AmericanLifelinesAlliance

A public-private partnership to reduce risk to utility and transportation systems from natural hazards and manmade threats

U.S. Geological Survey's ShakeMap and ShakeCast: Improving Utilization within the American Lifelines Alliance (ALA) Community

September 30, 2004

FEMA

National Institute of BUILDING SCIENCES

AmericanLifelinesAlliance

A public-private partnership to reduce risk to utility and transportation systems from natural hazards and manmade threats

U.S. Geological Survey's ShakeMap & ShakeCast: Improving Utilization within the American Lifelines Alliance (ALA) Community

September, 2004

www.americanlifelinesalliance.org

This report was written under contract to the American Lifelines Alliance, a public-private partnership between the Federal Emergency Management Agency (FEMA) and the National Institute of Building Sciences (NIBS). This report was reviewed by a team representing practicing engineers, academics and federal scientists

Acknowledgements

The following people (with their affiliations) contributed to this report.

David Wald, Principle Investigator U.S. Geological Survey, Golden, CO

Table of Contents

List of Figures

1.0 Introduction

The Federal Emergency Management Agency (FEMA) formed in 1998 the American Lifelines Alliance (ALA) as a public-private partnership. In 2002, FEMA contracted with NIBS through its Multihazard Mitigation Council (MMC) to, among other things, assist FEMA in continuing ALA earlier guideline development efforts. In 2004, ALA requested the U.S. Geological Survey to further develop and promote ShakeCast. ShakeCast, short for ShakeMap Broadcast, is a fully-automated system for delivering specific ShakeMap products to critical users and triggering established post-earthquake response protocols.

1.1 Background

ShakeMap is a tool used to portray the extent of potentially damaging shaking following an earthquake. It can be found on the Internet at http://earthquake.usgs.gov/shakemap and is automatically generated for both small and large earthquakes in areas where it is available. It can be used for emergency response, loss estimation, and public information.

Despite the popularity and acclaim of ShakeMap for emergency response and post-earthquake information, there is a lack of recognition of the full potential of ShakeMap. That is, critical users need to move beyond simply "looking at ShakeMap," and begin implementing response protocols that utilize the known shaking distribution in fully automated systems in order to more fully realize this potential to prioritize and greatly aid response efforts. To this end the USGS has begun the development of ShakeCast.

ShakeCast is a fully-automated system for delivering specific ShakeMap products to critical users and triggering established post-earthquake response protocols. ShakeCast allows utilities, transportation agencies, and other large organizations to automatically determine the shaking values at their facilities, set thresholds for notification of damage states (typically green, yellow, red) for each facility and then automatically notify (via pager, cell phone, email) specified operators, inspectors, etc., within their organizations who are responsible for those particular facilities so they can prioritize their responses.

For example, the California Department of Transportation (Caltrans) is testing the prototype ShakeCast system right now. Caltrans has over 25,000 bridges and overpasses under their responsibility in California; having an instantaneous snapshot of the likely damage to each will allow them to prioritize rerouting traffic, closures, and inspections following a damaging earthquake. They are pretty excited about this technology, and many critical users who have heard about it are excited to get it soon. However, we are limited in the resources we can devote to the development of ShakeCast, slowing progress, and the groundwork for establishing relationships with many other potential users is beyond our current effort and capacity. While some lifeline organizations could use these systems directly, others need further assistance as is outlined bellow.

In addition to real-time notification, an additional major benefit of the ShakeMap/ShakeCast combination is its built-in capacity to generate and deliver Scenario Earthquakes for evaluating system performance and response capabilities under a variety of earthquake conditions. ShakeMap is now used routinely to generate earthquake scenarios for many users; ShakeCast

will further allow these system tests to be performed with the same notification tools that will be in place and available for a responding to a real earthquake.

As an important side note, the delivery of critical post-earthquake information is a difficult problem and is serious business. In the ShakeCast development, reliable and redundant transmission, security and authentication, and documentation and version control of product delivery have all been considered from the onset. However, is quite clear that ongoing IT security concerns and changes within organizations will present ongoing development hurdles. These will be addressed as necessary through direct user interfacing in our prototype installations. We expect the solutions we develop through these initial installations to be robust and therefore can be used for wider distribution of ShakeCast to other users. These tools will also be beneficial for other rapid post-disaster notification and alerting systems.

1.2 Project Objective and Scope

The ShakeMap developers have long recognized the lifeline organizations as one primary class of users that could directly benefit the most from having rapid information in the aftermath of a significant earthquake. Through long-term interactions in southern California, many lifeline organizations and companies have had the necessary interaction to further the use of ShakeMap in post-earthquake response. However, a more central, nationwide distribution of information about ShakeMap/ShakeCast was deemed necessary to reach the wide array of potential users. We believe that many of the ALA partners and the intended users of ALA products will find these systems highly beneficial if they were fully informed of the potential applications. In this sense ALA via this ShakeCast funding will facilitate providing guidelines and the information necessary to reach this wider audience.

A second critical role that ALA is playing is in facilitating the development of fragility relationships necessary to utilize shaking information in real-time. That is, each utility and transportation organization has an array of facility and component types with varying degrees of knowledge of their individual vulnerability, and highly variable levels of in-house expertise needed to establish and properly use fragility relationships to assign likely damage states in real time. A systematic approach to this development, and formal guidelines on the use of such facility/fragility assignments would be more cost-effective than relying on individual organizations to develop their own strategies (a task many have not or may not otherwise take on). ShakeMap thus is a specific analysis tool for using standardized procedures for evaluating lifeline earthquake damage and functionality, and will provide added value to ALA products and the ALA mission.

