course, be the same program that creates the files.

The distribution includes a program called 'queue' and its associated configuration file 'queue.conf' that may be of interest. *queue* waits for an alarm announcing an event or cancellation (see the programs *shake_alarm* and *shake_cancel*) and then takes appropriate action depending on its configuration (i.e., given a location and magnitude it will either kick off a run of ShakeMap or ignore the event). It can prioritize and queue multiple events, and schedule events for automatic reprocessing at user-defined intervals. The program accesses a database to retrieve information on the earthquake, but should be fairly easy to adapt to other systems.

If you develop a program (or modify *queue*) that you think might be of interest to other ShakeMap installations, please let us know and we will include it in a future release.

3.4.9 Overriding the Automation (input/source.txt and input/timestamp_text.txt)

Because most ShakeMap installations automatically generate XML input files and dump them in the *input* directory, manual changes made by the operator to the *event.xml* file will generally be overwritten by the next automatic run. We therefore provide a mechanism by which the operator may override or supplement any of the event-specific data in *event.xml*. The operator may add an optional file to an event's input directory called *source.txt*. The structure of the file is one parameter per line, in the form parameter=value. In particular, the operator may specify the source mechanism with "mech" (this is the equivalent of the "type" attribute in event.xml), which may be one of "RS," "SS," "NM," or "ALL" for reverse slip, strike slip, normal, and unspecified mechanisms, respectively. Some, but not all of the GMPE modules use some or all of these values. They should generally do something sensible if they don't know what to do with a particular mechanism. The operator may also specify "zone" as one of "interface" or "intraslab" (which are currently only used by the Youngs97 GMPE, and will themselves be overridden by using one of Youngs97_interface or Youngs97_intraslab). Other interesting parameters are "aftershock" (which may be 1 or 0 and is used by CY08), "rake" (the rake angle of the rupture, used by CY08 when the mechanism is RS), and "region" (which may be "SCal" or "CCal" for CY08 when the magnitude is < 5.5 (and which will be overridden by the choice of either CY08_SMM_SCal or CY08_SMM_CCal), or "CA" (California) or "CEUS" (central and eastern U.S.) for AW07 (also overridden by either AW07_CA or AW07_CEUS). Any of the other source parameters may also be set: eid, string, year, mon, day, hour, minute, sec, timezone, lat, lon, depth, mag, pga, pgv, psa03, psa10, and psa30, but it isn't apparent why one would do so. Blank lines and lines beginning with '#' (i.e., comments) are ignored.

The operator may also place an optional file called *timestamp_text.txt* in an event's input directory. A line of text in this file will be printed on timestamp line on the maps, following the standard timestamp. There is limited space for the text -- about half a line -- and is intended to provide some region-specific data, as needed. Be careful with special characters and quotation marks, as they can cause the program to abort. If you use quotes, make sure they are paired. Make sure you test your text before adding it to any automatic processing.

3.4.10 Ground-Motion Prediction Equation (GMPE), Intensity Prediction Equation (IPE), and Ground-Motion/Intensity Conversion Equation (GMICE) Modules

The selection of the proper GMPE, IPE, and GMICE modules for a particular region or earthquake is beyond the scope of this document. Below we provide summary tables listing the available modules. See also *grind.conf* for information about configuring the system to select GMPEs and IPEs based on earthquake magnitude and depth. For geographic customizations, see the section on Zone Config.

Custom GMPE modules may be needed for some regions. If you are going to develop a module, the interface must be modeled after the ones found in *<shake_src>/src/lib/GMPE* (e.g., Small.pm). It will probably be easiest to select a module from the table that is closest in behavior to the new GMPE, copy it, and edit it as necessary. Once the module has been written, it will need to be added to the list of modules in the *Makefile*. A 'use' line for the module should

also be added to the file *<shake_src>/src/lib/GMPE.pm*. Then run 'make.' You will then be able to configure *grind.conf* to use the new module.

Similar procedures apply for writing a new Intensity Prediction Equation, though it should be placed in the directory *<shake_src>/src/lib/IPE*, and the 'use' line should be added to *<shake_src>/src/lib/IPE.pm*.

Similar procedures also apply for creating new Ground Motion/Intensity Conversion Equation modules, but substituting "GMICE" for "GMPE" or "IPE" above.

Similar procedures also apply for creating new modules for correcting amplitudes for basin depth, but substituting "Basin" for "GMPE," etc.

With ShakeMap V3.5, module writing has become a bit more complicated. We anticipate more extensive documentation to be available shortly. Look for it in the distribution's doc directory.

Notes on specific GMPEs:

BA08 – Includes the Atkinson and Boore (2011) modifications for small- to moderate-magnitude earthquakes.

CY08, **CY08_SMM_SCal**, **CY08_SMM_CCal** – For events $M > 5.5$, these modules function identically. For events $M \leq 5.5$, CY08_SMM_SCal uses the 2009 paper's small- to moderate-magnitude coefficients for southern California. CY08_SMM_CCal uses the central (northern) California coefficients. To use the more general module "CY08" for both regions, you may set the "region" attribute in the "earthquake" tag in the event XML, or set a "region=" field in the source.txt file. Acceptable values are "CCal" and "SCal". The default is "SCal".

Garcia05 – Does not have GMPE-native site amplification terms. Use of –gmpesc or –nativesc with grind will cause the program to abort.

Youngs97, Youngs97_(interface | intraslab) – The two tectonic environment-specific modules instantiate an instance of the general module, Youngs97, with the coefficients for their respective regimes. The general module can be used by specifying "crustal" or "interface" in the "zone" attribute in the "earthquake" tag of the event XML, or setting a "zone=" field in the source.txt file in the event's input directory. There is no default zone for Youngs97 – the tectonic regime must be specified or the program will terminate.

Zhao06, Zhao06_(crustal | interface | intraslab) – The three tectonic environment-specific modules instantiate an instance of the general module, Zhao06, with the coefficients for their respective regimes. The general module can be used by specifying "crustal," "interface," or "intraslab" in the "zone" attribute in the "earthquake" tag of the event XML, or setting a "zone=" field in the source.txt file in the event's input directory. The default is "crustal."

Table 1. Ground-Motion Prediction Equations (GMPEs) in ShakeMap 3.5

Module Name	Reference	Magnitude Range	Distance Range (km)	Distance Metric	PGV	PSA	Uncertainty Type	Site Term	Mech[4]	Region
AB06_ENA_BC	Atkinson & Boore (2006)[5]	≥ 4.0	0 – 1000	R_{Rup}	Yes	Yes[6]	Spatially constant[7]	Yes[8]	N/A	Eastern North America
AkkarBommer07	Akkar & Bommer (2007, 2007b)	5.0 ≤ M ≤ 7.6	5 – 100	R_{Rup}	Yes	Yes[9]	Spatially constant	Yes[10]	RS, NM, ALL	Europe
BA08[11]	Boore & Atkinson (2008)	3.0 ≤ M ≤ 8.0	0 – 200	R_{JB}	Yes	Yes	Spatially constant[12]	Yes	SS, RS, NM, ALL	NGA Active Tectonic
BJF97	Boore, Joyner, Fumal (1997)	5.0 ≤ M ≤ 7.4	0 – 80	R_{JB}	No[13]	Yes	Spatially constant	Yes	SS, RS, ALL	Western North America
Boatwright03	Boatwright, et al. (2003)	3.5 ≤ M ≤ 7.1	0 – 300	R_{Hypo}	Yes	No[14]	Spatially constant	Yes[15]	N/A	Northern California
CY08 CY08_SMM_CCal CY08_SMM_SCal	Chiou & Youngs (2008), Chiou, et al. (2009)	3.0 < M ≤ 7.7	0 – 200	R_{Rup}[16]	Yes	Yes	Spatially variable[17]	Yes	SS, RS, NM	NGA Active Tectonic, CA for SMM

[4] Allowable source mechanisms for GMPEs that differentiate. SS = strike slip; RS = reverse slip; NM = normal; ALL = unspecified; N/A = module ignores source mech parameter.
[5] Updated with the Atkinson & Boore (2011) modifications.
[6] Module uses 0.315 sec coefficients for 0.3 sec PSA, and 3.13 sec coefficients for 3.0 sec PSA.
[7] No inter-/intra-event differentiation; constant sigma for all frequencies.
[8] Uses site terms from BA08.
[9] Relation produces spectral displacement, module converts to SA.
[10] Relation provides amplification terms for "soft soil," and "stiff soil," which are taken to be Vs30 < 360 m/s and 360 ≤ Vs30 < 760, respectively.
[11] Updated with the Atkinson & Boore (2011) modifications.
[12] Inter-event uncertainty changes with specified/unspecified fault type.
[13] Module uses PGV from Joyner & Boore (1988).
[14] The module calls BJF97 for PSA.
[15] Uses BJF97 site amplification term.
[16] Hanging wall term uses R_{JB} and a custom distance measure, R_X.
[17] Magnitude, site, and amplitude dependent.

Module Name	Reference	Magnitude Range	Distance Range (km)	Distance Metric	PGV	PSA	Uncertainty Type	Site Term	Mech[4]	Region
Campbell2003	Campbell (2003; 2004)	≥ 5.0	0 – 1000	R_{Rup}	N&H'82	Yes	Spatially constant[18]	No[19]	N/A	Eastern North America
Garcia05	Garcia et al. (2005)	$5.2 \leq M \leq 7.4$	0 – 400	R_{Rup}, R_{Hypo}	Yes	Yes	Spatially constant	No	N/A	Mexico intra-slab
HazusPGV	Boore, Joyner, Fumal (1997)	$5.0 \leq M \leq 7.4$	0 – 80	R_{JB}	N&H'82[20]	Yes	Spatially constant	Yes	SS, RS, ALL	Western North America
Kanno2006	Kanno, et al. (2006)	≥ 5.5	0 – 500	R_{Rup}	Yes	Yes	Spatially constant	Yes	N/A	Subduction, Active Tectonic
MA2005	Motazedian & Atkinson (2005)	$3.0 \leq M \leq 8.0$	2 – 500	R_{Rup}	Yes	Yes	Spatially constant	No[21]	N/A	Puerto Rico
PP04	Pankow & Pechmann (2004)	$5.0 \leq M \leq 7.7$	0 – 100	R_{JB}	Yes	Yes[22]	Spatially constant	Limited[23]	N/A[24]	Extensional Tectonic
Small	Quitoriano	$3.0 \leq M \leq 5.2$	0 – 200	R_{JB}	Yes	Yes	Spatially constant[25]	Yes	N/A	Active Tectonic (CA)
Youngs97 Youngs97_interface Youngs97_intraslab	Youngs (1997)	$5.2 \leq M \leq 8.0$	0 – 300	R_{Rup}	N&H'82	Yes	Spatially constant[26]	No[27]	N/A	Subduction: interface, intraslab[28]

[18] No inter-/intra-event differentiation.
[19] Module uses site corrections from AB06_ENA_BC.
[20] PGV from PSA 1.0 sec, via Newmark & Hall 1982 conversion.
[21] Module uses site correction terms from HazusPGV (i.e., BJF97) module.
[22] Module uses 2.0 second coefficients for 3.0 second PSA.
[23] "Soil" and "rock" corrections.
[24] Assumed to be normal faulting.
[25] No inter-/intra-event differentiation.
[26] No inter-/intra-event differentiation. Magnitude dependent.
[27] Module requires operator to configure and use Borcherdt-style site tables.
[28] Rupture type need only be specified for Youngs97, the *_interface and *_intraslab modules have the rupture type hardwired.

