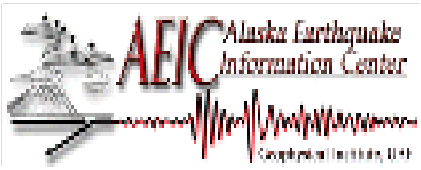

Alaska Earthquake Information Center

University of Alaska Fairbanks

The Trans-Alaska Pipeline ShakeMap system: Technical Report

AEIC Internal Report 2008-03

by Glenn Thompson, Artak Martirosyan, Mitch Robinson & Roger Hansen

June 12, 2008

Suggested citation:

Thompson, G., Martirosyan, A., Robinson, M. and Hansen, R., 2008, The Trans-Alaska Pipeline ShakeMap system: Technical Report. Alaska Earthquake Information Center, University of Alaska Fairbanks, AEIC Internal Report 2008-03.

This version was last revised: October 1, 2008

This document, any updates to it, and any additional information are available at:
http://www.aeic.alaska.edu/AEIC/internal/report/2008-03/

The Alaska Earthquake Information Center is a cooperative program between the Geophysical Institute of the University of Alaska and the U. S. Geological Survey with support from the Earthquake Hazards Programme.

Table of Contents

1 The TAPS ShakeMap system

1.1 Introduction

The ShakeMap software package was developed by the United States Geological Survey (USGS) for generating and distributing real-time ground-shaking maps in the aftermath of significant earthquakes. ShakeMap rapidly and automatically generates shaking and intensity maps, and combines instrumental measurements of shaking with information about local geology and earthquake location and magnitude to estimate shaking variations throughout a geographic area. The results are rapidly available via the Web through a variety of map formats, including Geographic Information System (GIS) shapefiles. These maps have become a valuable tool for emergency response, public information, loss estimation, earthquake planning, and post-earthquake engineering and scientific analyses.

The Alaska Earthquake Information Center (AEIC) has implemented the ShakeMap system for monitoring earthquake activity in Alaska. To make this possible, the AEIC has had to develop modules to get data from the its Antelope-based real-time data acquisition systems into the ShakeMap system. Thus there are two main components to the AEIC ShakeMap system:

(1) the Antelope-ShakeMap Interface, and

(2) the USGS ShakeMap modules (*"ShakeMap"*).

The ShakeMap configuration files, settings and libraries were also customised according to the regional specifics in Alaska.

The TAPS ShakeMap system mirrors the AEIC ShakeMap system, except that it is configured to focus on a corridor along the Trans-Alaska Pipeline, since this is the infrastructure of interest. Other differences are discussed in section 15.

2 System Architecture

The TAPS ShakeMap system is shown schematically in Figure 1. *rtexec* is a special Antelope program which controls other programs in a robust way. It continuously runs *shake_watch*, which monitors the summary event database for new candidate origins. It also continuously runs *shake_version* which monitors the MySQL database for new rows, and creates an Antelope database which mirrors it (this is then monitored by other processes which ultimately create the *pipe_event.xml* file, but this is covered elsewhere in this manual). The program *rtm* provides a convenient graphical user interface to help monitor that all processes under *rtexec* are running normally. This is shown in Figure 2.

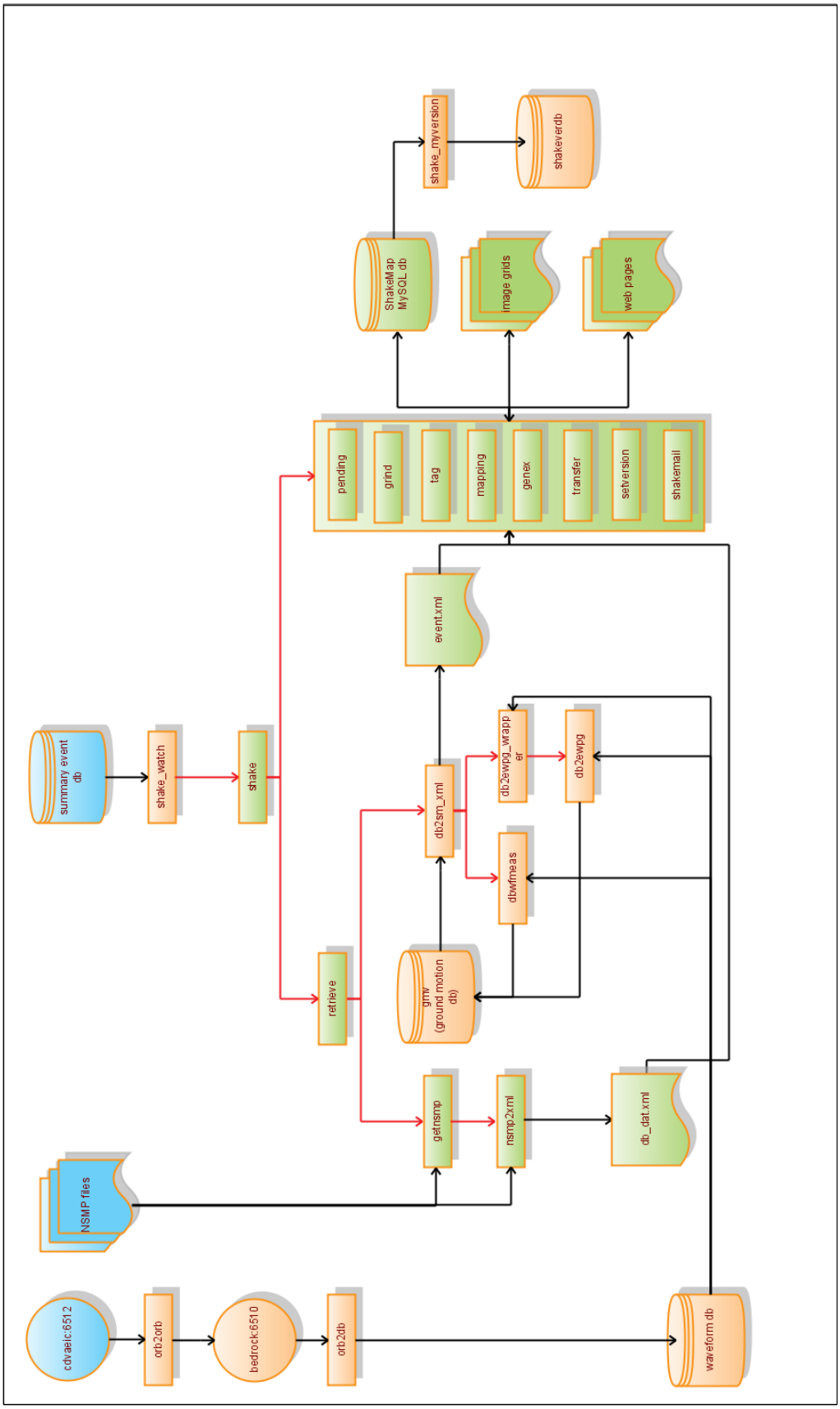
shake_myversion
shakeverdb
ShakeMap MySQL db
image grids
web pages
pending
grind
tag
mapping
genex
transfer
setversion
shakemail
event.xml
summary event db
shake_watch
shake
db2ewpg_wrapper
db2ewpg
db2sm_xml
dbwfmeas
retrieve
gmv (ground motion db)
getnsmp
nsmp2xml
db_dat.xml
NSMP files
cdvaeic:6512
orb2orb
bedrock:6510
orb2db
waveform db

Figure 1. Schematic diagram of the TAPS ShakeMap system architecture. Rectangles represent programs. Cylinders represent databases. Red arrows represent how programs call other programs. Black arrows represent data flow. The yellow objects comprise the Antelope-ShakeMap Interface. Green objects represent components of the USGS ShakeMap system.

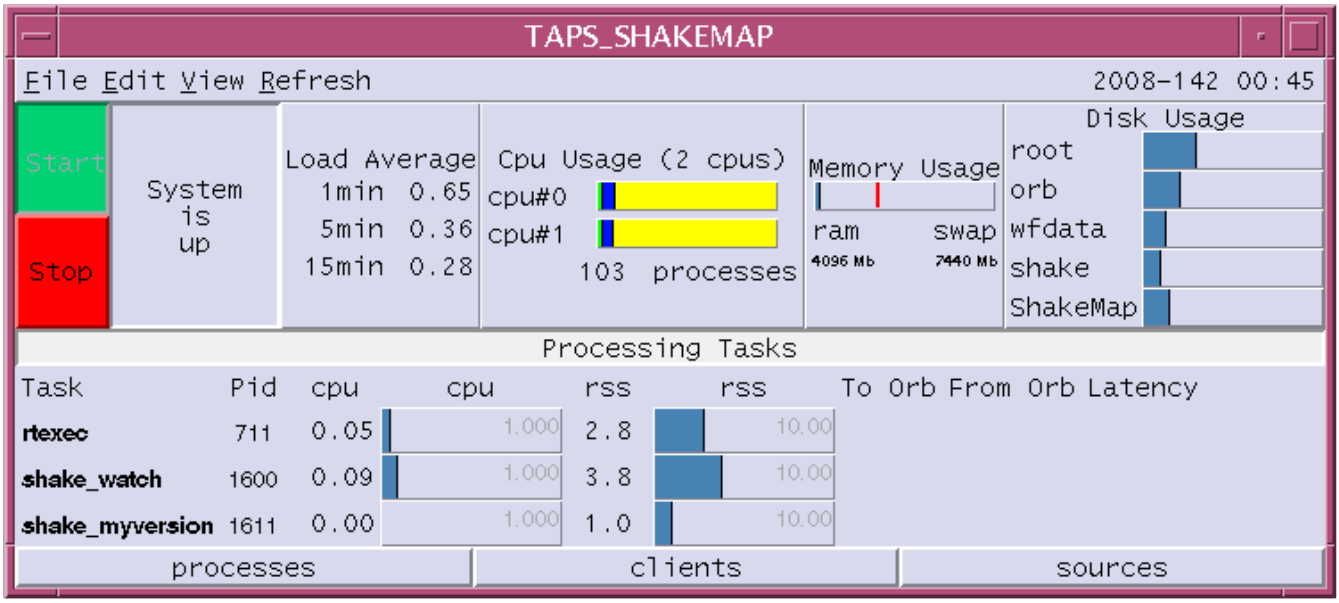

Figure 2: rtm, the graphical user interface which allows AEIC staff to monitor the Antelope-ShakeMap interface.