Based on these objectives, the project scope consisted of the following tasks. Funding was used for contract software developers working under USGS direction. All efforts were guided and supplemented by USGS staff.

1) Expedite development of the second and final phase of ShakeCast that will result in a well-documented, self-installing software package for use by utility and transportation systems and others.

2) Develop lessons learned from interfacing with at least 3 major users (possibly among WashDOT, Bureau of Reclamation, California DWR, Seattle City Light, others) that have different ShakeMap/ShakeCast applications than Caltrans and PG&E (including automated post-processing for loss-estimation), and integrate resulting lessons learned into initial user ShakeMap/ShakeCast guidelines by which current and future fragility analyses can take advantage of the rapid post-earthquake information provided by ShakeMap and ShakeCast considering the specific parametric values ShakeMap produces, mainly peak ground motion and spectral accelerations.

3) Initial development of default ShakeMap-specific facility fragility, damage, and notification tables based on the prototype user's system assignments. In consultation with NIBS and ALA, begin collection and assignment of additional default fragility, damage, and notification levels for other users according to established guidelines (ALA Lifeline Guidelines, HAZUS, ATC, etc.). Consistent with currently available ShakeMap parameters, these fragility assignments will be limited to dynamic (that is, seismic wave propagation-related) and not permanent ground deformation-related losses. Ongoing and additional studies will provide additional fragility values suitable for post-earthquake response and will continue expansion of a fragility matrix from which other users can select appropriate values.

4) Promote ShakeCast within the lifelines arena by producing a USGS Fact Sheet, an ATC Tech Brief, and/or other joint outreach recommendations through consultation and collaboration between NIBS, ALA and Contractor.

In addition to current USGS funding for ShakeCast at the level of $100,000, USGS provided no cost support for this project in the form of salary support for D. Wald and B. Worden for ShakeMap development and ShakeCast oversight and continued user interaction and assistance. Also, no-cost efforts have been provided by prototype users at Caltrans (mainly Loren Turner) and at PG&E (Marcia Mclaren and others).

The following deliverables have been provided under this contract:

1. A well-documented, self-installing software package by expediting development of the first release of the ShakeCast system.

2. Initial user guidelines that are the result of lessons learned from interfacing with at least 3 major users.

3. Default ShakeMap-specific facility fragility, damage, and notification tables based on the prototype user's system assignments.

4. USGS Fact Sheet, an ATC Tech Brief, and/or other joint outreach recommendations.

Early on in the project cycle, the following timeline was established, and monthly progress reports were provide to ALA.

2004 ALA ShakeCast Project - Time Line

Tasks	March	April	May	June	July	Aug.	Sept
1) ShakeCast (SC) Development							
Design & Implement Phase II SC System							
Implement Basic Istaller (Windows & UNIX)							
Notification & Messaging							
Templates							
Aggregation within message type							
Format for Summary Messages							
User Configuration							
System Sign-Up							
Database Interface							
User Documentation (Install, Admin, User)							
Server Installation upstream Signup System							
Address missing features, attributes							
2) User Support and Interfacing							
SC Install at Two Critical Lifeline Users							
SC Install at Two Additional Lifeline Users							
3) Develop Initial Default Facility Fragility							
Literature/Expert search for existing guidelines							
Assign Fragilities based on ShakeMap Parameters							
Implement Assignment as Default Option in SC							
4) ShakeCast Promotion							
ALA/USGS Discussion on Promotion Ideas							
Design USGS Fact Sheet or Info Sheet							
Design ATC Tech Brief (with S. King)							

2.0 Project Results

2.1 ShakeCast Development

A well documented, self-installing ShakeCast software package was developed. This package is now available to users (at no charge) via download at the official ShakeCast web page (http://www.shakecast.org/). This site has complete ShakeCast documentation, install scripts (Windows and Unix), user scripts for database loading, and an installation guide.

The home page for ShakeCast is shown in Figure 2.1. ShakeCast System software can be download from the "Download" page (Figure 2.2.) for either UNIX systems or Windows systems. UNIX has been developed for FreeBSD UNIX, but one of the users (PG&E) was able to operate the system easily on Sun's Solaris UNIX. The Windows version is operational in Windows 2000 and XP.

2.2 Lessons Learned from User Interfacing—Documentation and Support

A number of iterations between the development team and two users, Caltrans and PG&E, led to substantial changes in ShakeCast's software and documentation. Early-on discussions pointed to the need for both cell/pager summary notifications and complete summary information via email. It also became clear that we needed to "aggregate" notifications to avoid numerous notifications if individual facilities exceed a predefined notification threshold. This was accomplished by combining multiple notifications into one message for delivery.

The outcome was a system that sorts through a user's facilities, summarizes the likely impact on each facility, and sends a brief summary (short message) to a text pager/Cell service and the complete summary (list of facilities) via email. Further discussions led to 1) additional sorting capabilities on the summary information and 2) a common-separated value (CSV) email attachment that can be imported directly into Excel for further sorting or analysis.

Users also requested simplification in the interface to all databases, resulting in a scripted facility import tool (see below), additional functionality for added notifications, including allowing "groups" for similar facility and notification classes that could be easily grouped for simplifying notifications. A number of simplifications were also made to the ShakeCast Server Configuration approach as well as the user documentation.