Module Name	Reference	Magnitude Range	Distance Range (km)	Distance Metric	PGV	PSA	Uncertainty Type	Site Term	Mech[4]	Region
Zhao06 Zhao06_crustal Zhao06_interface Zhao06_intraslab	Zhao, et al. (2006)	$5.0 \leq M \leq 8.3$	0 – 300	R_{Rup}	N&H'82	Yes	Spatially constant	Yes[29]	SS, NM, RS[30]	Japan: crustal, interface, slab

[29] Four Vs30 site classes are recognized. Terms do not account for nonlinearity of amplification.
[30] Only crustal events are sensitive to rupture type, subduction events are not. Strike-slip and normal events are not differentiated. The default mechanism is SS.

Table 2. Intensity Prediction Equations (IPEs) in ShakeMap 3.5

Module Name	Reference	Magnitude Range	Distance Range (km)	Distance Metric	Uncertainty Type	Site Term	Region
AW07_CA AW07_CEUS	Atkinson & Wald (2007)	$2.0 \leq M \leq 7.9$	0 – 500 0 – 1000	R_{JB}	Spatially constant[31]	No	California Central and Eastern U.S.
TA12_mmi (was: TA09_mmi)	Allen, et al. (2012)			R_{Rup} & R_{Hyp}[32]	Distance dependent[33]	Yes	Active Tectonic
DefaultIPE	-	-	-	-	-	-	Behavior is GMPE- and GMICE-specific

Table 3. Ground-Motion/Intensity Conversion Equations (GMICEs) in ShakeMap 3.5

Module Name	Reference	Magnitude Term	Distance Term	Reversible[34]	PSA	Region
AK07	Atkinson & Kaka (2007)	Yes	Yes	No	Yes	California, Central and Eastern U.S.
DC11_CA DC11_ENA	Dangkua & Cramer (2011)	Yes	Yes	No	Yes	California (DC11_CA) Eastern U.S. (DC11_ENA)
FM10	Faenza & Michelini (2010)	No	No	Yes	No	Italy (MCS)
WGRW11	Worden, et al. (2012)	Yes	Yes	Yes	Yes	California
Wald99	Wald, et al. (1999)	No	No	No	No	California

[31] No inter-/intra-event differentiation.
[32] Uses R_{Rup} if fault is defined, R_{Hyp} otherwise.
[33] No inter-/intra-event differentiation.
[34] "Reversible" here means that the GMICE was developed with reversibility in mind. All of the modules may be used in a reversible manner.

3.5 Running ShakeMap

ShakeMap consists of a series of programs (refer to Table 3.3C) that, when run sequentially, produce the desired output and transfer it to its destination. All of the programs will print documentation when run with the '*-help'* flag, and most of them have an associated configuration file that controls the behavior of the program (again, see Table 3.3C, and the section "Configuration Files," above). The configuration files contain documentation that explains the various required and optional parameters.

3.5.1 Data Directory Structure

Before running ShakeMap you must collect some data. These data are stored in the *data* directory which, as mentioned elsewhere, can become quite large. Put it somewhere with lots of space and link to it from your distribution directory. Each event is stored in its own sub-directory named with the event ID, be it a number or a text string (i.e., '<shake_home>/data/<event_id>'). This event ID must be the same as the one found in the file containing the earthquake information – 'event.xml'. Within each event directory a number of subdirectories are created (Table 3.4). ShakeMap will create all of these directories except 'raw' and 'input'.

3.5.2 Creating the Maps

Once the ShakeMap software is installed and configured, creating a ShakeMap is simple. First, *cd* to '<shake_home>/bin' (e.g., '/opt/ShakeMap/bin'), then execute *shake*:

```
% ./shake -event <event_id>
```

This will run the pre-configured set of programs as specified in 'shake.conf.' If you would like a little more information about the progress of the run, use the *-verbose* flag to *shake*.
It is not always appropriate or necessary to run all of the programs. For instance, when running a historic event, or an event not otherwise in the database, *retrieve* will probably fail, causing *shake* to abort. One possibility is to reconfigure 'shake.conf' to skip the unnecessary program(s). Another option is to use the *-dryrun* flag:

```
% ./shake -event <event_id> -dryrun
```

Which will produce output showing the programs that *shake* would run (and their options) without actually running them:

```
/opt/ShakeMap/bin/retrieve -event 9108645
/opt/ShakeMap/bin/pending -event 9108645
/opt/ShakeMap/bin/grind -event 9108645 -qtm -boundcheck
     -lonspan 4.5 -psa
/opt/ShakeMap/bin/mapping -event 9108645 -timestamp
/opt/ShakeMap/bin/shakemail -event 9108645
/opt/ShakeMap/bin/tag -event 9108645 -mainshock
/opt/ShakeMap/bin/genex -event 9108645
/opt/ShakeMap/bin/print -event 9108645
```

```
/opt/ShakeMap/bin/transfer -event 9108645 -www -ftp
```

You may then run the programs you choose and ignore the others. For instance, if you were to make a change to your user-supplied estimates, you might just run *grind* and *mapping* and then look at the plots as PostScript (the .ps files in the '<shake_home>/data/<event_id>/mapping' directory). You could then run *genex* and look at the JPEGs. Or also run *transfer* and look at the images on your web site.

3.5.3 The Processing Sequence, *shake*, and Versioning

Of course, it is never that simple. And even if it were, there are reasons for having a better understanding of the system. Here, then, is more detailed information on configuring 'shake' and on the way the versioning system works.

3.5.3.1 shake.conf

The program 'shake' is the main ShakeMap program. Its job is to run a series of other programs in a specified order, possibly calling the programs with invocation flags that vary with magnitude. The program can also be told to call certain programs only the first time a given event is processed. Run *shake -help* to see other options.

At this point, it is recommended that you read 'shake.conf' (in '<shake_home>/config') to get a basic idea of what is available. The default configuration is probably about right for most installations. Some of the parameters ('once_only,' 'no_dep,' 'cancel,' and 'scenario_skip') probably won't need to be changed unless you add a new program to the processing sequence with the 'program' parameter (and maybe not even then).

In addition to the ShakeMap programs described in Table 3.3C, there is a command 'sleep' that may be used anywhere in the shake.conf processing sequence (defined by the sequence of 'program' lines). The usage is:

```
program : sleep N
```

where 'N' is an integer number of seconds for *shake* to wait before executing the next program. This command may be useful to delay the call to *retrieve* for a few seconds to allow databases to be updated or to deal with other system-specific timing issues. The sleep command may be used any number of times in the processing sequence.

'shake.conf' is also the configuration file for the program 'cancel,' which effectively undoes the effects of *shake*, removing the event from the system, sending cancellation notices, and rebuilding the web pages to reflect the absence of the cancelled event.

3.5.3.2 The Processing Sequence and shake.conf

ShakeMaps are not always automatically generated. Frequently, manual intervention is necessary or desirable, and we often run one or more of the programs repeatedly until we are satisfied with the results. For example, the automatic processing sequence might go something like this:

retrieve → *pending* → *grind* → *tag* → *mapping* → *genex* → *shakemail* → *transfer* →

setversion

But after the automatic run, we might wish to change the map dimensions or centering by changing the options to *grind*. Our manual sequence might look like this:

grind → *mapping* → *genex* → *transfer*

We might run the *grind* → *mapping* pair several times in succession until we are satisfied with the results. Satisfied, we then run *transfer* to update the web pages with our new maps. Previous versions of ShakeMap would happily do this, despite the fact that we forgot to run *genex* and, as a result, some of our products (e.g., the PostScript maps) do not agree with others (e.g., the JPEG maps and shapefiles).

Starting with ShakeMap V3.0 we have introduced the idea of program dependency. Simply put, a program is considered to be dependent on the programs that precede it in the processing sequence, and it will not run unless the sequence is run in the proper order. For instance, in the above example, *transfer* would recognize that *mapping* had run more recently than *genex* and would abort with an error message explaining the problem.

Things of which to be aware:

1) The processing sequence is defined by the order of 'program' lines in 'shake.conf.'
2) A program that does not affect the performance of programs later in the sequence (i.e., later programs do not depend on its output) can be identified with a 'no_dep' line in 'shake.conf.' For instance, *shakemail* sends email to interested parties, but does not generate products that any programs later in the processing sequence depend upon. Thus, *shakemail* is declared 'no_dep.' When a later program (e.g., *transfer*) runs, it will not include *shakemail* in its investigation of the processing sequence. But *shakemail* itself will still require the programs that precede it to be run in sequence. Thus, if *shakemail* is run immediately after *mapping*, it will complain that *genex* has not been run.
3) You do not have to always start at the beginning of the sequence. Once an event has been run once, you can start anywhere in the sequence. You can jump in and re-run *mapping*. You can run it a bunch of times in a row. Then you can run *genex*. Then you can run *mapping* again. Then you can run *grind*. What you can't do is use out-of-date output.
4) Yes, it seems complicated. But it is actually simple. Assume the function T() returns the time a program, *P*, was most recently run. Assume that 'P_n' is the n^{th} non-no_dep program in the processing sequence. The software enforces the relation:
   $$T(P_1) < T(P_2) < \ldots < T(P_{n-1})$$
   with the provision that each of the n-1 earlier programs has run at least once. Okay, maybe it is a little complicated.
5) You can always force a program to run with the *-forcerun* flag, but there may be consequences.

So how does the system keep track of all this? By using the 'shake_runs' database table described in the next section.

3.5.3.3 Flags, Versions, and the MySQL Database

During the ShakeMap installation process you created a number of tables in your MySQL database. These tables keep track of the earthquakes ShakeMap has processed, the flags with which each program was run for each version of the maps, and provide functionality to support versions and the processing sequence integrity system described above.

The database tables in the shakemap database can be listed with *mysql*:

```
mysql> use shakemap;
Database changed
mysql> show tables;
+--------------------+
| Tables_in_shakemap |
+--------------------+
| earthquake         |
| server             |
| shake_lock         |
| shake_runs         |
| shake_version      |
+--------------------+
5 rows in set (0.00 sec)
```

The ‘server’ table contains information the ShakeCast system needs to connect to a server. The ‘earthquake’ table is very similar to the earlier CSV table of the same name:

```
mysql> describe earthquake;
+-----------+-----------+------+-----+---------+-------+
| Field     | Type      | Null | Key | Default | Extra |
+-----------+-----------+------+-----+---------+-------+
| evid      | char(80)  |      | PRI |         |       |
| name      | char(255) | YES  |     | NULL    |       |
| locstring | char(255) | YES  |     | NULL    |       |
| tabsol    | datetime  | YES  |     | NULL    |       |
| tzone     | char(8)   | YES  |     | NULL    |       |
| mag       | double    | YES  |     | NULL    |       |
| lat       | double    | YES  |     | NULL    |       |
| lon       | double    | YES  |     | NULL    |       |
| mainshock | char(20)  | YES  |     | NULL    |       |
| cluster   | char(80)  | YES  |     | NULL    |       |
+-----------+-----------+------+-----+---------+-------+
10 rows in set (0.00 sec)
```

This table is accessed and modified by a number of programs (*tag*, *genex*, *cancel*, etc.). Its primary purpose is to maintain a complete inventory of the events for which ShakeMaps have been made. Under rare circumstances you may have to edit this table (using SQL commands), so the following table describes the columns.