When a suitable origin is detected, *shake_watch* calls *shake*, which calls *retrieve* and other USGS ShakeMap programs. Retrieve calls *getnsmp* and *nsmp2xml*, which retrieve XML files for USGS stations in the area. Retrieve also calls *db2sm_xml*, which calls dbwfmeas to measure peak ground velocity and acceleration values. *db2sm_xml* also calls *db2ewpg_wrapper* which in turn calls *db2ewpg*. The aim of these programs is to make peak spectral ground displacement, velocity and acceleration measurements. All these peak ground motion measurements go into a customised Antelope database, from where *db2sm_xml* reads them, and writes relevant data out to XML files for consumption by the ShakeMap program *grind*. Other ShakeMap programs read from and write to the ShakeMap MySQL database, and generate web content. The details can be found in the ShakeMap manual, and are not shown in Figure 1.

All components of the TAPS ShakeMap system are located on bedrock except the summary event database (on kobuk) and a waveform orb (on cdvaeic) which feeds the waveform orb on bedrock.

The top level directories of relevance are:

Home directory (for user shakepipe):	/home/shakepipe (on bedrock)
Run directory for Antelope-ShakeMap Interface	/home/shakepipe/run (on bedrock)
ShakeMap root directory:	/usr/local/ShakeMapPipe (on bedrock)

3 The Antelope-ShakeMap Interface

The Antelope-ShakeMap Interface is a collection of programs and configuration files developed by the AEIC to implement the USGS ShakeMap system within the Antelope real-time data acquisition and processing environment that forms the backbone of AEIC operations. The main programs are listed in Table 1. The key directories are listed in Table 2. The key parameter files are described in Table 3.

Table 1: The Antelope-ShakeMap interface: main programs

program	**configuration file**	**Description**
shake_watch	*shake_watch.pf*	The main program that monitors the summary event database for new origins, checks origins for eligibility and initiates a ShakeMap generation (*shake*) or cancellation (*cancel*)
db2sm_xml	*db2sm_xml.pf*	Called by *shake*, this Perl program measures ground motion values and puts these into XML files. It calls the Antelope program *dbwfmeas* It also calls *db2ewpg_wrapper*. Both these calls results in rows being added to Antelope databases, and *db2sm_xml* reads these rows and appends data to a corresponding event.xml XML file. These XML files are used by *grind* for calculating ground motion estimation grids.
dbwfmeas	*dbwfmeas.pf*	Computes peak velocity and acceleration values for a station-channel for a given time range. These values are outputted to the *gmv* database.
db2ewpg_wrapper		Called by *db2sm_xml*, this Perl program reads calibration, instrument type, units and sampling rate from the waveform database, and then calls *db2ewpg*.
db2ewpg		Called by *db2ewpg_wrapper*, this C program reads waveform data for a station/channel, and calls a USGS subroutine which computes peak ground spectral displacement, velocity and acceleration values. These values are then inserted into an Antelope database.

shake_myversion		Monitors the shakeversion table of ShakeMap MySQL database and outputs an equivalent row in the Antelope database *shakeverdb*. This is then monitored by a system running elsewhere which creates a *pipe_event.xml* file.

Table 2: Important directories for the Antelope-ShakeMap Interface.

Directory	Comment
/home/shakepipe/run	Main directory for Antelope-related files and programs used for ShakeMap generation.
/home/shakepipe/run/bin	Contains programs.
/home/shakepipe/run/db	Several ShakeMap-related Datascope databases.
/home/shakepipe/run/dbdata	Contains the waveform database.
/home/shakepipe/run/pf	Parameter files.
/home/shakepipe/run/state	State files for several modules.

Table 3. Partial list of parameter files.

Configuration file	Settings
shake_watch.pf	Controls ShakeMap triggering and cancellation process
db2sm_xml.pf	Controls calculation of ground motion parameters
dbwfmeas.pf	Controls waveform measurements

3.1 shake_watch

This program monitors the summary database for new origins and checks these origins for ShakeMap eligibility. It is run from */home/run/rtexec.pf*. The command line syntax is:

```
/home/shakepipe/run/bin/shake_watch parameter_file
```

The parameter file (*/home/shakepipe/run/pf/shake_watch.pf*) sets the eligibility criteria, which includes:

1) geographical control: two polygons can be defined with different magnitude thresholds (Figure 3);

2) minimum number of associated phases;

3) calculated Modified Mercalli Intensity threshold.

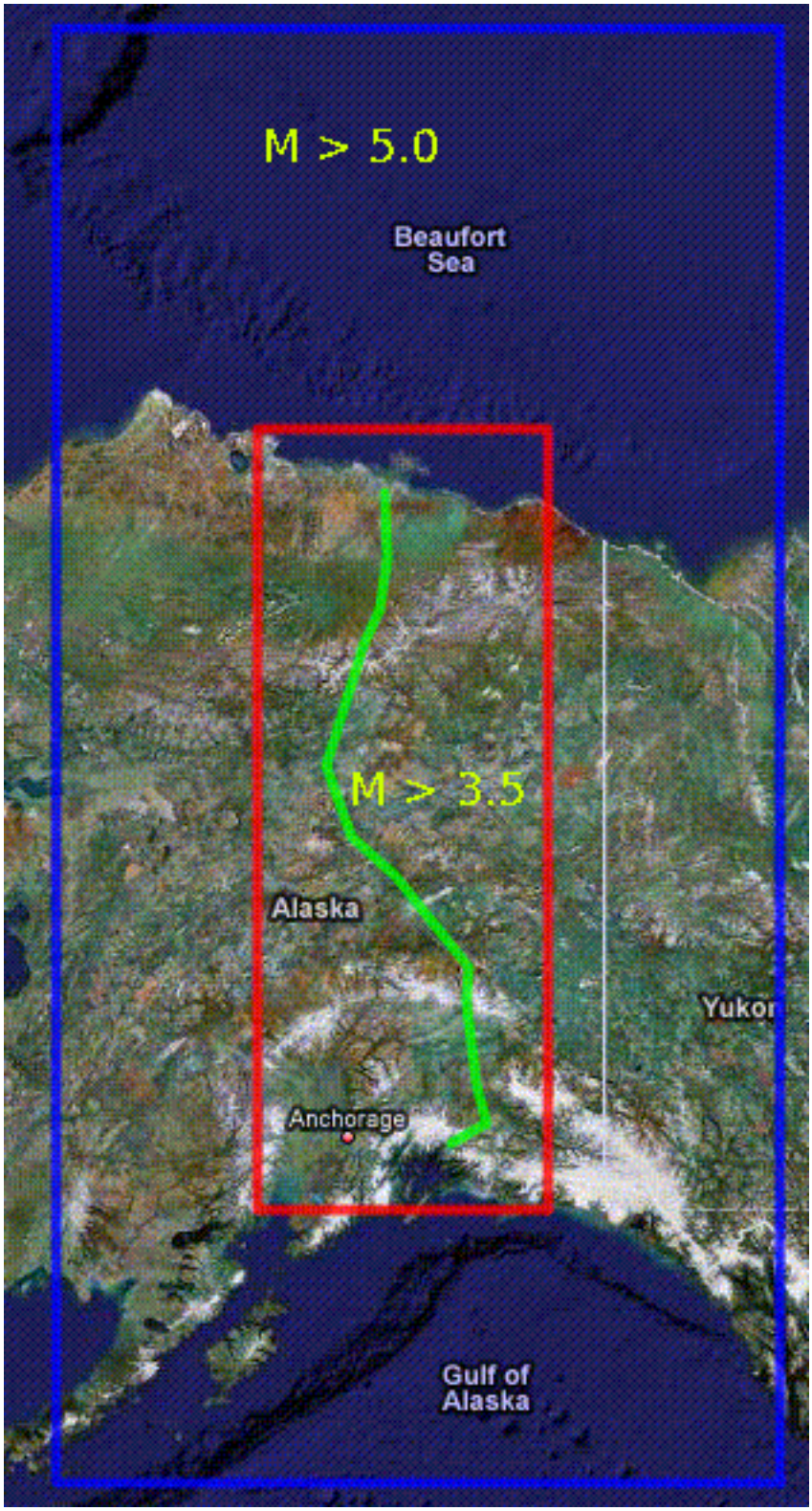

Figure 3: Shown here are the inner (red) and outer (blue) polygons and their associated

magnitude thresholds (3.8 and 5.0 respectively). The pipeline is indicated in green.

Other parameters are:

- del_time - waiting time in seconds for new origins to be processed. This delay is necessary as magnitude data are not usually available until a few seconds after an origin has been determined.
- auto_cancel – for switching automatic cancellations on and off;
- prefor_ex – to bypass the prefor check for the specified author;
- norids_max – maximum number of records in the *shake_watch* state file (*/home/shakepipe/run/state/shake_watch.state*). This file keeps one record for each ShakeMap in the following format (25124 25128 4), where the numbers are evid, orid and the source, respectively. Currently, the specified sources are:

  (1)AEIC automatic systems,

  (2)WCATWC,

  (3)NEIC,

  (4)AEIC analyst,

  *(5)*manual run, *and*

  (6)manual cancellation.

This is a listing of *shake_watch.pf.*

```
run_home         /home/shakepipe/run
bin_home         /usr/local/ShakeMapPipe/bin
database         /iwrun/sum/run/dbsum/dbsum
second_polygon   1 # 1=yes, 0=no
del_time         60
auto_cancel      0 # automatic cancellation 1=yes, 0=no
prefor_ex        #prefor exemption for this author
```

```
minmax &Arr{
nass_min        10
mmi_min         2.0
norids_max      200
}

ml_min          5.0
ak_polygon &Tbl{
        55,-135
        75,-135
        75,-160
        55,-160
}

ml_min2         3.5
ak_polygon2 &Tbl{
        60,-143
        71,-143
        71,-153
        60,-153
}

author_list &Arr{
oa_bk           1
oa_op           1
wcatwc          2
neic            3
UAF             4
manual          5
}
```

3.2 db2sm_xml

This module is called from *retrieve.conf* to perform waveform measurements, to establish an event's input directory, and to create *event.xml* (event data) and *db_dat.xml* (observational data)

files for the ShakeMap program *grind*. The command line syntax is:

```
db2sm_xml -event event_id
```

The program subsets the waveform database, calls *dbwfmeas* to calculate pga and pgv and writes these values in the data XML file. In addition, these values are first recorded in a temporary database table, then in the main ground motion database, *gmv*. The parameter file is *db2sm_xml.pf*, which contains the channel names, waveform window settings and other parameters. One of the parameters is ev_type, which enables (1) or disables (0) event type determination. If enabled, all events are classified into one of the three categories: crustal, intraplate and interface, by calling *earthquake_type.pl*. The algorithm is based on the position of the hypocenter with respect to the Alaska-Aleutian Megathrust. This is required for *grind* to utilize proper attenuation model.