Most important, the users installed the system based on the installation instructions. Numerous cases of unclear or confusing wording were remedied and additional suggestions for clarifying particular options were provided, resulting in a more intuitive installation and user guidelines.

User Documentation

Initial user guidelines were produced, put online, and provided with the initial ShakeCast software download. These initial user guidelines were then iteratively modified based on our test

users' experience with installation, internet security access issues, and setting up facility, fragility, and notification databases.

The ShakeCast System Specification is available online in the Documentation section of the ShakeCast web page and is included in Appendix A.4 of this report. Figure 2.3 shows the main Documentation portion of the ShakeCast web pages. Important sections included in the online documentation are outline below. However, as is customary, the documentation is meant to be interactive and interlinked, and thus is it not included in its entirety.

Installation
- UNIX
- Initial Installation
- Upgrading
- Windows

Configuration
- Configuring the ShakeCast Server
- Facility Importing: facimport
- The Configuration File
- Aggregation of Notifications
- Notification Templates

Another feature of the ShakeCast web site is the "What's New" section (see Figure 2.4), which provides a development timeline, version control, and specifications for each new release of the ShakeCast software.

In addition to the extensive User Documentation online, a separate section of the documentation exists called "Internal Documentation". This section is provided for those interested in developing add-on capabilities and features for ShakeCast, as is anticipated in the open-source environment. Several users have indicated the desire for additional functionality and it is anticipated that such efforts will be contributed back to the ShakeCast software for redistribution to other potential users. For example, Caltrans has proposed to add a more interactive Graphical User Interface (GUI) for interacting with the user databases, including notifications with internal funding.

User Support and Interfacing

The following ShakeCast users were engaged during this project. Below we describe the nature of their use, feedback and plans to use ShakeCast.

Pacific Gas & Electric (PG&E). Contact: Marcia McLaren

Initially, PG&E was interested in ShakeCast for the Electric group, using primarily the download and script start-up option to get ShakeMap reliably into their Map Server internal response system. Subsequently, the Gas (Pipelines) group has also installed ShakeCast to help address their pipelines Integrity Management System as required by US Department of Transportation. The Gas group is utilizing the full functionality of ShakeCast, getting notifications for each

pipeline in their inventory. Also, Stuart Nishenko has been expanding the scope of this effort at PG&E to include the PEER Lifelines Program in an effort to develop guidelines for Lifeline Utility and Transportation using ShakeMap and ShakeCast. They anticipate California Energy Commission funding. The use of ShakeCast will be expanded to PG&E's Nuclear, Hydro generation, and building and Land, Emergency Operation Centers.

PG&E's Gas Pipeline group installed the UNIX version under a new platform (SUN Solaris) for the first time, with little difficulty, and their feedback on this installation was incorporated in to the user documentation. PG&E also installed dual hardware systems external to their firewall system, communicating with two systems internally, which required additional documentation in the users' guide.

California Department of Transportation (Caltrans). Contact: Loren Turner

Caltrans is responsible for over 25,000 bridges and overpasses in California. The bridge engineering group is using ShakeCast for rapid evaluation of shaking at each, as facilities and fragilities are loaded in their ShakeCast database. The success of the use of this system is allowing Caltrans to expand their internal usage to other critical parts of the organization, including Traffic Management and Bridge inspection. Significant and useful suggestions were provided by Loren Turner (Senior Transportation Engineer) concerning several aspects of the ShakeCast system. Caltrans was instrumental in switching from a web-based to a scripted inventory database loading procedure, which makes sense given the number of structures that they are concerned with. Caltrans had trouble with SMTP email notifications in house and the contractor worked with them to resolve this, and documented the correct approach to avoid this difficulty for future users. Caltrans also had suggestions for documentation of the event notification procedures and event types, which involve actual, scenario, test, and heartbeat event types and wanted better documentation for testing the system. They also wanted screen shots of all configuration procedures included in the user documentation. These suggestions were folded into the software and user documentation.

Due to Caltrans' favorable review and desire for ShakeCast, they plans to support further enhancements of the ShakeCast system, including 1) an improvement to the Graphical User Interface for web-based access and modification to the user, notification, facility, and fragilities databases within ShakeCast, and 2) web-based tools for incorporating ShakeCast inventory status into an in-house distributed (shared) mapping system, perhaps ArcGIS or MapServer.

California Department of Water, Division of Dam Safety (DWR). Contact: William Frazier

DWR has installed ShakeCast to dispatch inspectors to over 1,200 dams statewide. At the time of this report, DWR was receiving ShakeCast notifications, but the extent of their use of the facility and notification subsystems was not known. They had little trouble with the ShakeCast installation, suggesting improvements made by interactions with the above users aided significantly.

Washington State Department of Transportation (WashDOT). Contact: Steve Malone

WashDOT will receive ShakeCast notifications from the University of Washington based ShakeMap system as soon as ShakeMap updates are accomplished there. WashDOT will be working with UW Civil Engineer Marc Eberhard, who has established appropriate fragilities based on Nisqually earthquake damage assessments. Dr. Eberhard has provided us with a very useful way of prioritizing the facility listing: by threshold exceedence. Hence, in addition to the ability to rank facilities in order of shaking level, one can sort them by the ratio of the shaking metric (PGA, PGV, etc) to the threshold of exceedence. Such a listing provides a more logical ranking of the structures to most likely have been damage. These changes have been added to the notification portion of the documentation

Federal Emergency (FEMA). Contact: Douglas Bausch (Region VIII)

FEMA has installed ShakeCast in Denver with plans to import ShakeMaps from all US regions producing these maps, and automating initiation of HAZUS-MH loss estimates. Though installed, delays in implementation have resulted due to Doug's assignments to respond to recent hurricanes in Florida. However, feedback from Doug Bausch on the use of web proxy servers was important since this type of firewall application is widespread and has now been folded into the ShakeCast documentation.