Table: Columns in the 'earthquake' table

Name	Description	Valid values
evid	The event identifier.	Any text string that forms a valid Unix filename, up to 80 characters.
name	The long, possibly descriptive name of the event; will be printed at the top of the maps.	Any text string up to 255 characters.
locstring	The location of the earthquake. If the name field is not specified (through the program 'tag'), this text will be used as the event name on the maps.	Any text string up to 255 characters.
tabsol	The date and time of the event in the format: yyyy-mm-dd hh:mm:ss	From 1000-01-01 12:00:00 AM to 9999-12-31 11:59:59 PM
tzone	The timezone of 'tabsol,' above.	Usually 'GMT,' but could be 'PST,' 'MDT,' etc.
mag	The earthquake magnitude.	Any valid magnitude.
lat	The latitude of the earthquake epicenter.	+90 to -90, north is positive, south is negative.
lon	The longitude of the earthquake epicenter	+180 to -180, west is negative.
mainshock	Value set by the program 'tag' to categorize the earthquake	Valid values include '', 'current,' 'historic,' 'scenario,' and 'invisible.'
cluster	If this event is part of a larger sequence, this field specifies the evid of the mainshock in the sequence. This may be useful for creating a special archive page for a particular sequence.	Any valid evid.

The table 'shake_lock' table is used to prevent multiple ShakeMap processes from operating on an event at the same time. Each ShakeMap program will acquire the lock before it begins processing, and will release the lock when it quits (or is killed).

```
mysql> describe shake_lock;
+---------+----------+------+-----+---------+-------+
| Field   | Type     | Null | Key | Default | Extra |
+---------+----------+------+-----+---------+-------+
| evid    | char(80) |      | PRI |         |       |
| program | char(80) |      |     |         |       |
| pid     | int(11)  |      |     | 0       |       |
| tepoch  | int(11)  |      |     | 0       |       |
+---------+----------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

The columns are: the event id, the name of the program, the process id of the locking process, and the Unix epoch time that the lock was acquired. Occasionally, a lock will be held when the locking process is dead or hung. The lock can be broken by 1) using the '*-forcerun*' flag to the

next program, or 2) calling the program 'unlock' with the event id of the locked event (this program will also optionally try to kill the locking process), or 3) if a lock is stale (more than fifteen minutes old), ShakeMap programs will automatically unlock the event and continue processing after issuing a warning message.

The 'shake_runs' table keeps track of the last run of each program for each version of an event. But first:

A Digression on Versioning

After a great deal of discussion and consideration, we decided that the most useful demarcation of a 'version' of a ShakeMap (which is really a collection of products) is the point at which the products are distributed to external destinations. In other words, we create a new version every time we run *transfer*, whether or not that version differs in any significant way from the previous version. (Models that assigned version numbers to each product based on its difference from the previous version of that product, while sexy, were ultimately found to be too complicated, unreliable, and unworkable. Consider, for example, a JPEG map that varies in no way from another map, except that the embedded processing date is different. Is that a different version? Some say "yes," some say "no." Plus, no one could come up with a compelling reason for defining versions this way. But our digression digresses…)

So how does this versioning system work? Let us assume that *transfer* has just run on an event and created version 'N' (if *transfer* has never run for this event, 'N' would be zero). We then run one of the other programs in the processing sequence. For instance, we run *grind* to change the "lonspan." The program will inspect the 'shake_version' table and determine that the most recent version of the event is version 'N.' *grind* will then declare itself to be working on version 'N+1.' It will check that the processing sequence is being honored, do its processing job, then insert some information about itself (its name, the current time and date, the version, and the flags with which it was invoked) in the 'shake_runs' table before exiting. If we were to run this program again, it would go through the same process, but when it found that a row already existed in the shake_flags table for that event/program/version combination, it would simply update the date/time and invocation flags. It would still be version N+1. We could run it twenty times and it would still be version N+1. We could then run *mapping* (version N+1) and *genex* (version N+1). We could go back and run *grind* some more (still version N+1), *mapping*, and *genex*. Finally, when we run *transfer*, the new version is declared complete, a new row is inserted in 'shake_version' for version N+1, and the products are transferred to the world. The next time a program in the sequence is run, it begins version N+2. And so on.

(In the situation where some programs were not run, the missing programs are inserted into the 'shake_runs' table with the new version number, but the date/time and flags of the previous version. For example, we could run *mapping*, *genex*, and *transfer*, without ever re-running *grind* (which is a valid thing to do – see the section on the Processing Sequence, above). When the new version was set, the system would copy the flags and time/date of the previous run of *grind*, but give it the new version number.)

By using this system, we have a complete record of the programs and their invocation flags for each version of the event that we transferred to the world. In conjunction with the judicious use

of the program 'setversion' (which will save a copy of the input data and the configuration files for an event in a version-specific directory) we can recreate any version of an event. Here is the a listing of a southern California event:

```
mysql> select program,flags from shake_runs where
evid='14007388' and version=4 order by lastrun;
+----------+-------------------------------------------------+
| program  | flags                                           |
+----------+-------------------------------------------------+
| retrieve |                                                 |
| grind    | -qtm -boundcheck                                |
| mapping  | -timestamp -notchecked -tvmap -itopo |
| genex    | -zip -metadata -shape shape                     |
| transfer | -www -ftp -push                                 |
+----------+-------------------------------------------------+
6 rows in set (0.01 sec)
```

By running these programs, with these flags, on the preserved input data and the preserved configuration files, we could re-create version 4 of this event.

Keep in mind:
1) *transfer* sets a new version unless you tell it not to with *-noversion*.
2) Versions can be created by *setversion*. *setversion* will also delete, modify, or query the version information for an event.
3) The default invocation of *setversion* (i.e., "*setversion –event* <event_id>") does nothing. Use the magnitude-dependent flags in 'shake.conf' to configure *setversion* to save the data for significant events without filling your disks up with data from a lot of magnitude 3.5 earthquakes.
4) *transfer* has a *-forget* flag that will prevent its flags from being saved in the database. This is useful for *cancel*, and *pending*, or if you are doing something unorthodox. *grind* also has a *-forget* flag. All of the programs probably should.

End of Digression

The 'shake_runs' table has the following structure:

```
mysql> describe shake_runs;
+---------+-----------+------+-----+---------+-------+
| Field   | Type      | Null | Key | Default | Extra |
+---------+-----------+------+-----+---------+-------+
| evid    | char(80)  |      | PRI |         |       |
| program | char(80)  |      | PRI |         |       |
| lastrun | datetime  | YES  |     | NULL    |       |
| version | int(11)   |      | PRI | 0       |       |
| flags   | char(255) |      |     |         |       |
+---------+-----------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Most of the columns are self-explanatory: the event id, the program name, the date/time of the last run, the version, and the invoking flags (sans the '*-event* <event_id>' and '*-verbose*' flags). Note that the primary key consists of (evid, program, version).

Version information is stored in the 'shake_version' table:

```
mysql> describe shake_version;
+---------+-----------+------+-----+---------+---------------+
| Field   | Type      | Null | Key | Default | Extra         |
+---------+-----------+------+-----+---------+---------------+
| evid    | char(80)  |      | PRI |         |               |
| version | int(11)   |      | PRI | NULL    | auto_increment|
| lddate  | datetime  | YES  |     | NULL    |               |
| comment | char(255) | YES  |     | NULL    |               |
+---------+-----------+------+-----+---------+---------------+
4 rows in set (0.00 sec)
```

The columns are obvious except for 'comment.' If the version was created by *transfer*, the comment will be "Automatic call from within transfer." If you use *setversion* to make the version, you can give a comment on the command line explaining your intent.

3.5.3.4 Passwords and mydb.conf

The configuration line for MySQL access in mydb.conf will look something like this:

```
database : mysql shakemap shake password
```

where you would substitute your database name for 'shakemap' and the username of the user running ShakeMap for 'shake.' E.g., 'jims_database' and 'jim' if user jim is experimenting with his own version of ShakeMap. See the section "Installing and Configuring MySQL" for instructions on giving jim his own database. If you are running MySQL on a remote machine, your config line will look something like this:

```
database : mysql database=shakemap;host=machine.domain.org
shake password
```

In the password file ('<shake_home>/pw/passwords,' by default), you will need a line:

```
shakemap shake <mysql_password_for_user_shake>
```

or, if you are using a remote database server:

```
database=shakemap;host=machine.domain.org shake
<mysql_password_for_user_shake>
```

with the obvious substitutions to make it work in your environment (or jim's). (Yes, the "database=shakemap…)" bit looks wrong, but the password module is comparing strings with what is found in "mydb.conf" and this is what is required to make it work.)

3.5.3.5 Backing up the MySQL database

Because we are maintaining a database, and because what we keep in our database is important, it is probably a good idea to do database backups on a regular basis. There are a number of ways to do this with MySQL, including logging every transaction in a way that lets you recreate the database after any failure. See the MySQL documentation for details if you would like to implement a more robust backup system than is described here.

The *mysqldump* program allows one to dump one's tables to a file as a set of SQL statements that can then be used to recreate the tables. For example:

```
% mysqldump --add-drop-table -u admin shakemap >
shakemap.sql
```

The file so created can then be used to restore the database (or to transfer the data to another system):

```
% mysql -u shake -p shakemap < shakemap.sql
Password:
```

Note that the user names and database name may need to be changed on your system. Also note that for *mysqldump* we use the 'admin' user that we created in the section "Installing and Configuring MySQL". This user does not need a password because its only SQL permission is SELECT, however it is necessary to grant the shakemap user the 'lock tables' permission to do the restore.

We have included a program 'mysqlbu' in the directory <shake_home>/util. This program performs the database dump, compresses the output and, optionally, copies the output to another machine for safekeeping. (The program contains hard-wired path and machine names, though, so you will have to modify it for your system.) 'mysqlbu' can be run daily – it will create a different file for each weekday. The program also prints an error summary that can be piped to a mail program. We run it with a crontab entry that looks like this:

```
0 2 * * * /home/shake/bin/mysqlbu | mail –t shake_admin
```

Which runs *mysqlbu* at 2:00 AM every day, and mails the status report to the user 'shake_admin.'