Spectral maps are only created if the program grind is run with the –psa flag. By default, this flag is not included. Default behavior can be changed by setting up variable flags in shake.conf which indicate the magnitude range for which to apply a different behavior. While this isn't part of the Antelope-ShakeMap Interface, its important to know about as it is one other place where there is a magnitude threshold. The line looks like:

```
variable_flags : grind 0.0 9.9 –psa
```

This indicates that grind will be run for all events with a magnitude between 0.0 and 9.9.

4 The (USGS) ShakeMap system

The ShakeMap package is open-source software written in Perl by the USGS. A detailed description is not attempted here. The installation instructions, user guide and technical guide are provided in the ShakeMap Manual [Wald et al.], available from http://pubs.usgs.gov/tm/2005/12A01/.

The ShakeMap modules are listed in Table 4. Each module generally has its own configuration file, which allows its behaviour to be customised. The main module is *shake*, which is a wrapper. Its configuration file lists the modules that will be run everytime a ShakeMap is initiated. Currently the order is: *retrieve, pending, grind, tag, mapping, genex, transfer, setversion* and *shakemail.*

The AEIC does not modify the USGS code (unless its absolutely necessary as a stop-gap measure), since any changes made will have to be remerged which each new release of the ShakeMap software by the USGS, creating a lot of overhead. Moreover, the AEIC wants to utilise the USGS code as-is, to keep in line with USGS best practice.

The key directories are listed in Table 5, and the main configuration files described in Table 6.

Table 4. Native ShakeMap programs. These are all in /usr/local/ShakeMapPipe/bin.

Program	*Configuration file*	Comment
shake	*shake.conf*	The main ShakeMap program; a wrapper program that calls other ShakeMap programs.

retrieve	*retrieve.conf*	A wrapper code that calls other programs to retrieve data and produce data XML files
pending	*pending.conf*	Sends a new home page to the web site to indicate that an event is being processed.
grind	*grind.conf*	Reads the data files from the event's input directory and generates grid files with interpolated ground motion values.
mapping	*mapping.conf* *colors.conf*	Reads the grids generated by grind and makes PostScript maps of ground motion and shaking intensity.
genex	*genex.conf* *web.conf*	Creates JPEG files from PostScripts, builds web pages, and generate GIS and other files for export via the web or FTP.
addon	*addon.conf*	Creates and copies a QDDS-formatted file to a local QDDS directory (currently not implemented).
transfer	*transfer.conf*	Transfers the output created by genex to the web and ftp sites.
setversion		Manipulates the version information for ShakeMaps and preserves versions as requested.
shakemail	*shakemail.conf*	Sends email notifications of ShakeMap generations and cancellations.
cancel	*shake.conf*	Undoes the effect of shake: it removes the event from data directory and the web. It can be called automatically by shake_watch if a revised origin for an earthquake event shows that it no longer is eligible for the ShakeMap system, or it can be called manually.
getnsmp		Associates NSMP station data with ShakeMap events and runs nsmp2xml.
nsmp2xml		Generates data XML file from NSMP XML files.

Table 5. Important ShakeMap directories.

Directory	Comment
/usr/local/ShakeMapPipe/bin	All ShakeMap executables plus several third-party programs
/usr/local/ShakeMapPipe/config	Configuration files of native ShakeMap programs

/usr/local/ShakeMapPipe/lib	Contains Perl library modules for various ShakeMap programs, including site correction data and web page templates
/usr/local/ShakeMapPipe/data	Repository of all event data and processed files. Each event has its own subdirectory such as data/12345 for event 12345.
/usr/local/ShakeMapPipe/web	Contains ShakeMap web pages

Table 6. Partial list of configuration files.

Configuration file	Settings
shake.conf	Main ShakeMap configuration file. Contains the list of programs to run, default flags, magnitude dependent flags, magnitude thresholds for spectral response maps and saving versions, etc.
retrieve.conf	List of programs run before grind to generate observational data XML files. Currently contains two programs: db2ShakeMap_xml and getnShakeMapp
grind.conf	Controls generation of ground motion grids. Many settings including grid parameters, size, stations to ignore, attenuation models, etc.
mapping.conf	Controls the appearance of ShakeMaps, colors, sizes, markers, additional map features,etc.
genex.conf	Controls generation of web pages and various output files for third party software (GIS, Hazus, Google Earth)
shakemail.conf	Specifies formats and addressees of email and pager notifications

5 Databases

Table 7. Databases relevant to the TAPS ShakeMap system.

Path	**Description**
/iwrun/sum/run/dbsum/dbsum	Summary event database.

/home/shakepipe/run/dbdata/archive	Waveform database for ShakeMap; contains only broadband and strong-motion channels.
/home/shakepipe/run/db/gmv	Database containing ground motion measurements.
/home/shakepipe/run/db/gmv_'evid'.wfmeas	Temporary database table for ground motion values.
/export/bedrock2/mysql/var/shakemappipe	The ShakeMap MySQL database.
/home/shakepipe/run/dbshakeversion/shakeverdb	An Antelope copy of the MySQL table shake_version.MYD table of the MySQL database. Created by the *shake_myversion* process.

6 Accounts and Passwords

There are three separate accounts relevant to the ShakeMap system. There is a username and password needed to modify the software or its configuration. There is a username and password needed to access the ShakeMap website. There is also a username and password needed to access and/or modify the ShakeMap MySQL database (it should not normally be necessary to modify this database manually, although it was needed to remove old scenarios from the database: see section 3.2.10).

Table 8: Usernames and passwords relevant to the TAPS ShakeMap system.

Username	**Password**	**Purpose**
shakepipe	(same as shake)	Solaris account to modify ShakeMap software and its configuration. There is no ssh access. There is ssh access for shake however, and su can be used from there.
shakepipe	5hakEp1pe	Needed to access ShakeMap website, *http://www.aeic.alaska.edu/~shakepipe/shake*.
shakepipe	(in the *mydb.conf* file)	MySQL account for the ShakeMap MySQL database: gives write access to create/modify/add/remove tables and records.
dlmon	dlmonquake	Needed to access the TAPS monitoring website *http://www.aeic.alaska.edu/~dlmon/.*

7 TAPS ShakeMap website

The URL for the ShakeMap website is given in Table 8. The website is password-protected. When logged on, the Home Page will be displayed, showing the most recent event, and recent significant events [Figure 4]. To see a full list of events, click the "Map Archive" link (upper left). The user is then presented with a table of events with columns: event id, name/epicenter, date, time, latitude, longitude and magnitude [Figure 5]. By clicking on any event in this list, a new page will show up with the instrument intensity map and links to other maps, and downloads. Returning to the archive web page, there is a menu along the top with links to archives from previous years, and also to earthquake scenarios. Clicking on the earthquake scenarios link will display *scenario.htm*, with a list of all the scenarios generated for this ShakeMap system [Figure 6]. By clicking on a scenario, the corresponding instrumental intensity map is displayed [Figure 7], along with links to the corresponding peak ground acceleration, peak ground velocity and spectral response maps. There is also a link to downloads, which includes jpg and postscript versions of the maps, other versions of the maps specifically for media purposes, raw grid files, GIS shapefiles for HAZUS, KML for display in GoogleEarth, and station lists [Figure 8].

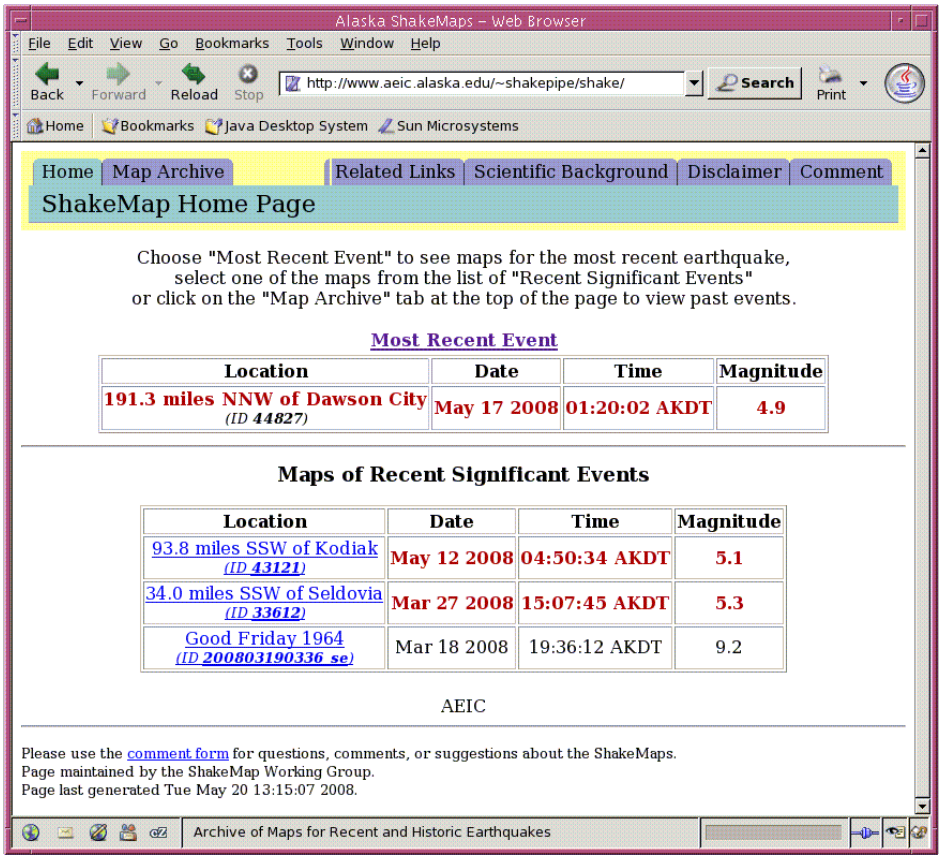

Figure 4: The TAPS ShakeMap website.