Additional Contacts

Initial contacts and interest have been made with SBC Communications, Nextel Communications, Los Angeles County Department of Water and Power (LADWP), and the California Earthquake Authority.

2.3 Development of Initial Default Facility and Fragility Databases

Recall that the approach for keeping a facility and fragility database in ShakeCast is the responsibility of the user, not the USGS. This approach was chosen due to a number of reasons, including maintaining confidentiality of facilities and their locations, and ever-changing databases that the USGS would not be able to keep current. Likewise, users need to specify the fragility of each facility in their database according to reported levels of the basic ShakeMap parameters (instrumental intensity, peak acceleration, peak velocity, or one of three spectral acceleration periods). Part of the effort with the ALA funded development went toward providing examples of how these databases should be derived and populated.

The initial plan for default user databases was to have intact database (MySQL) tables for example use. Discussions with users and the contractor resulted in a more logical approach. We now have example text files (comma delimited) in the ShakeCast software download that are easily readable and understandable. Then, a single-command processing script ("facimport", for facility import) was developed that propagates that "user friendly" format into the MySQL database as needed. In this way, users can easily edit the text version of their databases, and then simply update their relational database with the script.

Clear documentation for this script is provided in the user documentation. In addition, four

separate examples are provided under the Facility Import documentation, capturing the expected range of facility and notification combinations. These examples are 1) Point Facilities, 2) Fragility parameters, 3) Facility attributes, and 4) Multiple Attributes and Multiple Metrics.

ShakeCast can generate Damage Notification messages based on damage to specific facilities. These messages are created and sent after the user's ShakeCast Server has received most of the ShakeMap Grid data from the upstream ShakeCast Servers. This can occur up to several minutes after an earthquake event. The Damage Notification is used to alert specific people in the organization who have responsibility for specific facilities or groups of facilities.

In order for ShakeCast to be able to estimate shaking and the potential for damage to a user's facilities, the facility date need to be loaded into the system. One way to do that is using the Facility Administration Tool. This method is appropriate for defining small numbers of facilities, such as for testing or evaluating ShakeCast. Larger numbers of facilities should be loaded using the Facility Import Tool or directly into the ShakeCast MySQL database.

For each facility, low and high limits are chosen, along with the appropriate metric, one of the ShakeMap parameters (Instrumental Intensity, peak acceleration, peak velocity or one of three spectral acceleration periods). A text description of the three Damage Levels is associated with the three possible metric ranges, that is, below the low limit, between the low and high limits, and above the high limit. Typically these are Damage Unlikely, Damage Possible, and Damage Probable.

Multiple metrics for a facility and for notifications are allowed with ShakeCast. For example, a facility could be considered below the high limit for spectral acceleration at 1 sec period, but above the high limit for the 0.3 sec period spectral acceleration. In such a case, and depending on the way the user sets up notifications, the recipients would be notified of the exceedence for the latter metric.

Currently, the metrics for fragility assignments are limited to one or more of the ShakeMap metrics. It is conceivable that this approach could be expanded in the future to apply more complex functions of these metrics with computations within the MySQL database. It is anticipated that new, sophisticated users will desire such assignments and expand this capacity in the open-source environment in which ShakeCast has been developed.

2.4 ShakeCast Promotion

Part of the promotion intent was to produce general information for non-technical users to be come familiar with the ShakeCast system and its possibly uses. More technical guidelines were provided on the ShakeCast web site and with documentation accompanying the ShakeCast software download.

To this end, a ShakeCast Information Sheet (Appendix A.1) was produced and made available online (http://www.shakecast.org/). This Information Sheet is in the process of being transformed into a USGS Fact Sheet and an Applied Technology Council (ATC) Technical

Briefing. The publication of these, however, are at the mercy of resources, schedules, and printing capacity of these two agencies. We have a Draft version of the ATC Tech Brief, awaiting contributions from Caltrans and PG&E as user Case Histories.

ShakeCast promotion also included talks at National conferences, including Disaster Resistant California in Sacramento, CA, a NATO meeting on Strong Motion Seismology in Turkey, and the National Earthquake Conference in Saint Louis, MO. (Appendices A.1, A.2, and A.3). In addition, David Wald (USGS), Bruce Worden (USGS) and/or Phil Naecker (Gatekeeper Systems, Inc.) have been making onsite presentations to PG&E, Caltrans, Caltech/USGS CUBE Users (Earthquake Affiliates Program), Los Angeles County Department of Water and Power, and Nextel Communications.