3.5.4 Special Issues When Running ShakeMap

3.5.4.1 A Note About Program Flags

Since ShakeMaps are often generated (or regenerated) automatically, there needs to be some way to preserve manual modifications. For instance, a certain event is run by the queue, and then the operators decide that the scale should be larger, so they run the event manually, using the *-latspan* flag to *grind*. If this information were not preserved, any subsequent automatic run of that event would revert to the original settings. Thus, we created the "shake_runs" database table, which keeps track of the parameters with which each program was last run. The program *shake*

(and ONLY the program *shake*) reads that database and uses the flags found there when running each of the subprograms.

This can result in confusing behavior. For instance, if you were to make some changes to the web pages for a particular event, and then run *transfer* with only the *-www* flag (because only web changes were made), the next run of shake on that event would run *transfer* with only the *-www* flag, which would not update the ftp site or push to clients, which might lead to confusion. Because *transfer* is often used this way, it has the *-forget* flag, which prevents it from updating the shake_runs table for that run. *shake* has the *-default_fl* flag which causes *shake* to ignore the "shake_runs" table and use the default flags for each sub-program as specified in the config file. This flag is handy in combination with the *-dryrun* flag for determining the default behavior.

Please keep these facts in mind when you are manually running events.

3.5.4.2 Using User-supplied Estimates

V3.5 Note: In previous versions of ShakeMap, the user could supply ground motion estimates by placing an XML file (of the 'stationlist' XML type) in the input directory. With version 3.5, we have changed that facility to use GMT grid files, rather than XML.

grind, unless directed otherwise, will attempt to make estimates of ground motion (based on an attenuation relation of your choosing). If one or more files called "<param>_estimates.grd" exist in the "ShakeMap/data/<event_id>/input" directory, these estimates will be used instead of those produced automatically by *grind*. The user-supplied estimate grid(s) should cover an area large enough to completely enclose the area being mapped. The grid spacing need not be the same as that used by *grind* because *grind* will resample it to the required grid spacing using GMT's *grdsample* program. For user grids coarser than those required by *grind*, bi-linear interpolation will be used. Be aware, however, that for user grids finer than that used by *grind*, aliasing may result from the down-sampling. (*grind's* grid size is set within grind.conf.)

In a similar vein, if "*_sd.grd" files are present in the input directory, *grind* will assume they contain the standard deviation of the estimates at each grid point. If no such files exist, *grind* will use the configured GMPE to compute uncertainties for the user-supplied estimates. This is almost certainly not what the user intends, so it is advisable to supply uncertainties corresponding to the user-supplied estimates. If your standard deviations are constant across the grid, it is a fairly easy matter to create the uncertainty grid for each parameter. For example, for the PGA grid (assuming the std. for the PGA estimates is XXX.YYY):

```
% grdmath pga_estimates.grd 0 MUL XXX.YYY ADD = pga_sd.grd
```

Finally, unless "*_estimates.grd" are in the input directory, *grind* will always recompute the estimates (i.e., the files in the directory "<shake_home>/data/<event_id>/richter" are regenerated with each run.) We do this because the input data could change (e.g. additional data arrives (affecting bias) or the event magnitude is revised), and the estimates should reflect this fact.

3.5.4.3 A Note About Estimates and Flagged Stations

grind will flag (i.e. cause not to be included in the maps) stations that appear to be outliers (relative to the GMPE estimates). *grind* will put these flagged stations into a file in the "ShakeMap/data/<event_id>/richter" directory called "flagged_stations.txt." Even in cases where the operator has supplied his or her own estimates (see previous section for details), *grind* will still use the GMPE for the purpose of flagging outliers. The user can override this behavior by placing a file "flagged_stations.txt" in the input directory. This file will be used in preference to the GMPE-based flagging done by *grind*.

In summary, if the operator computes estimates via some external program and places them in the input directory, *grind* will use them, but will flag outliers based on its own model. If the operator is using a sophisticated finite fault model, he or she will probably want to compute his or her own outliers and put them in a file "flagged_stations.txt" in the "input" directory, too.

3.5.4.4 A Note about Finite Faults

Events accept an optional finite fault file that will be used in generating estimates (for real events or scenarios), and are plotted on the map (see the parameters *ff_width* and *ff_color* in the configuration file *mapping.conf* for control of the way the fault is plotted). The fault filename must end in "_fault.txt" and should be placed in the event's *input* subdirectory.

The finite fault file is composed of a set of (latitude, longitude, depth) points defining the fault trace or fault surface. (Disconnected segments and polygons should be separated by a line in the file containing only the character '>' (greater than).) For example, two points can define a simple strike slip fault. A closed polygon (first and last points identical) can represent a fault surface. Each fault is defined by a set of 4-point planar segments (quadrilaterals) joined by common sides. The points should be arranged in clockwise- or counterclockwise order. Polygons must have an odd number of points (minimum five points) such that each quadrilateral consists of the points (n, n+1, (N-1)-n, N-n) for n = 1 to n = ((N-1)/2)-1, where N is the number of points in the polygon. See Figure 1 for an example.

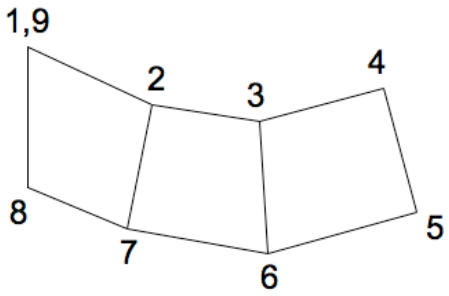

Figure 1: Example of point ordering for a polygonal fault surface. The fault surface consists of connected quadrilaterals. The first and last point of each segment should be the same (points 1 and 9 in the figure).

Each quadrilateral segment (i.e., 1278, 2367, 3456, in Figure 1) must have 4 corner points that are coplanar and non-collinear. Multiple fault segments must connect in linear fashion as shown in Figure 1; more degenerate configurations are not supported. One planar segment (4 points + the first point repeated) or two connected planar segments should be sufficient for most cases.

ShakeMap computes distance-to-fault to each line segment or polygon in the fault and uses the closest distance. ShakeMap modules may use either Joyner-Boore distance, or distance-to-rupture. A point inside a closed polygon is considered to be at zero distance by the Joyner-Boore function (as depth is ignored by this measure (though many GMPEs add a fitted pseudo-depth term to the JB distance)). It is recommended that all faults be defined as polygons, even if your normal GMPE uses JB distance. Polygonal faults allow the use of all GMPEs, as well as the directivity function within *grind*. If the rupture depth is unknown, it can be estimated using relations like those of Wells and Coppersmith (1994).

Note that to use the flag '-directivity' with *grind,* a polygonal fault as described above is required, regardless of the distance measure used by the chosen GMPE. For directivity, the first three points of a quadrilateral will define the plane, and the fourth point will be mapped onto the plane, as necessary. For example, in the middle segment of the fault in Figure 1, points 2, 3, and 6 will define a plane, and point 7 will be projected onto the plane in order to define the quadrilateral. Similarly, the hypocenter will be projected onto the nearest plane. In both cases, the projection is normal to the plane, not normal to the Earth's surface.

The fault file should be formatted as the input of the GMT *psxy* command using the '-:' flag (a '>' header, followed by space-delimited latitude-longitude-depth triplets.) Note that the coordinate ordering is the reverse of the conventional longitude-latitude paring.

Finally, the program *genex* transfers the fault files to the download directory so they are available on the web site. The info.xml file contains the name(s) of the fault file(s) used in making the maps.

Special note about faults in CY08: The CY08 GMPE module includes a hanging wall term for reverse slip earthquakes. This term requires special fault treatment. Specifically, the first (N-1)/2 points in the fault file (i.e., points 1, 2, 3, and 4 in Figure 1) are expected to be the shallow points of the fault.

3.5.4.5 Directivity

The flag *–directivity* to *grind* applies the directivity functions of Rowshandel (2006, 2010) to the amps predicted by the GMPE. The effect is limited to areas fairly close to the fault (especially for everything other than 3.0-second PSA). In cases where the fault does not reach the surface, the effect may not be noticeable at all. To use directivity, you must define a polygonal fault as described in section 3.5.4.4.

You can use -directivity with any GMPE, but the functions were developed for the four PEER-NGA GMPEs. While it is probably reasonable to use directivity with other active tectonic GMPEs, it is probably not applicable to other tectonic environments.

While it will function for any magnitude, Rowshandel does not endorse these functions for earthquakes less than M6.0.

Except for the DefaultIPE, directivity is not applied to intensity. If you are using the DefaultIPE, the directivity functions are applied to PGA and PGV before they are converted to intensity. We hope to revisit this topic as soon as there is research describing the application of directivity to intensity.

There is a new program called 'plotdirect'. If you run it ('plotdirect -event <evid>'), it will produce contour plots of the directivity functions in .../data/<evid>/richter. (You must run grind with the -directivity flag, first.) One figure is called dparam.jpg -- it is the directivity parameter itself (i.e. the thing you get from Equation 3 in Rowshandel (2010)). The other figures are called 'direct_<param>.jpg' -- they are the directivity factors applied to the estimates themselves (i.e., the thing you get (after exponentiation) with Equation 9 of Rowshandel (2010)). All the figures also have the fault and epicenter plotted.

3.5.4.6 Scenarios

ShakeMap supports the generation of earthquake scenarios. The user need only create the appropriate "*_dat.xml," "event.xml," and (optionally) "*_estimates.grd," "*_sd.grd," and finite fault files (see "A Note About Finite Faults," below) in an input directory. The scenarios are distinguished from real earthquakes in one of two ways: A) through the conscientious use of the *-scenario* flag in the many ShakeMap programs (not recommended), or B) by ending the event id with "_se" (e.g., event "myscenario_se" in directory "<shake_home>/data/myscenario_se/") (highly recommended).

Scenario earthquakes are distinguished from real ones by a staggering number of appearances of the word "Scenario" on the maps and web pages, including a big one emblazoned across the face of the maps themselves. We do this to prevent the misunderstandings in the press and public that would likely occur if we were any less zealous. Scenarios have their own place on the archive page, distinct from the real earthquakes, and they will not appear in the lists of real events or on the homepage.

Most of the programs are scenario-savvy. 'shakemail', for instance will not email scenarios unless you force it to. *transfer* will transfer to web sites (*-www*) and ftp sites (*-ftp*) but will not push (-push) unless you force it to. Run the various programs with *-help* to learn the scenario-related options and behavior.

To create a new scenario, the most straightforward way is:

1) Create a new event subdirectory (say, "<shake_home>/data/1857_se") and a new "input/" directory under that ("<shake_home>/data/1857_se/input").

2) Copy the “event.xml” file from an existing event over to the new input directory, and modify the parameters. (Don't forget to change the ‘id’ field to be the same as your directory name (e.g., 1857_se).)
3) Add a finite fault file, if desired (see “A Note About Finite Faults,” below).
4) Run 'shake -event *event_id* -dryrun' to get a listing of the programs to run and to check the flags are correct. Make sure *tag* is run with ‘*-scenario* <scenario_description>.’ Then run the programs in the order specified by *shake*.