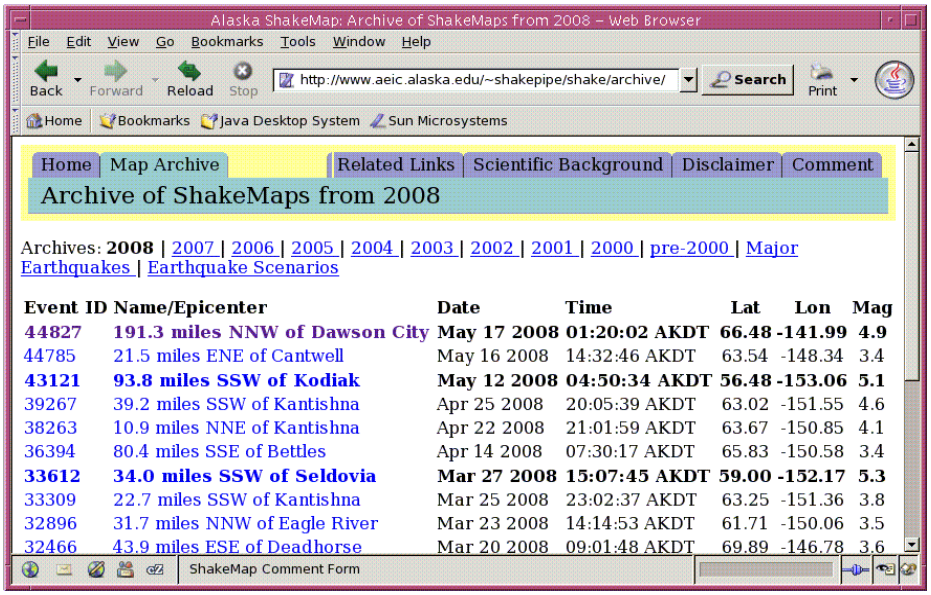

Figure 5: The TAPS ShakeMap archive webpage. This lists all the events a ShakeMap currently exists for, in the current year. It also has links previous years, as well as scenarios.

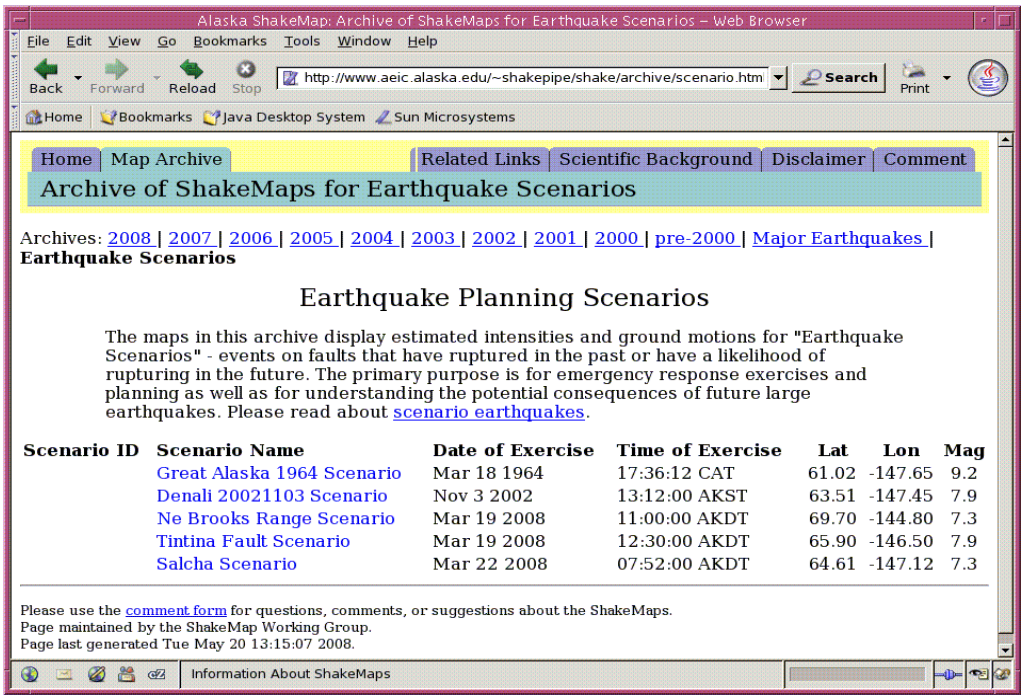

Figure 6: Scenarios index webpage.

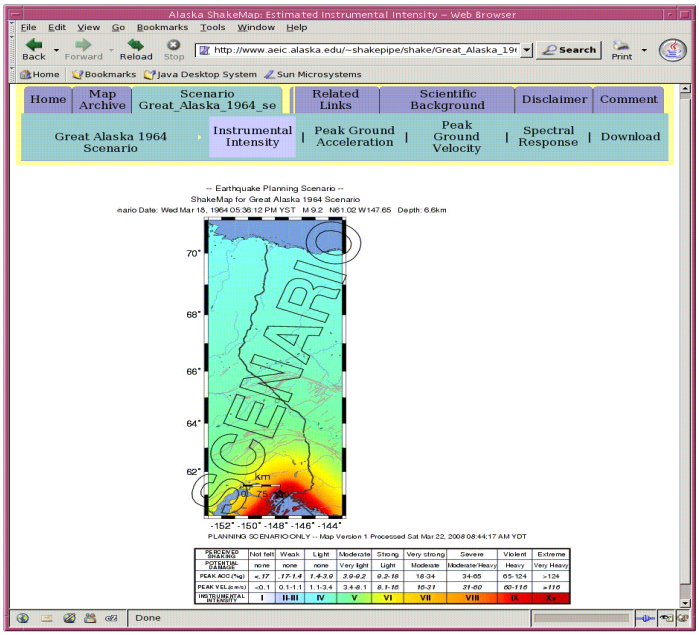

Figure 7: Instrumental intensity webpage for a particular scenario. An instrumental intensity webpage for an event is essentially identical to this.

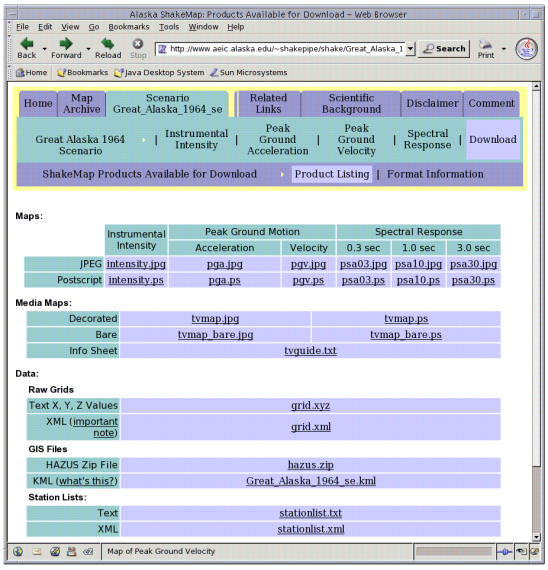

Figure 8: ShakeMap download webpage for a particular scenario. A download webpage for an event is essentially identical to this.

8 Generating a ShakeMap manually

Manual Run: In order to manually run a ShakeMap, it is necessary to login to bedrock (as user shakepipe) and run the following command:

```
shake -event evid -once_only -default_fl
```

where evid is the event id. Use -help flag to get more information about the program:

```
shake -help
```

Flag -dryrun may be used to show the list the commands that will run without actually running them:

```
shake -event evid -once_only -default_fl -dryrun
```

In order to prevent manual runs from being overwritten, it is necessary to edit the shake_watch state file (*/home/shakepipe/run/state/shake_watch.state*) by setting 5 for the manual run next to the evid and orid, for example (30234 30237 5) for evid=30234.

9 Finite Faults

By default, the earthquake source in ShakeMap calculations is assumed to be a point. However, it is possible to incorporate extended source geometry by placing an ASCII file with fault coordinates in the event's input directory. The file must contain set of (latitude, longitude) points defining the surface rupture, and the filename must end in "*_fault.txt*". See ShakeMap manual for more details.

This is a file which defines the lat/lon line segments of a particular fault. The > character is used to separate line segments. The name can be anything, but it must end with *_fault.xml*, e.g. *example_scenario_fault.xml*. A critical parameter used to determine the intensity of the shaking at each grid point is the distance from the source to that grid point. If there is no fault file, a point source is used for the earthquake, which is fine for small earthquakes but isn't going to give a very good representation for larger events.

If it is desired to translate one rupture (such as Denali 20021103) to another location, use the Matlab program *load_faultfile.m* in */home/glenn/PROJECTS/PIPELINE*.

The fault rupture can be plotted on the map by modifying the configuration file *mapping.conf*.

10 Cancelling a ShakeMap manually

To manually cancel a ShakeMap:

```
/usr/local/ShakeMapPipe/bin/cancel -event evid
```

In order to prevent a manual cancellation from being overwritten, it is necessary to edit the *shake_watch* state file (*/home/shakepipe/run/state/shake_watch.state*) by setting 6 cancellation next to the evid and orid, for example (30234 30237 6) for evid=30234.

11 Generating a Scenario

In general, the scenario earthquakes should be a replication of a historical strong event or a hypothetical event consistent with the Alaska seismic hazard assessments (Wesson et al. 1999).

The scenario settings are controlled by several configuration files, including *shake.conf*, *grind.conf*, and *mapping.conf*. While a description of scenario generation is given in the main ShakeMap Manual, those instructions do not work, and the following procedure has been developed.

Configuring a scenario consists of the following steps:

1. Make a name for the new scenario event, ending in '_se' to indicate it is a scenario. For the sake of these instructions, call this *example_scenario_se*.
2. Under the directory *data*, create a directory called e*xample_scenario_se*.
3. cd to that directory, and create a directory called *input*.
4. cd to that directory and create an *event.xml* and a *db_dat.xml* file. Examples can be copied from other events/scenarios in the data directory. The *db_dat.xml* file probably will not need altering. The *event.xml* file will. Make sure it has the id set to the scenario name (*example_scenario_se*). Give it the appropriate coordinates, origin time, magntude, description, creation time etc.
5. Optionally create a fault file (see 3.2.9).
6. Optionally create an *estimates.xml* file. This contains real data values if they are available. Its simply a way to allow real data to be used as part of the ShakeMap model generated for this scenario.