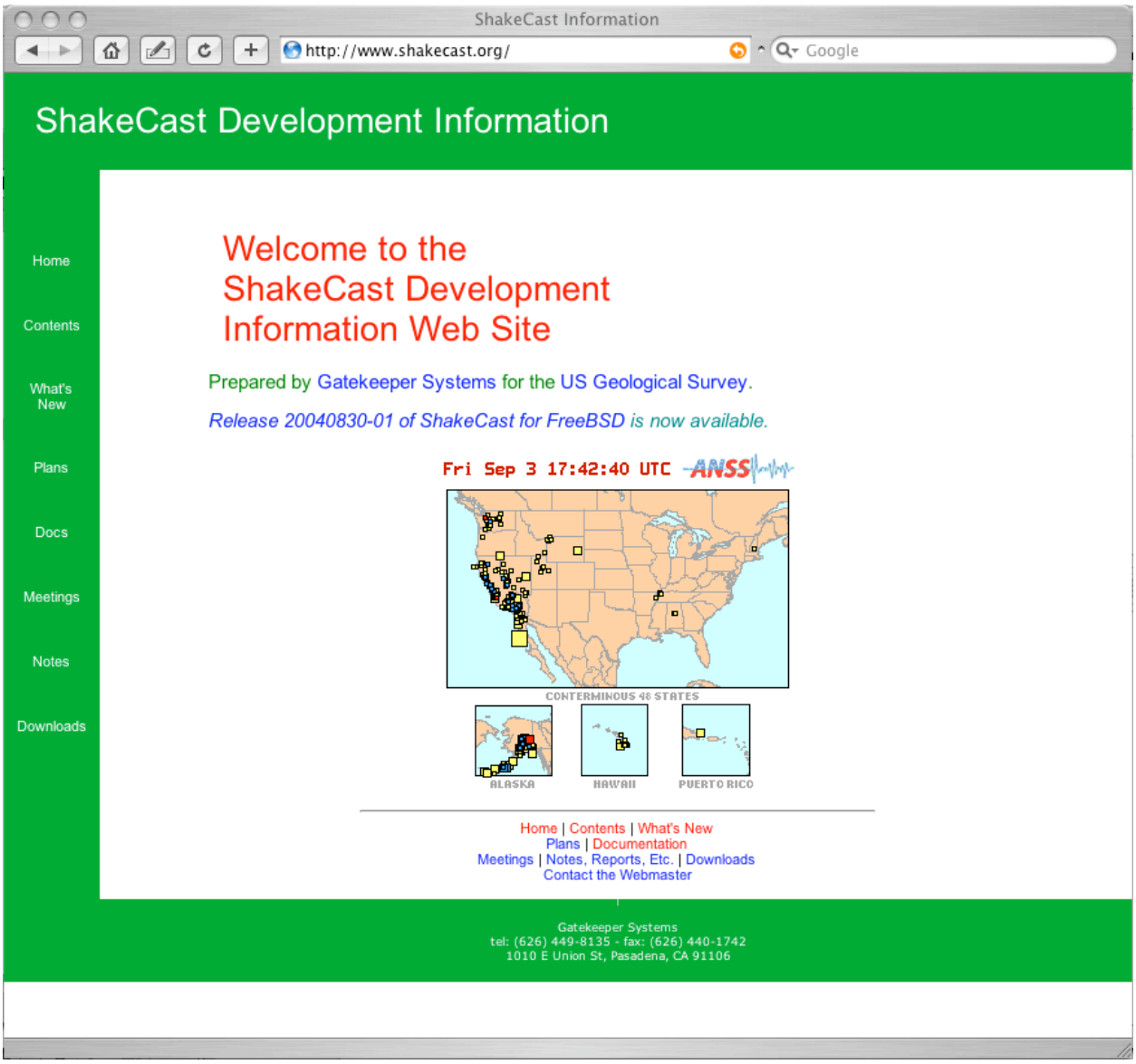


Figure 2.1 ShakeCast Web Home Page

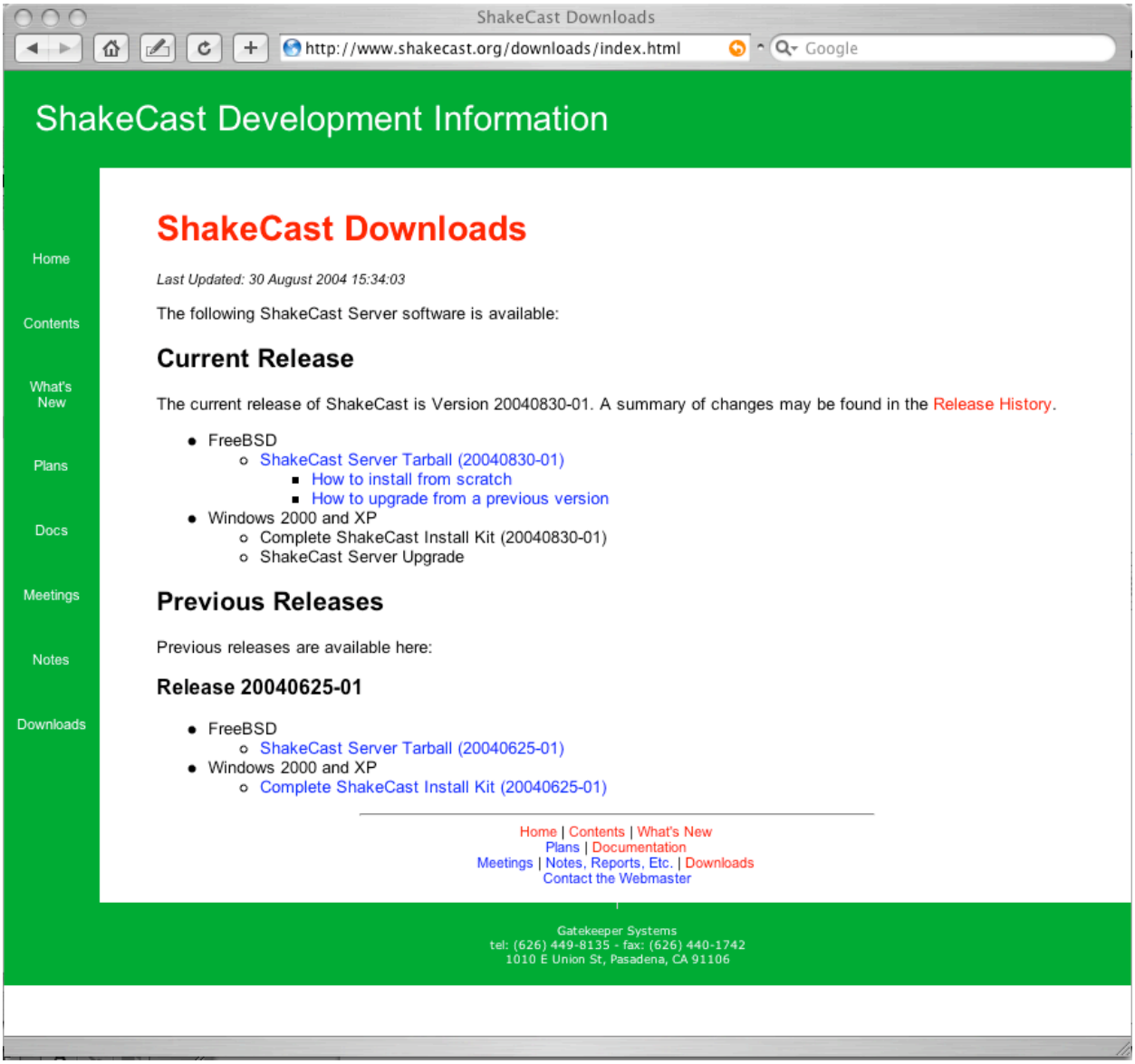