The user may, additionally, create estimates using a non-ShakeMap program and include them in the input. These estimate files should be GMT .grd files that cover at least the area being mapped, and should be named like ‘<param>_estimates.grd’ where ‘<param>’ is ‘pga,’ ‘pgv,’ ‘psa03,’ etc. There should also be .grd files containing the point-by-point uncertainty (given as standard deviation), which should be named like ‘<param>_sd.grd.’ (If uncertainty is constant over the grid, it is easy to make a .grd file using *grdmath*.)

3.5.4.7 Getting Help

You may contact us through the shake-dev mailing list:

https://geohazards.usgs.gov/mailman/listinfo/shake-dev

This mailing list goes out to many ShakeMap operators and developers, but is a fairly low-traffic list. Please sign up. We use this list not only to respond to questions, problems, and bug reports, but also to announce software updates, bug fixes, etc.

Visit the site to add yourself to the list.

3.6 Common Problems

We welcome contributions to this section. Please let us know about problems you have had with ShakeMap, and your workarounds (if any).

3.6.1 Shake flags database causes confusion

See section 3.5.4.1, *A Note about Program Flags*, above.

3.6.2 Files in incorrect format:

When configuring region-specific files, make sure to create files following the formats in the example (i.e., southern California) files. If the code is written to read a space-delimited file, commas will cause problems and vice versa. For the *GMT* files make sure you have the latitude and longitude in the correct columns. Remember that files that define faults are in lat, lon, depth order rather than the more conventional lon, lat, depth order.

3.7 XML Formats in ShakeMap

3.7.1 About XML

XML is a system for tagging text to indicate the structure of information in the text. XML started as a generalization of HTML (or a simplification of SGML, depending on your perspective), and XML markup is similar in appearance to HTML tags. However, in XML the tags are defined on a per-application basis. With this flexibility, XML can be used as a means of structuring data in a cross-platform, human-readable form, in addition to its use handling textual documents.

A complete specification of XML is available at http://www.w3.org/TR/REC-xml (http://www.w3.org/TR has a number of interesting documents) and an annotated version is at http://www.xml.com/axml/axml.html.

However, preparing XML files for ShakeMap does not require knowing the XML specification. For working with ShakeMap, it will probably be enough to get a short summary, in particular contrasting XML with the more familiar HTML.

An XML file starts with a declaration line:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

Version refers to the XML standard to which the file is written. Currently, "1.0" and "1.1" are the only options, and we always use 1.0. Encoding refers to the character set in which the file is written. Standalone indicates whether the XML file is free of references to outside definitions in other XML files.

Following the declaration is an optional Document Type Definition (DTD) block, which may refer to a definition in another file:

```
<!DOCTYPE earthquake SYSTEM "earthquake.dtd">
```

or present the definition in place:

```
<!DOCTYPE earthquake [
  ... DTD description ...
]>
```

The DTD system has been almost entirely supplanted by the use of XML Schemas, but we have not yet adopted that approach for our input files, and it remains a low priority.

Then the XML itself starts. XML tags look a lot like HTML tags with a tag label and possibly attributes:

```
<tag att1="val1" att2="val2">
```

In contrast to HTML, XML tags and attributes are case sensitive, so <station> and <STATION> are different. Also, attribute values must always be wrapped in quotes, so <station code="PAS"> rather than <station code=PAS>.

In HTML, some tags are simple tags that don't contain other tags or blocks of text. For example:

```
<img src="..." border="0">
```

The equivalent in XML is called an empty tag, and only differs from HTML by closing with "/>" rather than ">":

```
<pga value="0.25"/>
```

Non-empty tags contain blocks of other tags and/or character data, such as:

```
<station code="PAS">
 <comp name="HLN">
  <acc value="0.25"/>
 </comp>
</station>
```

Example codes that demonstrate writing XML are available in the ShakeMap distribution package (in <shake_home>/src/xml), and since XML files are text files this consists mainly of simple printing of formatted output. For input, XML parsers are freely downloadable for the Perl, C and Java programming languages. ShakeMap is predominantly written in Perl, so we use a well-regarded parser library in that language. As with XML output, example codes in the ShakeMap distribution show how input parsing is handled. A list of XML parser libraries in various programming languages is available at http://www.w3.org/XML/#software.

Every XML file has a set of tags used in a pattern particular to that type of file. This pattern is set by the developer and can be indicated in a Document Type Definition (or schema). The DTD defines the tags that it expects, the order it expects them in, and how tags can nest within one another. It also indicates what tags are optional, what tags can appear multiple times in succession, what attributes are associated with each tag, and (optionally) a range of values accepted for an attribute.

Some parsers have an option to 'validate' an XML file according to its DTD, but the parser used by ShakeMap does not do so. However, we have found it useful to define DTD's for the various XML file types that ShakeMap works with, if only for documentation purposes during development. These ShakeMap DTD's will be discussed below for each file type. Again, this approach should be revised to use formal schemas, but that approach will not change the basic structure of the files.

3.7.2 ShakeMap XML Files

Before ShakeMap is run for a particular event (identified by an event id), the following set up is needed:

- a directory in <shake_home>/data/<event_id>/input

- an 'event.xml' file in this directory
- zero or more files with filenames ending in '_dat.xml' in this directory

The contents of the 'event.xml' file are earthquake parameters in the 'earthquake.dtd' format. This format is a single, empty tag with a number of attributes of the earthquake. The attributes are given in the following table.

Event information	
id	the event id
created	file creation time (Unix epoch -- seconds since Jan 1, 1970)
Hypocenter information	
lat	latitude (in decimal degrees, negative in southern hemisphere)
lon	longitude (in decimal degrees, negative in western hemisphere)
depth	in km, positive down
locstring	a free-form descriptive string of location relative to landmarks
mag	magnitude
type	a string specifying the rupture type; the accepted types are RS, SS, NM, and ALL, for reverse slip, strike slip, normal, and unspecified ruptures, respectively.
Origin time parameters	
year	4 digit format
month	1-12
day	1-31
hour	0-23
minute	0-59
second	0-59
timezone	abbreviation (i.e., GMT, PST, PDT)
Amplitudes at the epicenter	
pga	peak acceleration (units of %g)
pgv	peak velocity (units of cm/s)
sp03	Spectral acceleration at 0.3 sec period (units of %g)
sp10	Spectral acceleration at 1.0 sec period (units of %g)
sp30	Spectral acceleration at 3.0 sec period (units of %g)

The amplitude attributes in 'earthquake.dtd' are estimates produced by ShakeMap during processing. These attributes should be left out of the 'event.xml' file input to ShakeMap, and will be ignored if present.

An example 'event.xml' file looks like:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE earthquake [
  ... DTD description ...
]>
<earthquake id="14000376" lat="34.2722" lon="-118.7530"
mag="3.6" year="2003" month="10" day="29" hour="23" minute="44"
```

```
second="48" timezone="GMT" depth="13.81" locstring="2.6 mi W of
Simi Valley, CA" created="1069292035" />
```

Files in the input directory named like '*_dat.xml' are station parameters in the 'stationlist.dtd' format. This format has a root 'stationlist' element containing one or more 'station' elements. The 'stationlist' can have a 'created' attribute with the file creation date in Unix epoch time (seconds since Jan 1, 1970). Each station has a set of attributes:

code	the station code
name	station name and/or description
insttype	description of instrument type
lat	station latitude (in decimal degrees)
lon	station longitude (with negative sign in western hemisphere)
source	agency that maintains the station (i.e., SCSN, CDMG, NSMP,...)
netid	the network ID string; for MMI observations this must be one of 'MMI,' 'CIIM,' 'DYFI,' or 'INTENSITY.'
commtype	digital or analog communications (DIG or ANA)
loc	free form text describing the location of the station (optional)
intensity	the intensity value of the observation (decimal)

Each station element contains one or more 'comp' elements, one for each component of ground motion. Comp elements have the following attributes:

name	a channel name/code in SEED convention
originalname	the original channel name if it was not SEED (optional)

The name attribute may be a SEED-convention component name, or it may be a string describing the source of the observation. If the name is not known (for example if the source of amplitudes only gives a single summary value for the station), then use the most generic code for a horizontal component, HL1. Use a horizontal code rather than HLZ because ShakeMap uses only horizontal components in processing. In the ShakeMap output stationlist.xml file, 'name' may also be "DERIVED" to indicate that the ground motions were derived from a primary intensity observation.

If the amplitude is from an agency that does not use SEED component codes, you will have to map their codes to a comparable SEED code for the name attribute. If you would like the original code carried through the processing and used in the HTML, XML, and text stationlists, then put the original code in the originalname attribute.

Each 'comp' element must contain one 'acc' element, and one 'vel' element, and may contain 'psa03,' 'psa10,' and 'psa30' elements (one of each). These refer to peak acceleration, velocity, and 5%-damped pseduo-spectral acceleration (at 0.3, 1.0, and 3.0 sec period) values for the named channel at the named station. The acc, vel, psa03, psa10, and psa30 elements are empty but have the following attributes:

value	the amplitude value
flag	flag indicating problematic data (optional)

The value attributes are expected to have units of:

acc	%g (i.e., percent of the earth's nominal gravitational acceleration)
vel	cm/s (centimeters per second)
psa	%g

The flag attribute indicates problematic data. Any value other than "0" (zero) or "" (i.e., an empty string) will cause ShakeMap to reject the amplitude (and, in fact, all the amplitudes of that type for that station). ShakeMap also does automatic flagging of outliers (see the program *grind* and the section "Running ShakeMap," above, for more information on automatic flagging). Though any non-zero flag will kill an amplitude, the following flags are currently defined:

T	Automatically flagged by ShakeMap as an outlier
M	Manually flagged (in grind.conf) by the ShakeMap operator
G	Amplitude clipped or below the instrument noise threshold
I	Incomplete (a data gap existed in the time window used to calculate the amplitude)

An example of a '*_dat.xml' file is:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE stationlist [
  ... DTD description ...
]>
<stationlist created="1070030689">
<station code="ADO" name="Adelanto Receiving Station"
insttype="TriNet" lat="34.55046" lon="-117.43391" source="SCSN
and TriNet" commtype="DIG" netid="CI" loc="Adelanto, on Hwy 395
">
<comp name="HHE">
<acc value="0.0083" flag="0" />
<vel value="0.0030" flag="0" />
<psa03 value="0.0146" flag="0" />
<psa10 value="0.0049" flag="0" />
<psa30 value="0.0003" flag="0" />
</comp>
<comp name="HHN">
<acc value="0.0088" flag="0" />
<vel value="0.0028" flag="0" />
<psa03 value="0.0111" flag="0" />
<psa10 value="0.0040" flag="0" />
<psa30 value="0.0004" flag="0" />
</comp>
<comp name="HHZ">
<acc value="0.0087" flag="0" />
```

```
<vel value="0.0016" flag="0" />
<psa03 value="0.0080" flag="0" />
<psa10 value="0.0013" flag="0" />
<psa30 value="0.0002" flag="0" />
</comp>
</station>

… additional station tags (omitted)…

<station code="WSS" name="West Side Station" insttype="TriNet"
lat="34.1717" lon="-118.64971" source="SCSN and TriNet"
commtype="DIG" netid=”CI” loc="Hidden Hills, Valley Circle Dr.">
<comp name="HHE">
<acc value="0.0225" flag="0" />
<vel value="0.0031" flag="0" />
<psa03 value="0.0182" flag="0" />
<psa10 value="0.0016" flag="0" />
<psa30 value="0.0002" flag="0" />
</comp>
<comp name="HHN">
<acc value="0.0209" flag="0" />
<vel value="0.0029" flag="0" />
<psa03 value="0.0234" flag="0" />
<psa10 value="0.0019" flag="0" />
<psa30 value="0.0001" flag="0" />
</comp>
<comp name="HHZ">
<acc value="0.0187" flag="0" />
<vel value="0.0020" flag="0" />
<psa03 value="0.0073" flag="0" />
<psa10 value="0.0005" flag="0" />
<psa30 value="0.0000" flag="0" />
</comp>
</station>
</stationlist>
```

Intensity data uses the same format of input XML as other ground motion data, but uses two new attributes to the *station* tag: the *intensity* attribute should be set to the decimal intensity for the “station;” the *netid* attribute should be set to “MMI,” “CIIM,” “DYFI,” or “INTENSITY” (all four are currently equivalent). If *netid* is set to one of these values, any amplitude data (i.e., data enclosed in a *comp* tag) will be ignored and *grind* will use the mmi2pgm function to derive the ground motions. Likewise, if *netid* is not one of these values, the *intensity* attribute will be ignored and *grind* will compute intensity using the pgm2mmi function.