Now a scenario has been configured, it must be run. This ShakeMap manual states the following command should work:

```
shake -event example_scenario_se
```

However, this causes *shake* to crash, complaining that the *retrieve* program needs to be run. The problem is that *shake* is not correctly interpreting the *shake.conf* file in the *config* directory. This file tells *shake* to ignore the program *retrieve* (and others) when it is running a scenario as opposed to a real event (for a scenario, there is no real data available, unless it is provided via the optional *estimates.xml* file). USGS suggested the following possible solutions:

1. Make a local copy of the *config* directory in each scenario directory, and place the appropriate version of *shake.conf* there. This was tried and did not work.
2. Modify the *shake.conf* file so that *retrieve* can be ignored. However, this would then prevent real events being processed (they wouldn't get any data).

A workaround based on the latter is a tolerable solution. A script, *run_scenario.csh* in */usr/local/ShakeMapPipe* was written which temporarily moves a copy of a *shake.conf* file configured for scenarios (*shake.conf.scenario*) into the (main) config directory any time a

scenario is run. When the scenario has finished running, it moves back a copy of the *shake.conf* file configured for real events (*shake.conf.real*) immediately afterwards. This is how to call it:

```
run_scenario.csh example_scenario_se
```

To run multiple scenarios a batch script can be written. The script *scenario_wrapper.csh* does this. It is also in */usr/local/ShakeMapPipe*.

If everything worked, all scenarios should appear in */usr/local/ShakeMapPipe/web/shake/archive/scenario.html*.

One more step is still required to enable the scenario to be digested by downstream software for testing purposes. XML describing the scenario must be added to the file */Seis/web/dlstat2xmldir/events/pipe_scenario.xml*. The format of this file follows:

```
<quakes>
   <event>
      <event_id>Great_Alaska_1964_se</event_id>
      <origin_time>03/19/1964  03:36:12.000</origin_time>
      <latitude>61.0170</latitude>
      <longitude>-147.6480</longitude>
      <magnitude>9.20</magnitude>
      <shakemap>y</shakemap>
      <arrivals>91</arrivals>
      <review>y</review>
      <timestamp>3/18/2008  21:24:44.024</timestamp>
   </event>
   <event>
     etc ...
     ...
   </event>
</quakes>
```

This file can be accessed with the URL *http://www.aeic.alaska.edu/~dlmon/events/pipe_scenario.xml*.

12 Cancelling a Scenario

According to the ShakeMap manual, scenarios should be deleted in the same way as events, for example:

```
cancel -event example_scenario_se
```

Once again this did not work. Instead it must be done manually. There are two steps:

1. delete the directory structure
2. delete the scenario from the MySQL database.

Deleting the directory structure:

This is a trivial process. Simply change to the relevant directory (e.g. data/example_scenario_se). Then remove all subdirectories with the exception of the input directory (this allows you to rerun the scenario at a later date if you wish; all other subdirectories can safely be destroyed).

Deleting a scenario from the MySQL database:

1. Login to MySQL:

   `mysql -u shakepipe -p` (enter password from *mydb.conf* file)

2. You should now get the mysql prompt:

   mysql>

3. Change to the appropriate database (in this case, shakemappipe):

   ```
   use shakemappipe;
   ```

4. (Optional) Show the tables:

   ```
   show tables;
   ```

(should list: earthquake, server, shake_lock, shake_runs, shake_version)

5. (Optional) View the schema for the earthquake table:

   ```
   describe earthquake;
   ```

6. (Optional) View records that contain this scenario:

```
   select * from shake_runs where evid = 'example_scenario_se';
   select * from shake_version where evid = 'example_scenario_se';
   select * from earthquake where evid = 'example_scenario_se';
```

7. Delete those records!

```
   delete from shake_runs where evid = 'example_scenario_se';
   delete from shake_version where evid = 'example_scenario_se';
```

```
delete from earthquake where evid = 'example_scenario_se';
```

8. Commit those changes:

```
commit;
```

9. Exit MySQL

```
quit
```

13 Observations and Attenuation Models

The ground shaking grids for ShakeMaps are produced on the basis of observed ground motion values (maximum peak ground accelerations and velocities), complemented by calculated ones using empirical attenuation relationships. Only horizontal strong-motion (BNN, BNE, HNN, HNE) and horizontal broadband (BHN and BHE) channels are utilized for ShakeMap calculations. There are a number of settings in *grind.conf* that control which observations are included in calculations and how the grids are calculated. Figure 9 shows the distribution of strong-motion and broadband stations used in ShakeMap generation in Alaska.

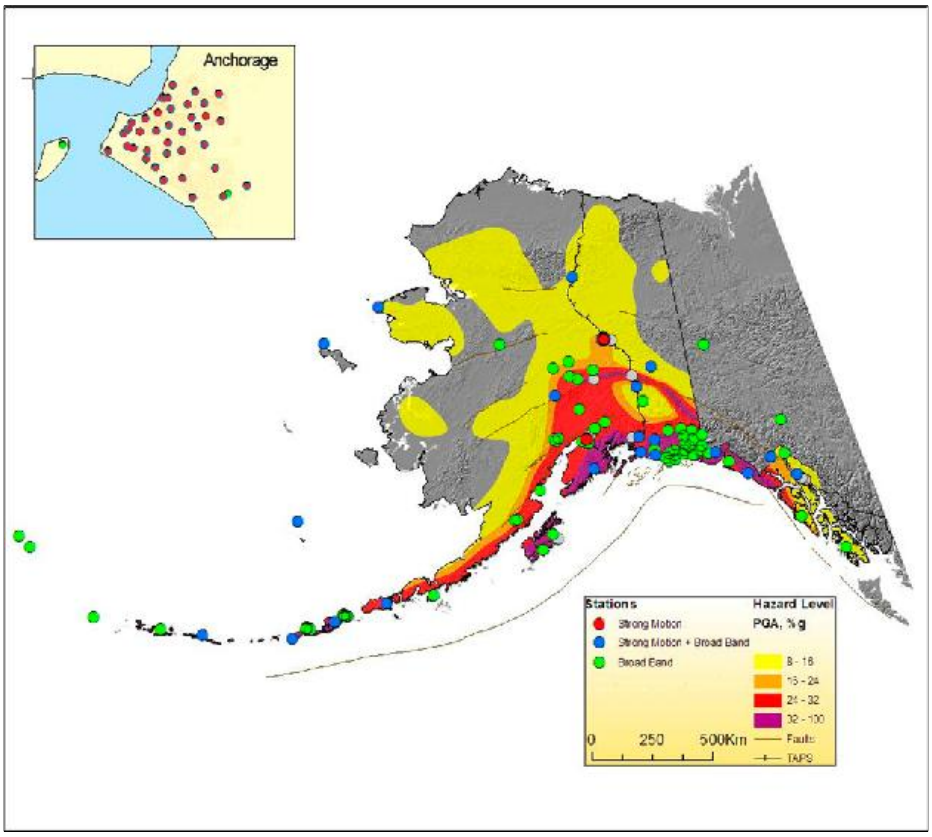

Figure 9. Map of strong-motion and broadband stations in Alaska on the background of Alaska seismic hazard map (pga, 10% in 50 years).

Peak ground values are calculated by *dbwfmeas*. The settings are controlled by

/home/shakepipe/run/pf/dbwfmeas.pf. The spectral calculations for periods 0.3, 1.0 and 3.0 sec are done by */usr/local/ShakeMapPipe/bin/db2ewpg*. The corresponding waveforms are preliminarily filtered by 5th order high-pass filter with 0.1 Hz corner frequency.

The velocity time series are differentiated in order to obtain accelerations and the acceleration time series are integrated in order to obtain velocities. Only the maximum of two horizontal components are utilized in grid computations. For sites, where the strong-motion and broadband sensors are collocated, the velocity data is used only for the velocity computations, and acceleration data only for accelerations.

The attenuation relationships for ground motion calculations are defined in *grind.conf*. The following models are currently used:

- Boore et al. (1997) model for the crustal events with M>5.3.
- Youngs et al. (1997) model for subduction-zone events.
- ShakeMap Small Regression model for shallow events with M<=5.3.

14 Customized Content

The most important regional characteristic for ShakeMap calculations is the uppermost 30m average shear-wave velocity (Vs30) grid for the whole state. The Vs30 values are required to correct the observed and computed ground motions according to the local geological conditions. We are currently using a vs30 grid derived from topography by correlating the Vs30 values with the gradient of the surface. This grid is compiled by the USGS.

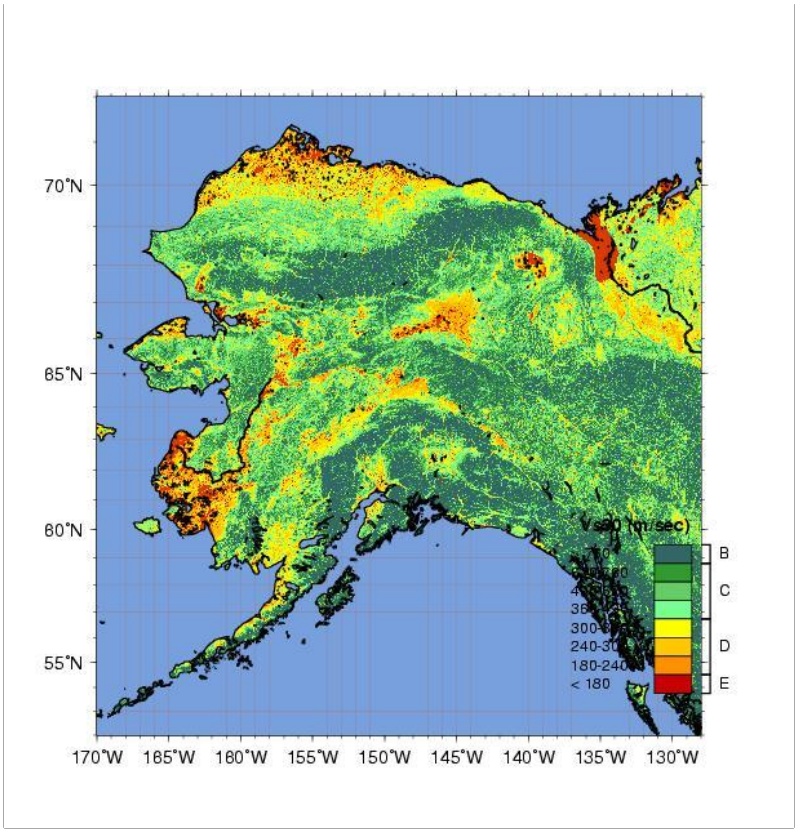

Figure 10. Vs30 map, currently used in ShakeMap calculations.