Figure 2.2 ShakeCast Web Page (Downloads)

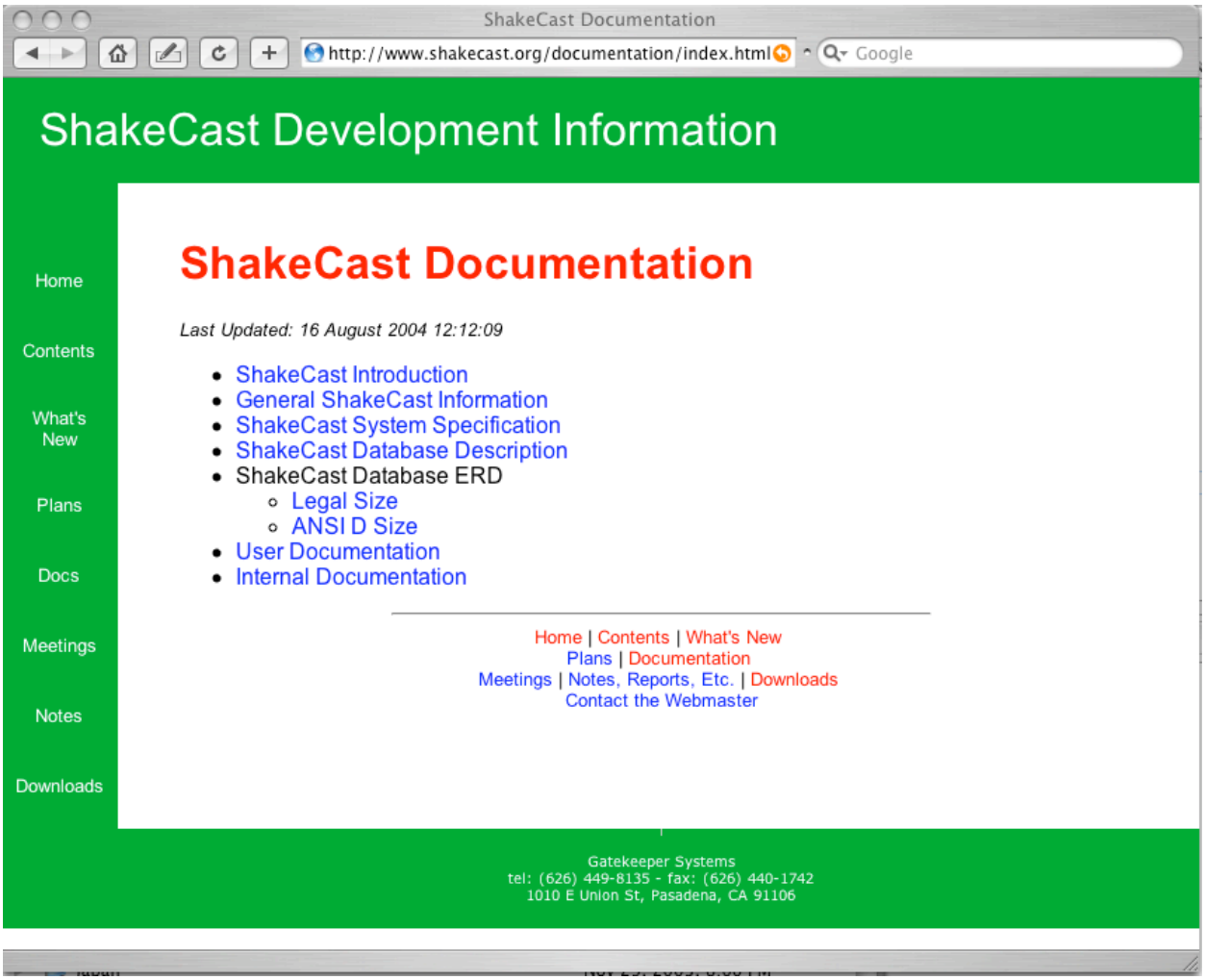


Figure 2.3 ShakeCast Web Page (Documentation)