Below is an example of a station tag that contains intensity information:

```
<station code="91042" name="ZIP Code 91042 (Intensity VII, 38
responses)" insttype="USGS (Did You Feel It?)" lat="34.282604"
```

```
lon="-118.237943" source="USGS (Did You Feel It?)" netid="CIIM"
commtype="USGS (Did You Feel It?)" intensity="7.4">
```

The earthquake and stationlist XML files are combined in the output file provided to the public. This file is made available as XML and is also the basis for a raw, non-XML text stationlist and the HTML web stationlist linked to the ShakeMap click-maps. Since the output XML file combines the event and station files, it also merges the earthquke and stationlist DTD's into a 'shakemap_data' DTD that is included in the file.

3.7.3 Retrieving Data from a Database

As run by SCSN/CISN, ShakeMap is triggered by a realtime processing system and accesses a database for event parameters and amplitude values from Caltech/USGS-Pasadena stations. Additional amplitude values are received from CGS and NSMP stations and are incorporated in the processing as they arrive. See the section "External Data XML Files," below.

To access the database, ShakeMap launches *retrieve* which launches any number of specific helper codes (defined in a configuration file) to build the "event.xml" and "*_dat.xml files." These codes can be used as examples of database access to build input files. If your network is running a DBMS with the schemas used by the southern or northern California Earthquake Data Centers, then you may be able to use the ShakeMap codes directly. If you are using a DBMS with a different schema, it will be necessary to modify at least the SQL calls embedded within the example programs and possibly the logic of the programs themselves if the schema differences are large.

3.7.4 External Data XML Files

External (i.e., not directly from database) amplitudes can be included in ShakeMap once they are associated with an earthquake. Just make a stationlist.dtd-format XML file with a unique name ending in _dat.xml and drop it in the correct <event id>/input directory.

In order to associate amps, data need to be received in a structured way. One possibility is defining an XML format. We have taken this approach with some external data, and the XML format is described here as an example. In this case, the data arrive as amplitudes unassociated with an earthquake. The main difference between the stationlist XML files fed directly to ShakeMap and the external amplitude XML files is the addition of timing information (the basis for the association). The root element of the external amplitude file is an 'amplitudes' element. 'amplitudes' has an 'agency' attribute so we can know who the amplitude report is from. The amplitudes element contains one or more 'record''elements. The record element can have an agency-defined 'id' attribute assigned to it.

The record element contains 'timing' and 'station' elements. The timing element has no attributes but contains 'reference' and 'trigger' elements. The reference element has two attributes, 'zone' for a timezone code (i.e., GMT, PST, or PDT) and 'quality' for an agency-defined indicator of the timing quality. 'reference' contains a set of elements:

year	4-digit year
month	1-12

day	1-31
hour	0-23
minute	0-59
second	0-59 (60 for leap second)
msec	0-999

each of which has an integer ‘value’ attribute as defined above. ‘trigger’ is an empty tag with a ‘value’ attribute assigned the time in seconds of the amplitude trigger relative to the reference time. CGS has a common trigger time for all components in a record, so the trigger tag is not stored at the component level.

The ‘station’ element has four attributes:

code	station code
name	station name or description
lat	station latitude (in decimal degrees, negative in the southern hemisphere)
lon	station longitude (in decimal degrees, negative in the western hemisphere)

and contains one or more ‘component’ elements. Each component has a ‘name’ attribute that defines the component (in an agency-defined way), and contains ‘acc’, ‘vel’, and ‘sa’ elements. Each of these elements has ‘value’ and ‘units’ attributes, where value is the amplitude value itself, and units is a string expressing the units (i.e., g, or %g, or cm/s/s). ‘sa’ has an additional attribute, ‘period’, that defines the period, in seconds, of the spectral value. For each component, there is one acc, one, vel, and zero or more sa elements.

An example of an external amplitude XML file is:

```
<?xml version="1.0" encoding="US-ASCII" standalone="yes"?>
<amplitudes agency="CDMG">
 <record>
  <timing>
   <reference zone="GMT" quality="0.5">
    <year value="2000"/>
    <month value="02"/>
    <day value="21"/>
    <hour value=" 13"/>
    <minute value="49"/>
    <second value="0"/>
    <msec value="0"/>
   </reference>
   <trigger value="0"/>
  </timing>
  <station code="23920" lat="34.004" lon="-117.058"
name="Yucaipa Valley">
   <component name="Up">
    <acc value=" .013" units="g"/>
    <vel value="  .32" units="cm/s"/>
```

```
    <sa period="0.3" value="0.01160" units="g"/>
    <sa period="1.0" value="0.00204" units="g"/>
    <sa period="3.0" value="0.00070" units="g"/>
   </component>
   <component name="90">
    <acc value=" .026" units="g"/>
    <vel value="  .63" units="cm/s"/>
    <sa period="0.3" value="0.02261" units="g"/>
    <sa period="1.0" value="0.00418" units="g"/>
    <sa period="3.0" value="0.00135" units="g"/>
   </component>
   <component name="360">
    <acc value=" .028" units="g"/>
    <vel value="  .58" units="cm/s"/>
    <sa period="0.3" value="0.02152" units="g"/>
    <sa period="1.0" value="0.00375" units="g"/>
    <sa period="3.0" value="0.00205" units="g"/>
   </component>
  </station>
 </record>
</amplitudes>
```

Example codes that parse this XML format and convert it to the ShakeMap input format are part of the 'dirwatch' modules found in <shake_home>/src/watcherlib and <shake_home>/src/cdmglib. In particular, see the module watcherlib/AssocAmp.pm.

3.8 Development Model

We are trying to handle ShakeMap development as an open-source project. That means that various developers can contribute to the project the code that they feel improves the overall product. This also means that those contributions must not be site-specific unless they are easily bypassed by other users (through configuration options, for example). Changes, improvements, additions, etc. will be sent back to the USGS, to be included in the distribution product (or to be sent back to the originator for revision). If all goes smoothly, your site may make extensive changes to the core product, send them back to the distribution source, have them integrated into the code base, and then receive them back with the next release of the source. This should lead to (relatively) painless upgrades, not to mention a better product for everyone.

None of this prevents a site from taking the code and running totally wild with it. It simply means that their work will not be included in future releases and upgrades to the core ShakeMap product.

3.9 Tables

Table 3.2A. Files and directories in the top-level of the ShakeMap distribution

Makefile	The highest-level makefile in the distribution.
config	Initially contains only a README file explaining how the configuration files are formatted; once a 'make' is done, the directory will be populated

	with various config files for ShakeMap; these files will be edited by the operator to meet region-specific needs.
doc	Most of the ShakeMap documentation.
install	The first stop when doing an install of a ShakeMap distribution; see "Installing the Software," above.
lib	Contains mapping and data files, site correction data, web pages, web page templates, and supporting graphics; see "Configuring ShakeMap," above, and sections below for more information.
src	The directory where the ShakeMap source code and Perl modules live.
util	Directory containing a few (occasionally) handy programs. It may be safely ignored.

Table 3.2B. Subdirectories of Interest

src/cdmglib	Contains perl modules that are used by dirwatch, the directory watching program; these modules are used in the conversion of CGS (nee CDMG) XML or CGS two-line parametric files into ShakeMap XML.
src/cfgsrc	The source for the default configuration files; the installation process copies these into <SHAKE_HOME>/config, then merges them with any existing config files. The user then customizes them for a specific environment.
src/config	Contains the modules ShakeConfig.pm and WatcherConfig.pm which hold global variables used by most of the ShakeMap programs; these modules have site-specific customizations made to them and are installed in <SHAKE_HOME>/lib by the program 'config' (also found in this directory). No user intervention is required.
src/contour	Contains the source to the 'contour' program. 'contour' converts GMT .grd files into GIS shapefiles (polygons of "constant" parametric value). Also in this (now-misnamed) directory are the sources for the 'sm_nearneighbor' and 'dist_rrupt' programs.
src/genexlib	Directory with modules specific to the program *genex*.
src/lib	Directory containing modules used by several of the ShakeMap programs; most of these modules have (non-POD) documentation within them.
src/lib/GMPE	Sub-directory containing the source code for various Ground Motion Prediction Equations (typically named by author(s) and year: BJF97, Youngs97, etc.)
src/lib/IPE	Sub-directory containing the Intensity Prediction Equations. Named for author(s) and year.
src/lib/GMICE	Sub-directory containing the source of various PGM⇔Intensity (MMI) conversion functions (Ground Motion/Intensity Conversion Equations). Like the modules in the GMPE directory (above), these are typically named for author and year (e.g., Wald99, for Wald et al., 1999).
src/misc	Contains a couple of helpful programs: a perl version of 'echo' and the infamous configconfig, the programs required by the MySQL conversion of pre-V3.0 ShakeMap databases (mktables, eq2mysql, and shake2mysql), and some other ad hoc programs.

src/queue	Contains the event queueing and automatic ShakeMap initiating program used by the southern California network; individual sites will probably want some custom variation of this program; see src/cfgsrc/queue.conf for customization options; directory also contains the alarming and cancellation scripts.
src/shake	Contains the core of the ShakeMap software; most of these programs have a configuration file (in src/cfgsrc) that explains how each may be customized; see "Shake Programs" below for a discussion of the individual programs.
src/util	This directory holds programs to convert the ascii lat-lon-velocity file to binary and back to ascii; see the section on configuring ShakeMap for more information. Also in this directory are programs to create the instrumental intensity scales for the II map and the TV map.
src/watcher	Contains the dirwatch program; the dirwatch program and its associated modules provide the service of watching a directory for the arrival of a file, and then dispatching that file to its proper destination; see the description of the modules in src/watcherlib, below; see the README in src/watcher for a discussion of the program's capabilities.
src/watchercfg	Contains configuration files for the watcher modules.
src/watcherlib	Currently contains two modules (three, actually, but Base.pm is general purpose): AmpDir.pm: Takes the 2-line CDMG text parameter files as input, converts them to unassociated XML, and deposits the new file in a user-specified directory. AssocAmp.pm: Takes the unassociated XML file, tries to associate it with a TriNet event, converts the XML to ShakeMap XML, deposits this file in the input directory for ShakeMap, and, after waiting a user-specified time, alarms the queue that the event has been updated.
src/xml	Contains various programs for converting data files and database results into ShakeMap XML files. See Table 3.5, Miscellaneous Programs for additional documentation. This directory also contains the DTD files describing the earthquake and stationlist XML formats.
lib/genex	A collection of HTML and templates that, through the magic of the *genex* program, become the ShakeMap products and web site.
lib/mapping	Contains data files used by the *mapping* program: highways, faults, cities, topography, colormap, etc. Much of the customization of ShakeMap happens in this directory. See config/mapping.conf for more details.
lib/ps	Contains the PostScript of the Instrumental Intensity scales for the intensity map and the TV map.
lib/sitecorr	Contains the station velocity file, the site amplification table, and the text and binary versions of the geology file; review these files and create versions specific to your region.

lib/transfer	Contains dummy files used by *transfer* when pushing data files to remote sites.
lib/xml	Holds the DTD's for the ShakeMap XML; the DTD's are prepended to the earthquake and stationlist data files.