ShakeMap requires the binary version of this file (*/usr/local/ShakeMapPipe/lib/sitecorr/ak_vsgrid.bin*), which can be obtained by converting the ASCII version to binary by using */usr/local/ShakeMapPipe/bin/qtmlatlon2bin*. A color-coded map of Vs30 is shown in Figure 10.

Vs30 values are used to determine site coefficients. Ideally, these numbers should be customized according to the regional characteristics; right now, however, we are using the coefficients derived from California earthquake data. They are set in */usr/local/ShakeMapPipe/lib/sitecorr/site_corr_cdmg.dat*.

There are numerous settings in the ShakeMap configuration files that are customized according to the regional specifics, including ground motion calculation settings in *grind.conf*. In addition, several auxiliary region-specific files are used for mapping, such as Alaska roads, faults, the pipeline route, etc.

15 Differences between the TAPS ShakeMap system and AEIC ShakeMap system

There are several important differences between the TAPS ShakeMap system and the AEIC ShakeMap system:

- All calculations for the TAPS ShakeMap are done for a fixed, narrow corridor along the pipeline route (specified in *grind.conf*).
- Two polygons, which control the geographical eligibility of earthquakes for ShakeMap, are different from the main system. See */home/shakepipe/run/pf/shake_watch.pf,* Figure 3 or section 17 for these polygons.
- Ground motion calculations are done for bedrock site conditions.
- Magnitude and MMI thresholds will be much higher (when operational) than those for the AEIC ShakeMap system.

16 Known Issues

The following list contains certain problems in the ShakeMap software and related programs, and other potential problems that the ShakeMap system may encounter in the future:

- ShakeMap doesn't work without observational data, i.e. without the **_dat.xml* file. Therefore, when no observations are available, *db2sm_xml* copies the following blank data file (*/home/shakepipe/run/pf/dummy_dat.xml*) to the event's input directory.
- ShakeMap archival web page doesn't automatically create a new link for the new year. In order to create this link, it is necessary to manually introduce the new year in the archival page template: */usr/local/ShakeMapPipe/lib/genex/web/config/archivepages.xml.*
- At high latitudes (~70N) the scalebar of ShakeMap doesn't fit into the map frame. In order to fix this, line 921 in program mapping was modified from "my $sb_offset = 1.0;" to "my $sb_offset = 1.5;". This is the only change done in the native ShakeMap codes.
- Contrary to the ShakeMap manual instructions, adding "_se" at the end of the event id doesn't automatically add –scenario flags in the ShakeMap programs.
- When a scenario is generated, *shake* calls *retrieve* even though it is configured not to do so, with the result that no scenario is created. A workaround is to temporarily introduce a new *shake.conf* file, which is semi-automated by using the script *create_scenario.csh*, as described under "3.2.11 Generating a Scenario".

- When a scenario is cancelled, it is not automatically removed from the MySQL database as it should be. Follow instructions under "3.2.12 Cancelling a Scenario".
- The spectral calculation algorithm has not been comprehensively tested, for example, in case of a strong event with large number of observations, or when there are problems with waveforms such as spikes, missing or late samples, etc. Also, it is necessary to check the reliability of spectral calculations by comparing the results from collocated broadband and strong motion channels.
- There is no mechanism to identify if the broadband channels are clipped, so ShakeMap in such cases will be based on amplitudes.
- While running ShakeMap over ssh connection, *grind* doesn't properly interpret the vs30 grid file. It is necessary, before running the ShakeMap programs, to run the following command:

```
setenv LC_ALL
```

- Currently, both ShakeMap and Antelope software support the same Perl version. In future releases, however, these versions may be different (at least for certain time period), which can create problems in Perl libraries and influence ShakeMap operations.
- All attenuation models are based on Mw magnitude while mostly Ml is used in ShakeMap calculations. Ml and Mw may differ considerably at larger magnitudes.

17 Current settings and other info

- Current ShakeMap version: 3.2
- Triggering parameters to generate a ShakeMap:
  i. Number of associated arrivals (nass) > = 10
  ii. MMI >= 2.0
  iii. magnitude >= 3.5 for an event inside the rectangle from (-153,60) to (-143,71), or

iv. magnitude >= 5.0 for an event inside the rectangle from (-160,55) to (-135,75).

- Magnitude threshold for saving all ShakeMap versions: M>=5.0
- Magnitude threshold for spectral calculations: M>=3.8
- Delay time (number of seconds shake_watch will wait for a magnitude calculation) 60.
- Automatic cancellation OFF (ShakeMaps are never automatically cancelled)

18 Creating pipe_event.xml

pipe_event.xml is an XML file which resides at /Seis/web/dlstat2xmldir/events/. It contains a list of all events in the summary event database which meet the subsetting criteria defined for the pipeline. Whenever a ShakeMap version is updated, the ShakeMap tag for that event in pipe_event.xml is updated with the latest version number. These are all data driven processes.

The creation of this pipe_event.xml is complicated. The program /home/shakepipe/run/bin/watchshake_version/mycommand/mycommand (running under rtexec on bedrock) monitors the version table of the ShakeMap MySQL database and writes out an equivalent Antelope database table at /home/shakepipe/dbshakeversion/shakeverdb whenever a change is made. By subsetting the summary events database using criteria which define the geographic range and minimum number of arrivals of interest, db2dbsubset creates an Antelope database of events of interest to TAPS. Another program monitors these two Antelope databases, and merges them (using db2xml as part of this process) to create pipe_event.xml.

19 Testing the TAPS EMS ShakeMap System

Between July and September a number of different simulations were performed to test the interaction of the TAPS ShakeMap system, the pipe_event.xml and the ignore_events.xml files with downstream reporting software created by Jimmy de la Garza. The final and most comprehensive test is discussed below, and can be easily repeated at any time. It consists of a 19 step sequence which samples the following possibilities in different orders:

(i) A new qualifying event
(ii) A new qualifying preferred origin for an existing event
(iii) A new ShakeMap for an existing preferred origin
(iv) A new manually modified ShakeMap for an existing preferred origin
(v) A new preferred origin for an existing event which no longer qualifies
(vi) Manual cancellation of an existing event

It should be pointed out that this type of simulation does not test the systems which ultimately create the pipe_event.xml file, or the ignore_events.xml file. It simply causes those files (or proxies for them) to be modified in ways that reflect what is designed to happen if scenario

earthquakes were to happen in reality. Ten scenario origins are used in the test sequence, and these are listed in Table 10.

Table 10. Test sequence.

Step	Event Name	Origin Version	ShakeMap Version	Remarks
1	Denali 7.3	1	0	Initial notification, event info only; no ShakeMap
2	Coldfoot 4.2	1	0	Initial notification, event info only; no ShakeMap
3	Denali 7.3	1	1	ShakeMap grid file Version 1 available
4	Prince Wm Sound 4.1	1	0	Initial notification, event info only
5	Tsaina 6.7	1	0	Initial notification, event info only
6	Denali 7.3	2	1	Revised magnitude, location; ShakeMap grid file remains at Version 1
7	Coldfoot 4.2	1	0	Event cancelled
8	Tsaina 6.7	1	1	ShakeMap grid file Version 1 available
9	Denali 7.3	2	2	ShakeMap grid file revised to Version 2
10	Denali 7.3	2	3	Manual execution of ShakeMap; grid file revised to Version 3
11	Salcha 4.2	1	0	Initial notification, event info only
12	Tsaina 6.7	2	2	ShakeMap grid file revised to Version 2
13	Prince Wm Sound 4.1	2	0	Revised magnitude, location
14	Tsaina 6.7	3	2	Event moved out of bounds of interest.
15	Denali 7.3	3	3	Revised magnitude, location; ShakeMap grid file remains at Version 3

16	Prince Wm Sound 4.1	2	0	Event cancelled
17	Denali 7.3	3	4	ShakeMap grid file revised to Version 4
18	Salcha 4.2	1	0	Event cancelled
19	Denali 7.3	3	0	Event cancelled

Table 11: Origins used as part of the test sequence..

Origin No.	Origin Name	Mag.	Depth	Latitude	Longitude	
1	Denali 1	7.3	4.2	63.5141	-147.4529	
2	Coldfoot 1	4.2	10	67.2230	-149.8090	
3	Prince William Sound 1	4.1	10	61.0170	-147.6480	
4	Tsaina 1	6.7	10	61.2670	-145.6800	
5	Denali 2	7.4	5.2	63.5131	-147.4539	
6	Salcha 1	4.2	10	64.6100	-147.1200	
7	Tsaina 2	6.8	11	61.2680	-145.6810	
8	Tsaina 3	6.9	11	61.2670	-125.6820	
9	Prince William Sound 2	4.2	10	61.0160	-147.6470	
10	Denali 3	7.5	5.2	63.5161	-147.4549	

To automate this test sequence, the program */home/shakepipe/TAPS_TEST/test_wrapper2.pl* was written. It is very simple to re-run the tests:

1. Logon on bedrock as user shake (ssh –l shake –X bedrock)

2. Switch user to shakepipe (su shakepipe)
3. Change directory to /home/shakepipe/TAPS_TEST (cd /home/shakepipe/TAPS_TEST)
4. Issue the following command: unset LC_ALL
5. Call Jimmy de la Garza
6. Run test_wrapper2.pl, discussing the output at each stage with Jimmy

References

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

Youngs, R. R., S.-J. Chiou, W. J. Silva, and J. R. Humphrey (1997). Strong ground- motion relationships for subduction zones, Seism. Res. Letters, 68, 58-73.

Wesson, R., Frankel, A., Mueller, C., and Harmsen, S. (1999) Probabilistic seismic Hazard Maps of Alaska, USGS Open-File Report 99-36.

Wald, D. J., Worden, B. C., Quitoriano, V., Pankow, K. L, ShakeMap Manual: Technical Manual, User's Guide, and Software Guide, http://pubs.usgs.gov/tm/2005/12A01/.

Appendix 2: Listing of test_wrapper2.pl

This is a Perl script that was hurriedly written to simulate the AEIC-side parts of the TAPS EMS system in response to test sequence 2, as defined by Jimmy de la Garza.