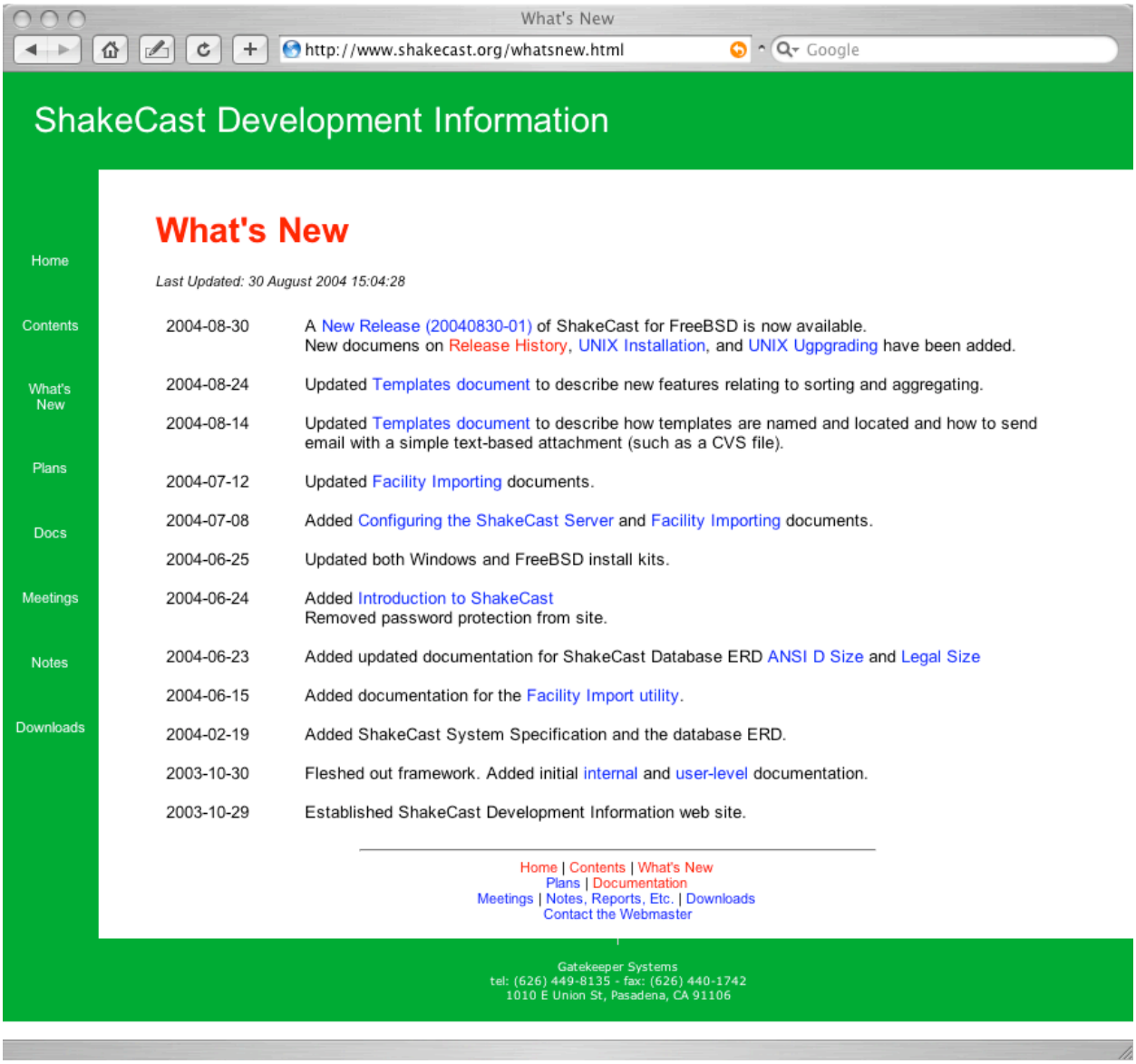


Figure 2.4 ShakeCast Web Page (What's New)

References

Applied Technology Council (2003). *ATC-54:* Guidelines for Using Strong Motion Data and ShakeMaps in Post-Earthquake Responses, published by Calif. Geological Survey & Applied Technology Council. Available online at http://www.atccouncil.org/.

Development of a ShakeMap-based, earthquake response system within Caltrans, 2003, D. Wald, P. Naecker, C. Roblee, and L. Turner, in *Advancing Mitigation Technologies and Disaster Response for Lifeline Systems*, J. Beavers, Ed., Technical Council on Lifeline Earthquake Engineering, Monograph No. 25, August 2003, ASCE.

ShakeMap—A Tool for Earthquake Response, 2003. , D. Wald, L.Wald, B. Worden, and J. Goltz, *U.S. Geological Survey Fact Sheet 087-03*, found online at http://pubs.usgs.gov/fs/fs-087-0.