Table 3.2C Directories Created After Installation

bin	All of the executable programs will end up here after a 'make all.'
data	Repository of all event data and processed files. Discussed below ("Data Directory Structure") and elsewhere.
pw	(Actually, the name and location of this directory is user-defined); this is where database passwords are kept; should be read protected for security; see the db.conf configuration file and the Password.pm module (in src/lib) for examples of use.
perl	Directory where the various perl modules end up after a 'make all'; it is also permissible to install other perl modules used by ShakeMap (e.g. DBI) here.
include	Holds the macros used by makefiles and the config program.
logs	Directory in which the queue puts its logging and error files.
watcher	Host directory where the various directory watcher modules (dirwatch program) look for config files and dump bits of information. May also hold the logs. This directory can be ignored if you do not use the dirwatch program.

Table 3.3A. Region-Specific Files in 'grind.conf'

Parameter: ampfactor_file **File:** lib/sitecorr/[name].dat	File containing site amplification factors as a function of Vs30 and frequency of input ground motion. See the southern California file site_corr_cdmg.dat for documentation and format.
Parameter: none **File:** lib/sitecorr/[region].txt	Geology file. nx by ny (where dx=dy) rectangular grid of the Vs30 values for the ShakeMap region. This file must be comma delimited: lon, lat, Vs30 (where west longitude is negative). This file is not strictly necessary but many regions have created it as an intermediate step to creating the (now obsolete) .bin file. See the next two entries, below, for more explanation.
Parameter: none **File:** lib/sitecorr/[region].bin	Binary form of the above file. Now supplanted by a .grd file (see the next entry, below). If you have this file but not its text version (see previous entry), use qtmbin2latlon to generate the text from this file.
Parameter: qtm_file **File:** lib/sitecorr/ [region].grd	A GMT .grd file of VS30 over the entire region of interest (plus any buffering you wish to add). You may generate this file in any number of ways, but it can be generated from the .txt file, above, with a command like: `xyz2grd -R<WESN of region> -I60c usgs_vs30_60s.txt -Gusgs_vs30_60s.grd` See the GMT documentation for more information about .grd files.
Parameter: mmifactor_file **File:** lib/sitecorr/[region]	The basename of GMT CPT files containing forward and reverse site corrections for MMI. See grind.conf for information on creating these files.

Parameter: stavel_file **File:** lib/sitecorr/ dig_[region].txt	File containing station information: lat, lon, sta name, Vs30; stations not found in this file will be assigned the Vs30 of the nearest grid point from the geology file, above. This may be the same file that is given as fwstatlist, below.
Parameter: fwstatlist **File:** lib/grind/ [region]statlist.txt	List of stations used by the *-scenario* option (to *grind*) to create dig_dat.xml by forward modeling.

Table 3.3B. Region-Specific Files in 'mapping.conf'

Parameter: topo_cmap **File:** lib/mapping/ [region].cpt	GMT colormap file for plotting regional topography; the default file 'tan.cpt' may work for many regions, but you can use any colormap you find appropriate.
Parameter: ii_cmap **File:** lib/mapping/ [region].cpt	This GMT colormap converts intensity (MMI) into a color. We suggest using the default "Ii.cpt" unless we all reach consensus that it needs to be changed. If it is changed, note that the scale at the bottom of the intensity maps will need to be changed, as well.
Parameter: ii_tvmap_cmap **File:** lib/mapping/ [region].cpt	Like ii_cmap above, this colormap is used to convert intensity into a color, but for the maps intended for broadcast TV (which have limits on the colors they can broadcast). This has turned out to be of limited use, and we currently us a colormap that is identical to the one we use for ii_cmap.
Parameter: sd_cmap **File:** lib/mapping/ [region].cpt	Another GMT colormap. This one converts the uncertainty (expressed as the ratio of site standard deviation to the GMPE standard deviation) into a color.
Parameter: map_roads **File:** lib/mapping/ [region]_roads.xy	Text file containing GMT *psxy*-style coordinates of road segments: lon, lat pairs grouped by segment , segments separated by a '>'.
Parameter: map_faults **File:** lib/mapping/ [region]_faults.xy	Text file containing GMT *psxy*-style coordinates of fault segments: lon, lat pairs grouped by segment, segments separated by a '>'.
Parameter: map_topo and map_topo_hires **File:** lib/mapping/ [region]_topo.grd	GMT grid file for the regional topography. Optionally, you can have both high and low resolution forms. Note, these files are not the topography files used by 'topo2grd' (defined in grind.conf).
Parameter: topo_intensity and topo_intensity_hires **File:** lib/mapping/ [region]_topo_intens.grd	GMT grid file of intensity* for the regional topography grid given above. If this file (or the high resolution version) does not exist, the *mapping* program will generate it. *Note this is not ground motion intensity, but plotting intensity as produced by *grdgradient* from specifying the position of a light source.

Parameter: map_cities, map_bigcities, and map_verybigcities **File:** lib/mapping/ [region]_cities.txt, [region]_bigcities.txt , and [region]_verybigcities.txt	Files containing city names and locations. The use of these files is now deprecated; use the '_label' versions instead. See 'mapping.conf' for more details. A program 'fix_cities' is provided to convert old city files to new ones; read the program source for documentation.
Parameter: none **File:** lib/mapping/ tvguide.txt	Optional, edit this file to reflect local contact information.

Table 3.3C. ShakeMap Programs

shake	Config: **shake.conf** The main program; actually a wrapper program that calls the other programs. The configuration file controls what programs shake calls and how they are called. After shake calls the first program in the list (usually *retrieve*, see below), it expects a file, "event.xml," in the event's input directory.
retrieve	Config: **retrieve.conf** Usually the first program called by *shake*; retrieve is itself a wrapper code that calls other programs that are meant to retrieve data and put it in the event's input directory; the configuration file explains the customization options.
pending	Sends a new home page to the web site to indicate that an event is being processed; pending calls *genex* with the *-pending* flag, and *transfer*.
grind	Config: **grind.conf** *grind* reads the data files it finds in the event's input directory and generates grid files with interpolated ground motions, as well as the text and XML parameter files, and the station and uncertainty files. *grind* puts its output in a directory called '<shake_home>/data/<event_id>/output.'
tag	ShakeMap keeps an earthquake database that it uses to generate the home page and the archive pages; *tag* specifies to the database that an event is a) ordinary, b) a mainshock, c) an historic, named, event, d) invisible (on the web pages), or e) part of an aftershock cluster associated with a mainshock.
mapping	Config: **mapping.conf**, **colors.conf** Reads the grids generated by *grind* and makes PostScript maps of ground motion and shaking intensity, contour files, and generates information needed to make imagemaps; all of this output is placed in the event's 'mapping' directory.
~~asciimap~~	Obsolete. Removed in V3.5.
genex	Config: **genex.conf**, **web.conf** Uses the output of *grind* and *mapping* to create JPEGs, build web pages, and generate GIS and other files for export via the web or FTP.

shakemail	Config: **shakemail.conf** Generates a number of different email notifications of ShakeMap availability (long format, short format, attached JPEG, and list of flagged stations). See shakemail.conf for details.
addon	Config: **addon.conf** Creates and copies a QDDS-formatted file to a local QDDS directory; QDDS should then add a link to the just-created ShakeMap from the Simpson maps. Will also send a delete message for cancelled events.
print	Config: **print.conf** Sends plots to printers.
transfer	Config: **transfer.conf** Transfers the output created by *genex* to the web and ftp sites, also 'pushes' ShakeMap data to remote sites via FTP. *transfer* has been pirated for other uses as well: it is used to transfer the permanent parts of the web pages to the web site(s), and it transfers a temporary 'pending' page to the web while an event is being processed. Unless instructed otherwise, *transfer* cements a version of an event's maps and causes subsequent maps to have an incremented version number.
setversion	Manipulates the version information for an event and preserves versions as requested. Run *setversion –help* for more information. Also, see the section on version control in this manual.
plotdirect	Config: **genex.conf** If *grind* was run with the flag *–directivity*, this program will create plots in the richter sub-directory of the directivity parameter and the directivity factors for each of the ground motion types (called *dparam.jpg* and *direct_<param>.jpg*, respectively.
plotregr	Config: **mapping.conf, genex.conf** Creates plots of the PGM and intensity attenuation (from the GMPE or IPE), both biased and unbiased, with the amplitudes from the stations. Also plots lines indicating the standard deviation bounds used for flagging. The plots are found in the event's data directory in the sub-directory *regression*. There are PostScript, PDF, and PNG versions of each plot. Note that because of the complex way distance and standard deviation are treated by ShakeMap and some GMPEs, some flagged stations may appear within the std. bounds, and unflagged stations may appear outside them.
cancel	Config: **shake.conf** *cancel* undoes the effect of *shake*: it removes the event (except what is found in the input directory) from the data directory and removes the event from the earthquake database; it removes the web pages for the event and updates the home and archive pages to reflect the removal of the event; it deletes all associated data from the ftp site(s) and it pushes a file, '<event_id>.cancel,' to push clients
unlock	If an event is locked, preventing the execution of ShakeMap programs, this program will break the lock.