To create and modify XML documents, here documents and sed-like Perl algorithms were implemented in new subroutines.

```
#!/usr/bin/perl
# Glenn Thompson 20080821
# Testing for the TAPS interface

################################################################################
################################

# INITIALISE PATHS

# Test directory
$testdir = "/home/shakepipe/TAPS_TEST";

# ShakeMap home
$smhome = "/usr/local/ShakeMapPipe";

# xml directory
$xmldir = "/Seis/web/dlstat2xmldir/events";
$indexfile = "$xmldir/pipe_scenario.xml";
$catalogfile = "$xmldir/pipe_scenarioall.xml";

# shakemap directory for web
$shakeweb = "/usr/local/ShakeMapPipe/web/shake";

# ignore file
$ignorefile = "$xmldir/ignore_scenarios.xml";


################################################################################
################################

# INITIALISE FILES & DIRECTORIES

# Make sure the indexfile and ignorefile are blank, just with outer tags
&blank_indexfile();
&blank_ignorefile();
&update_catalog();

# Cancel any ShakeMaps for all scenarios
print "Cancelling ShakeMaps (if they exist)\n";
system("$smhome/bin/cancel -event scenario1_se");
system("$smhome/bin/cancel -event scenario2_se");

# Remove data directories
system("rm -r $smhome/data/scenario1_se");
```

```
system("rm -r $smhome/data/scenario2_se");

# Remove web directories
system("rm -r $shakeweb/scenario1_se");
system("rm -r $shakeweb/scenario2_se");
&run_scenario($evid);


# wait
print "Check that no data or web directories exist\n";
&wait_for_user();


################################################################################
################################

# DATA
my $Denali1   = "Denali_se   63.5141 -147.4529 7.3 4.2  20 n 1";
my $Coldfoot1 = "Coldfoot_se 67.2230 -149.8090 4.2 10.0 10 n 1";
my $PrinceWm1 = "PrinceWm_se 61.017  -147.648  4.1 10.0 10 n 1";
my $Tsaina1   = "Tsaina_se   61.267  -145.680  6.7 10.0 10 n 1";
my $Denali2   = "Denali_se   63.5131 -147.4539 7.4 5.2  30 n 2";
my $Salcha1   = "Salcha_se   64.610  -147.120  4.2 10.0 30 n 1";
my $Tsaina2   = "Tsaina_se   61.268  -145.681  6.8 11.0 10 n 2";
my $Tsaina3   = "Tsaina_se   61.267  -125.682  6.9 11.0 10 n 2";
my $PrinceWm2 = "PrinceWm_se 61.016  -147.647  4.2 10.0 10 n 2";
my $Denali3   = "Denali_se   63.5161 -147.4549 7.5 5.2  30 n 3";

# HERE ARE THE TESTS - THIS IS INTENDED TO MATCH JIMMY DE LA GARZA'S EMAIL OF
20080916

# Step 1 - Denali1, SMversion 0 - New event
my ($evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid) = split
/\s+/, $Denali1;
my ($year1, $month1, $day1, $hour1, $minute1, $second1) = gettime();
&new_event($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid,
0, $year1, $month1, $day1, $hour1, $minute1, $second1);
&wait_for_user();


# Step 2 - Coldfoot1, SMversion 0 - New event
my ($evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid) = split
/\s+/, $Coldfoot1;
my ($year2, $month2, $day2, $hour2, $minute2, $second2) = gettime();
&new_event($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid,
0, $year2, $month2, $day2, $hour2, $minute2, $second2);
&wait_for_user();


# Step 3 - Denali1, Generate SMversion 1
my ($evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid) = split
/\s+/, $Denali1;
&new_shakemap($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag,
$orid, 1, $year1, $month1, $day1, $hour1, $minute1, $second1);
&wait_for_user();
```

```
# Step 4 - PrinceWm1, SMversion 0 - New event
my ($evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid) = split
/\s+/, $PrinceWm1;
my ($year4, $month4, $day4, $hour4, $minute4, $second4) = gettime();
&new_event($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid,
0, $year4, $month4, $day4, $hour4, $minute4, $second4);
&wait_for_user();


# Step 5 - Tsaina1, SMversion 0 - New event
my ($evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid) = split
/\s+/, $Tsaina1;
my ($year5, $month5, $day5, $hour5, $minute5, $second5) = gettime();
&new_event($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid,
0, $year5, $month5, $day5, $hour5, $minute5, $second5);
&wait_for_user();


# Step 6 - Denali2, SMversion 1 - New origin (old ShakeMap)
my ($evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid) = split
/\s+/, $Denali2;
my ($year6, $month6, $day6, $hour6, $minute6, $second6) = gettime();
&new_origin($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag,
$orid, 1, $year6, $month6, $day6, $hour6, $minute6, $second6);
&wait_for_user();


# Step 7 - Coldfoot1 - event cancelled
my ($evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid) = split
/\s+/, $Coldfoot1;
&event_cancelled($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag,
$orid, 0, $year2, $month2, $day2, $hour2, $minute2, $second2);
&wait_for_user();


# Step 8 - Tsaina1, Generate SMversion 1
my ($evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid) = split
/\s+/, $Tsaina1;
&new_shakemap($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag,
$orid, 1, $year5, $month5, $day5, $hour5, $minute5, $second5);
&wait_for_user();


# Step 9 - Denali2, Generate SMversion 2
my ($evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid) = split
/\s+/, $Denali2;
&new_shakemap($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag,
$orid, 2, $year6, $month6, $day6, $hour6, $minute6, $second6);
&wait_for_user();


# Step 10 - simulates manually run ShakeMap without new origin, for example a
fault file
&new_shakemap($step, $evid, $lat, $lon, 7.7, $depth, $nass, $reviewflag,
$orid, 3, $year6, $month6, $day6, $hour6, $minute6, $second6);
&wait_for_user();
```

```
# Step 11 - Salcha1, SM version 0 - New event
my ($evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid) = split
/\s+/, $Salcha1;
my ($year11, $month11, $day11, $hour11, $minute11, $second11) = gettime();
&new_event($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid,
0, $year11, $month11, $day11, $hour11, $minute11, $second11);
&wait_for_user();


# Step 12 - Tsaina2, SM version 2 - New origin & ShakeMap
my ($evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid) = split
/\s+/, $Tsaina2;
my ($year12, $month12, $day12, $hour12, $minute12, $second12) = gettime();
&new_origin($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag,
$orid, 1, $year12, $month12, $day12, $hour12, $minute12, $second12);
&new_shakemap($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag,
$orid, 2, $year12, $month12, $day12, $hour12, $minute12, $second12);
&wait_for_user();


# Step 13 - PrinceWm2, SMversion 0 - New origin (old ShakeMap)
my ($evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid) = split
/\s+/, $PrinceWm2;
my ($year13, $month13, $day13, $hour13, $minute13, $second13) = gettime();
&new_origin($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag,
$orid, 0, $year13, $month13, $day13, $hour13, $minute13, $second13);
&wait_for_user();


# Step 14 - Tsaina3, SMversion 2 - New origin (old ShakeMap) OUT OF BOUNDS
my ($evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid) = split
/\s+/, $Tsaina3;
my ($year14, $month14, $day14, $hour14, $minute14, $second14) = gettime();
&origin_outofbounds($step, $evid, $lat, $lon, $mag, $depth, $nass,
$reviewflag, $orid, 2, $year14, $month14, $day14, $hour14, $minute14,
$second14);
&wait_for_user();


# Step 15 - Denali3, SMversion 3 - New origin (old ShakeMap)
my ($evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid) = split
/\s+/, $Denali3;
my ($year15, $month15, $day15, $hour15, $minute15, $second15) = gettime();
&new_origin($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag,
$orid, 3, $year15, $month15, $day15, $hour15, $minute15, $second15);
&wait_for_user();


# Step 16 - PrinceWm2 - event cancelled
my ($evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid) = split
/\s+/, $PrinceWm2;
&event_cancelled($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag,
$orid, 0, $year13, $month13, $day13, $hour13, $minute13, $second13);
&wait_for_user();
```

```
# Step 17 - Denali3, SMversion 4 - New ShakeMap
my ($evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid) = split
/\s+/, $Denali3;
&new_shakemap($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag,
$orid, 4, $year15, $month15, $day15, $hour15, $minute15, $second15);
&wait_for_user();


# Step 18 - Salcha1 - event cancelled
my ($evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid) = split
/\s+/, $Salcha1;
&event_cancelled($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag,
$orid, 0, $year13, $month13, $day13, $hour13, $minute13, $second13);
&wait_for_user();


# Step 19 - Denali3 - event cancelled
my ($evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid) = split
/\s+/, $Denali3;
&event_cancelled($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag,
$orid, 0, $year13, $month13, $day13, $hour13, $minute13, $second13);


##############################################################################
##########################################

# HERE ARE THE SUBROUTINES

sub new_event {
      my ($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid,
$version, $year, $month, $day, $hour, $minute, $second) = @_;
      if ($orid == 1 && $version == 0) {
            print "Step $step - $evid - origin $orid, SMversion $version:
This is a new event, should be origin 1\n";
            &index_new($evid, $lat, $lon, $mag, $year, $month, $day, $hour,
$minute, $second, $depth, $version, $nass, $reviewflag);
            &update_catalog();
      }
      else
      {
            "For a new event, orid should be 1, version should be 0\n";
      }
}

sub new_origin {
      my ($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid,
$version, $year, $month, $day, $hour, $minute, $second) = @_;
      print "Step $step - $evid - origin $orid, SMversion $version: This is a
new origin\n";
      &index_modify($evid, $lat, $lon, $mag, $year, $month, $day, $hour,
$minute, $second, $depth, $version, $nass, $reviewflag);