dist_rrupt	A helper program for *grind*. Given a 3-D fault description and a list of points, computes the closest distance from each point to the fault rupture plane. Implemented in C for efficiency.
sm_nearneighbor	Another helper program for *grind*. Based on GMT's *nearneighbor* program, this program computes a weighted average of nearby data points for each point in a grid. Implemented in C for efficiency.
geterror	Another helper program for *grind*. Computes the letter grade and uncertainty score for the grids produced by *grind*, following the method of Wald, et al. (2008).
topo2grd	A helper program for *grind*. Converts digital elevation models into Vs30 grids suitable for use by *grind* when running with the -qtm flag. This program is only likely to be of use within the Global ShakeMap program -- most operators will want to produce static Vs30 grids for their geographic area.
view_rcg	If *grind* is run with the -rcg flag, it will produce Relative Contribution Grids within the event's *richter* directory. *view_rcg* converts those grids into PostScript maps and leaves them in the *richter* directory.
plot_vs30	Plots the Vs30 grid used by *grind*. This program may be configured in *shake.conf* to run automatically. It is generally intended for use where the Vs30 grid is not static, but is constructed on the fly (see *topo2grd*, above).
zone_config	Allows the selection of region-specific configuration files. *zone_config* reads box definitions just like *queue* and looks for "[config*].boxname" files in the *zone* directory (within <*shake_home*>/*config*) then copies them into event-specific config directory. The operator may define boxes in the *zone_config.conf* file then place specific config files into the zone directories. The default *zone_config* config file contains most region/craton boundaries. *Where "config" one or more program configuration files (e.g., *grind.conf, mapping.conf, etc*.
fix_cities	A now-obsolete file that was used to convert old-style city files to new-style city files in a previous release. Remains in the distribution mostly as a reference.
getnsmp	A program used to associate unassociated NSMP data with an event, then call nsmp2xml to generate an NSMP data file in the event's *input* directory. Not generally useful, but remains as a reference.

iiscale tvscale	These programs use the formulae of the various GMICE functions to generate a bit of PostScript that makes the intensity scale at the bottom of the intensity plots. These programs are run when ShakeMap is installed and put the PostScript in files in *<shake_home>/lib/ps*. The conversion modules in *GMICE* contain a function that returns the name of the correct PostScript file to use with that module, which *grind* extracts in puts in an info file for *mapping*. *iiscale* should be run like: *% iiscale <GMICE>*, where <GMICE> is replace with the name of one of the modules ("Wald99" or "AK07", for example). The output file will be named like: *scale_<lc(GMICE)>.ps*, where *<lc(GMICE)>* is the lower case name of the GMICE used (e.g., Wald99 generates file *scale_wald99.ps*, and AK07 generates *scale_ak07.ps*). *tvscale* produces a simpler, more generic scalebar that does not depend on the specifics of the GMICE used.

Table 3.4 Subdirectories Found Within an Event Data Directory

input	Directory in which the input XML file or files are placed. The operator may also include a finite fault file, estimates/uncertainty files, and flagged station files into this directory.
output	Directory in which *grind* places its output.
richter	Another directory that contains output from *grind*. The estimate grid and flagged stations files are written here if *grind* is called upon to generate them. *Grind* also writes files to this directory that *plotregr* uses.
mapping	This directory will contain PostScript files generated by *mapping*, and JPEG files converted from the PostScript by *genex*; also contains contour files, the ASCII map, and other miscellaneous mapping products.
genex	This directory contains products ready for transfer to the web and ftp sites. It contains two sub-directories 'web' and 'ftp.' Each of these contains files set up in a directory structure that lends itself to being copied wholesale to its destination.
raw	This directory is not created by the ShakeMap software, but may be created by the user; it is a holding area for input data that is not in the proper XML format. Some programs (*dig2xml*, *ana2xml*, *hist2xml*, etc.) look in this directory for event-specific input which they convert to XML and place in the 'input' directory.
config	This directory may also be manually created by the operator to hold one or more event-specific configuration files. The ShakeMap programs look first to their command line for a user-specified configuration file, followed by a search of this directory. If neither is specified, the programs default to the configuration files found in <shake_home>/config.
save	Within this directory are subdirectories representing the version number of saved events. This directory will only exist if at least one version of an event has been saved with *setversion*.
regression	Exists only if *plotregr* is run. Contains the plots produced by that program.

Table 3.5 Miscellaneous Programs

src/xml/eq2xml	Probes the CISN database for information specific to a numbered event then writes an XML file in the event input directory describing the event.
src/xml/db2xml	Queries the CISN database for event-specific amplitudes then writes the appropriate XML.
src/xml/… ana2xml cdmg2xml csvnsmp2xml dig2xml hist2xml nsmp2xml scenario2xml	These programs take various formats of text data files and convert them to ShakeMap XML input. Most of them are no longer in use, but we leave them in the distribution to provide examples for operators to use in creating their own conversion programs.
src/xml… ciimcdi2xml ciimuscdi2xml knet2xml obs2xml	These programs access the CIIM (a.k.a. "Did You Feel It?") or other Intensity data files and produce XML data files containing intensity values. These programs are used at the NEIC for national and global ShakeMaps, but may serve as examples for operators who collect their own intensity data.
src/xml/cube2xml	Reads a CUBE-format earthquake file and generates an XML file in the event's input directory conforming to the earthquake.dtd spec.
src/xml… jbest2xml richt2xml	Take the programmatic output from estimate-generating programs and produces XML in the stationlist format. These programs produce a grid of estimates that was used in pre-3.5 *grind*. They are now obsolete.

3.10 References

Allen, T.I. & Wald, D.J., (2009). On the use of high-resolution topographic data as a proxy for seismic site conditions (vs30), Bull. Seism. Soc. Am., 99, 935–943.

Allen, T. I., Wald, D. J., and Worden, C. B. (2012). Intensity Attenuation for active crustal regions, J. Seismol., 16(3), 409-433.

Akkar, S., and J. J. Bommer (2007). Empirical Prediction Equations for Peak Ground Velocity Derived from Strong-Motion Records from Europe and the Middle East, Bull. Seism. Soc. Am., 97(2), 511-530.

Akkar, S., and J. J. Bommer (2007b). Prediction of elastic displacement response spectra in Europe and the Middle East, Earthquake Engng. Struct. Dyn., 36, 1275-1301.

Atkinson, G. M., and D. M. Boore (2006). Earthquake Ground-Motion Prediction Equations for Eastern North America, Bull. Seism. Soc. Am., 96(6), 2181-2205.

Atkinson, G. M., and D. M. Boore (2011). Modifications to Existing Ground-Motion Prediction Equations in Light of New Data, Bull. Seism. Soc. Am., 101(3), 1121-1135.

Atkinson, G. M., and S. I. Kaka (2007). Relationships between Felt Intensity and Instrumental Ground Motion in the Central United States and California, Bull. Seism. Soc. Am., 97, 497-510.

Atkinson, G. M., and D. J. Wald (2007). "Did You Feel It?" intensity data: A surprisingly good measure of earthquake ground motion, Seism. Res. Lett. 78, 362-368.

Boatwright, J., H. Bundock, J. Leutgert, L. Seekins, L. Gee, P. Lombard (2003). The Dependence of PGA and PGV on Distance and Magnitude Inferred from Northern California ShakeMap Data, Bull. Seism. Soc. Am., 93(5), 2043-2055.

Boore, D. M., W. B. Joyner, and T. E. Fumal (1997). Equations for estimating horizontal response spectra and peak acceleration from western North American earthquakes: A summary of recent work, Seismol. Res. Lett. 68, 128–153.

Boore, D. M., and G. M. Atkinson (2008). Ground-Motion Prediction Equations for the Average Horizontal Component of PGA, PGV, and 5%-Damped PSA at Spectral Periods between 0.01 s and 10.0 s. Earthquake Spectra, 24(1), 99-138.

Borcherdt, R. D. (1994). Estimates of Site-Dependent Response Spectra for Design (Methodology and Justification), Earthquake Spectra, 10(4), 617-653.

Campbell, K. W. (2003). Prediction of Strong Ground Motion Using the Hybrid Empirical Method and Its Use in the Development of Ground-Motion (Attenuation) Relations in Eastern North America, Bull. Seism. Soc. Am., 93(2), 1012-1033.

Chiou, B. S.-J., and R. R. Youngs, Abrahamson, N., and Addo, K. (2010). Ground-Motion Attenuation Model for Small-To-Moderate Shallow Crustal Earthquakes in California and Its Implications on Regionalization of Ground- Motion Prediction Models, Earthquake Spectra, 26(4).

Chiou, B. S.-J., and R. R. Youngs (2008). An NGA Model of the Average Horizontal Component of Peak Ground Motion and Response Spectra. Earthquake Spectra, 24(1), 173-215.

Dangkua, D. T. and C. H. Cramer (2011). Felt Intensity Versus Instrumental Ground Motion: A Difference between California and Eastern North America?, Bull. Seism. Soc. Am., 101(4), 1847-1858.

Faenza, L. and Michelini, A. (2010). Regression analysis of MCS Intensity and ground motion parameters in Italy and its application in ShakeMap, Geophys. J. Int., 180, 1138-1152.

Kanno, T., A. Narita, N. Morikawa, H. Fujiwara, and Y. Fukushima (2006). A New Attenuation Relation for Strong Ground Motion in Japan Based on Recorded Data, Bull. Seism. Soc. Am., 96(3), 879-897.

Garcia, D., S. Singh, M. Herraiz, M. Ordaz, J. Pacheco (2005). Inslab Earthquakes of Central Mexico: Peak Ground-Motion Parameters and Response Spectra, Bull. Seism. Soc. Am., 95(6), 2272-2282.

Motazedian, D., and G. Atkinson (2005). Ground-motion relations for Puerto Rico, Geol. Soc. Am. Special Papers, 385, 61-80.

Pankow, K. L., and J. C. Pechmann (2004). The SEA99 Ground-Motion Predictive Relations for Extensional Tectonic Regimes: Revisions and a New Peak Ground Velocity Rerlation, Bull. Seism. Soc. Am., 94(1), 341-348.

Rowshandel, B. 2006. Incorporating Source Rupture Characteristics into Ground-Motion Hazard Analysis Models, Seism. Res. Lett., 77, No. 6, 708-722.

Rowshandel, B. 2010. Directivity Correction for the Next Generation Attenuation (NGA) Relations, Earthquake Spectra, 26, No. 2, 525-559.

Wald, D. J., V. Quitoriano, T. H. Heaton, H. Kanamori (1999b), Relationships between Peak Ground Acceleration, Peak Ground Velocity and Modified Mercalli Intensity in California, Earthquake Spectra, 15, 557-564.

Worden, C.B., M.C. Gerstenberger, D.A. Rhoades, and D.J. Wald (2012). Probabilistic Relationships Between Ground-Motion Parameters and Modified Mercalli Intensity in California, Bull. Seism. Soc. Am., 102(1), 204-221.

Youngs, R. R., S.-J. Chiou, W. J. Silva, J.R. Humphrey (1997). Strong Ground Motion Attenuation Relationships for Subduction Zone Earthquakes, Seism. Res. Lett., 68(1), 58-73.

Zhao, J. X., J. Zhang, A. Asano, Y. Ohno, T. Oouchi, T. Takahashi, H. Ogawa, K. Irikura, H. K. Thio, P. G. Somerville, Y. Fukushima, and Y. Fukushima (2006). Attenuation Relations of Strong Ground Motion in Japan Using Site Classification Based on Predominant Period, Bull. Seism. Soc. Am., 96(3), 898-913.