      &update_catalog();
}

sub new_shakemap {
      my ($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid,
$version, $year, $month, $day, $hour, $minute, $second) = @_;
```

```
      print "Step $step - $evid - origin $orid, SMversion $version: This is a
new ShakeMap\n";
      &create_input_directory($evid, $lat, $lon, $mag, $year, $month, $day,
$hour, $minute, $second, $depth)
      &run_scenario($evid);
      &index_modify($evid, $lat, $lon, $mag, $year, $month, $day, $hour,
$minute, $second, $depth, $version, $nass, $reviewflag);
      &update_catalog();
}

sub event_cancelled {
      my ($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid,
$version, $year, $month, $day, $hour, $minute, $second) = @_;
      print "Step $step - $evid - origin $orid, SMversion $version: Cancel
event\n";
      &ignore_event($evid, $year, $month, $day, $hour, $minute, $second);
      &cancel_scenario($evid);
      &index_remove($evid);
      &update_catalog();
}

sub origin_outofbounds {
      my ($step, $evid, $lat, $lon, $mag, $depth, $nass, $reviewflag, $orid,
$version, $year, $month, $day, $hour, $minute, $second) = @_;
      print "Step $step - $evid - origin $orid, SMversion $version: New
origin - out of bounds\n";
      &cancel_scenario($evid);
      &index_modify($evid, $lat, $lon, $mag, $year, $month, $day, $hour,
$minute, $second, $depth, $version, $nass, $reviewflag);
      &update_catalog();
      &index_remove($evid);
}



sub create_input_directory {
      use File::Path qw(mkpath);
      my ($evid, $lat, $lon, $mag, $year, $month, $day, $hour, $minute,
$second, $depth) = @_;
      my $locstring = $evid;

      my $creation_time = substr(`epoch $year-$month-$day `,1,12);
      #$creation_time = epoch("$year-$month-$day $hour:$minute:second");
      my $INPUTDIR = "/usr/local/ShakeMapPipe/data/$evid/input";
      print "Creating input directory $INPUTDIR\n";

      my $eventxml = <<"ENDOFEVENTXML"  ;
<?xml version="1.0" encoding="US-ASCII" standalone="yes"?>
<!DOCTYPE earthquake [
<!ELEMENT  earthquake EMPTY>
<!ATTLIST earthquake
  id            ID      #REQUIRED
  lat           CDATA   #REQUIRED
  lon           CDATA   #REQUIRED
  mag           CDATA   #REQUIRED
  year          CDATA   #REQUIRED
  month         CDATA   #REQUIRED
```

```
  day           CDATA   #REQUIRED
  hour          CDATA   #REQUIRED
  minute        CDATA   #REQUIRED
  second        CDATA   #REQUIRED
  timezone      CDATA   #REQUIRED
  depth         CDATA   #REQUIRED
  locstring     CDATA   #REQUIRED
  pga           CDATA   #REQUIRED
  pgv           CDATA   #REQUIRED
  sp03          CDATA   #REQUIRED
  sp10          CDATA   #REQUIRED
  sp30          CDATA   #REQUIRED
  created       CDATA   #REQUIRED
>
]>
<earthquake id="$evid" lat="$lat" lon="$lon" mag="$mag" year="$year"
month="$month" day="$day" hour="$hour" minute="$minute" second="$second"
timezone="GMT" depth="$depth" locstring="$locstring" created="$creation_time"
/>
ENDOFEVENTXML

      mkpath($INPUTDIR);
      print "$INPUTDIR/event.xml\n";
      open(fout,">$INPUTDIR/event.xml") or die("Cannot open lsevent.xml");
      print fout "$eventxml\n" ;

      close(fout);

      system("cp /usr/local/ShakeMapPipe/data/Tsaina_se/input/db_dat.xml
$INPUTDIR");
#     system("cp /usr/local/ShakeMapPipe/data/Tsaina_se/input/estimates.xml
$INPUTDIR");
}



sub wait_for_user {
        sleep(300);
      #print "<Return> to continue\n";
      #$temp =<STDIN>;
}


sub index_new {
      my ($evid, $lat, $lon, $mag, $year, $month, $day, $hour, $minute,
$second, $depth, $smflag, $nass, $reviewflag) = @_;
      my $indexxml = <<"ENDOFINDEXXML"  ;
   <event>
      <event_id>$evid</event_id>
      <origin_time>$month/$day/$year  $hour:$minute:$second</origin_time>
      <latitude>$lat</latitude>
      <longitude>$lon</longitude>
      <magnitude>$mag</magnitude>
      <shakemap>$smflag</shakemap>
      <arrivals>$nass</arrivals>
      <review>$reviewflag</review>
      <timestamp>$month/$day/$year  $hour:$minute:$second</timestamp>
```

```
   </event>
ENDOFINDEXXML

      my $tmpindexfile = $indexfile .".tmp";
      open(IND, "$indexfile") or die "$!";
      open(TMP, "> $tmpindexfile") or die "$!";
      my $line;
      while (defined($line = <IND>)) {
            chomp $line;
            if ($line =~ /<\/quakes>/) {
                  print TMP $indexxml;
                  print $indexxml;
            }
            print TMP "$line\n";
            print "$line\n";
      }
      close(TMP) or die "$!";
      close(IND) or die "$!";
      rename("$tmpindexfile", "$indexfile") or die "$!";
      return 1;
}

sub index_modify {
      # same as index_new except we stop printing of old record when we
insert new
      my ($evid, $lat, $lon, $mag, $year, $month, $day, $hour, $minute,
$second, $depth, $smflag, $nass, $reviewflag) = @_;
      my $indexxml = <<"ENDOFINDEXXML"  ;
      <event_id>$evid</event_id>
      <origin_time>$month/$day/$year  $hour:$minute:$second</origin_time>
      <latitude>$lat</latitude>
      <longitude>$lon</longitude>
      <magnitude>$mag</magnitude>
      <shakemap>$smflag</shakemap>
      <arrivals>$nass</arrivals>
      <review>$reviewflag</review>
      <timestamp>$month/$day/$year  $hour:$minute:$second</timestamp>
ENDOFINDEXXML
      my $thisrecord = 0;
      my $tmpindexfile = $indexfile .".tmp";
      open(IND, "$indexfile") or die "$!";
      open(TMP, "> $tmpindexfile") or die "$!";
      my $line;
      while (defined($line = <IND>)) {
            chomp $line;
            if ($line =~ /$evid/) {
                  print TMP $indexxml;
                  print $indexxml;
                  $thisrecord = 1; # turn printing off, so we miss old record
            }
            $thisrecord = 0 if ($line =~ /<\/event>/); # turns printing on
again
            unless ($thisrecord == 1) {
                  print TMP "$line\n";
                  print "$line\n";
            }
```

```
      }
      close(TMP) or die "$!";
      close(IND) or die "$!";
      rename("$tmpindexfile", "$indexfile") or die "$!";
      return 1;
}

sub index_remove {
      # same as index_modify, except there is nothing to insert when skipping
old record
      my ($evid) = $_[0];
      my $thisrecord = 0;
      my $tmpindexfile = $indexfile .".tmp";
      open(IND, "$indexfile") or die "$!";
      open(TMP, "> $tmpindexfile") or die "$!";
      my $line;
      while (defined($line = <IND>)) {
            chomp $line;
            if ($line =~ /$evid/) {
                  $thisrecord = 1; # turn printing off, so we miss old record
            }

            unless ($thisrecord == 1) {
                  unless ($line =~ /<event>/) { # modifications necessary so
we don't leave an empty <event> tag.
                        if ($line =~ /<event_id>/) {
                              print TMP "   <event>\n";
                              print "   <event>\n";
                        }
                        print TMP "$line\n";
                        print "$line\n";
                  }
            }
            $thisrecord = 0 if ($line =~ /<\/event>/); # turns printing on
again


      }
      close(TMP) or die "$!";
      close(IND) or die "$!";
      rename("$tmpindexfile", "$indexfile") or die "$!";
      return 1;

}

sub ignore_event {
      # This is like index_new, except we're working with a different file,
      # matching on </ignore_quakes> rather than </quakes> and inserting
different XML
      my ($evid, $year, $month, $day, $hour, $minute, $second) = @_;
      my $ignorexml = <<"ENDOFIGNOREXML"  ;
   <event>
      <event_id>$evid</event_id>
      <timestamp>$month/$day/$year  $hour:$minute:$second</timestamp>
   </event>
ENDOFIGNOREXML
```

```
	my $tmpignorefile = $ignorefile .".tmp";
	open(IGN, "$ignorefile") or die "$!";
	open(TMP, "> $tmpignorefile") or die "$!";
	my $line;
	while (defined($line = <IGN>)) {
		chomp $line;
		if ($line =~ /<\/ignore_quakes>/) {
			print TMP $ignorexml;
			print $ignorexml;
		}
		print TMP "$line\n";
		print "$line\n";
	}
	close(TMP) or die "$!";
	close(IGN) or die "$!";
	rename("$tmpignorefile", "$ignorefile") or die "$!";
	return 1;

}

sub blank_indexfile {
	system("cp $testdir/pipe_scenario_blank.xml $indexfile");
}

sub blank_ignorefile {
	system("cp $testdir/ignore_scenarios_blank.xml $ignorefile");
}

sub gettime {
	my ($second, $minute, $hour, $day, $month, $year) = gmtime();
	$month++;
	$second=makeNdigits($second,2);
	$minute=makeNdigits($minute,2);
	$hour=makeNdigits($hour,2);
	$day=makeNdigits($day,2);
	$month=makeNdigits($month,2);
	$year += 1900;
	return ($year, $month, $day, $hour, $minute, $second);
}

sub makeNdigits {
	my ($var, $l) = @_;
	while (length($var) < $l) {
		$var = "0" . $var;
	}
	return $var;
}

sub run_scenario {
	my $evid = $_[0];
	print "Running ShakeMap for $evid\n";

	system("cp /usr/local/ShakeMapPipe/config/shake.conf.scenario
/usr/local/ShakeMapPipe/config/shake.conf");
	system("$smhome/bin/shake -event $evid");
	system("cp /usr/local/ShakeMapPipe/config/shake.conf.real
/usr/local/ShakeMapPipe/config/shake.conf");
```

```
      if (-e "$shakeweb/$evid") {
            print "ShakeMap is now available\n";
      }
      else
      {
            print "Oops - no ShakeMap\n";
      }

}

sub cancel_scenario {
      my $evid = $_[0];
      print "Cancelling ShakeMap for $evid\n";

      system("cp /usr/local/ShakeMapPipe/config/shake.conf.scenario
/usr/local/ShakeMapPipe/config/shake.conf");
      system("$smhome/bin/cancel -event $evid");
      system("cp /usr/local/ShakeMapPipe/config/shake.conf.real
/usr/local/ShakeMapPipe/config/shake.conf");

      if (-e "$shakeweb/$evid") {
            print "Oops - ShakeMap still available\n";
      }
      else
      {
            print "ShakeMap cancelled\n";
      }

}

sub update_catalog {
      system("cp $indexfile $catalogfile");
}
```