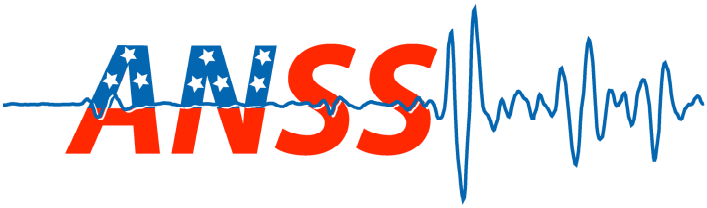
Wald, D. J. B. Worden, H. Kanamori, T. H. Heaton, and V. Quitoriano (1999). TriNet "ShakeMaps": Rapid Generation of Peak Ground Motion and Intensity Maps for Earthquakes in Southern California, 1999. *Earthquake Spectra*, Vol. 15, No. 3, 537-556.

Wald, D. J., B. Worden, V. Quitoriano, and K. Pankow, (2004). ShakeMap Manual: Technical Manual, Users Guide, And Software Guide, U.S. Geological Survey Open-File Report, in Review.

Appendix: Supplementary Documents

A number of supporting documents were produced not for final the ALA report, but rather for actual user installation, operations, and use of ShakeCast, as well as for ShakeCast outreach to the user community and technical documentation for programmers. These documents are provided here separately in Appendix subsections.

A.1 ShakeCast Information Sheet



ShakeCast Information Sheet

ShakeCast: Automating, Simplifying, and Improving the Use of ShakeMap for Rapid, Critical, Post-Earthquake Decision-Making and Response

When a potentially damaging earthquake occurs, utility and other lifeline managers, emergency responders, and other critical users have an urgent need for information about the impact on their facilities so they can make appropriate decisions and take quick actions to ensure safety and restore system functionality. ShakeCast, short for ShakeMap Broadcast, is a fully automated system for delivering specific ANSS ShakeMap products to critical users and triggering established post-earthquake response protocols. ShakeCast allows utilities, transportation agencies, and other large organizations to automatically determine the shaking value at their facilities, set thresholds for notification of damage states (typically, damage unlikely, moderate, or serious) for each facility, and then automatically notify (via pager, cell phone, or email) specified operators, inspectors, etc., within their organizations who are responsible for those particular facilities so they can prioritize response.

Background. **ShakeMap** *(http://earthquake.usgs.gov/shakemap) is a tool used to portray the extent of potentially damaging shaking following an earthquake. It is automatically generated for both small and large earthquakes in areas where it is available, currently California, the Seattle area, and in much of Utah; It is currently being developed in other areas of the country. It can be used for emergency response, loss estimation, and public information. ShakeMap was developed and is provided by the U.S. Geological Survey and its regional seismic network collaborators as a product of the Advanced National Seismic System (ANSS).*

Despite the popularity and acclaim of ShakeMap for emergency response and post-earthquake information, there is a lack of recognition of the full potential of ShakeMap. That is, critical users need to move beyond simply "looking at ShakeMap," and begin implementing response protocols that use the known shaking distribution in fully automated systems in order to fully realize this potential to prioritize and greatly aid response efforts. To this end the USGS has begun the development of ***ShakeCast***.

ShakeCast, short for ShakeMap Broadcast, will be a fully automated system for delivering specific ShakeMap products to critical users and triggering established post-earthquake response protocols. The ShakeCast system is currently being developed by Gatekeeper Systems, Incorporated of Pasadena, under contract to the USGS and in collaboration with ShakeMap personnel. ShakeCast allows utilities, transportation agencies, and other large organizations to automatically determine the shaking value at their facilities. By setting thresholds for notification of increasing damage states (typically, green, yellow, or red) of each facility type based on the shaking level determined by ShakeMap at that site, notifications can be sent automatically via pager, cell phone, or email to specified operators, inspectors, etc., within their organization responsible for those particular facilities. Notifications can be organized to reach specific recipients depending on the level of damage and where the damage occurred.

Summary tables then allow managers, inspectors, and crews to respond in an organized, prioritized fashion.

As an example of the need for such a system, the California Department of Transportation (Caltrans) has over 25,000 bridges and overpasses under their responsibility in California. In addition, regional control of traffic is under the auspices of 12 different operational centers, or Traffic Management Centers (TMCs). Having an instantaneous snapshot of the likely damage to each will allow Caltrans to prioritize rerouting traffic, closures, and inspections following a damaging earthquake. Caltrans is testing the prototype ShakeCast system at this time.

ShakeCast Features. Organizations using ShakeMap/ShakeCast first download and install a software package on a hardened in-house computer system. Initial setup then involves 1) populating a database of facility locations and types or retrieving these from the user's database, 2) tabulating the fragility to specific ShakeMap parameters (e.g., peak or spectral acceleration) and the corresponding likely damage states for these facilities (damage unlikely, moderate damage, and serious damage thresholds, for example), 3) specifying who receives notifications by listing addresses of facility managers and response personnel (email, pager, cell phone, and 4) selecting under what circumstances the alerts are sent (damage "likely" at specific facilities). In addition to simplified pager notifications, a summary report is automatically made and distributed as configured, providing a prioritized list of facilities based on their likely damage state and including facility locations (including post miles) and the shaking level there. If applicable, this summary information can be populate the user's GIS or other relational database.

As soon significant shaking occurs and a ShakeMap is generated, and if basic (configurable) magnitude or shaking levels are met, the user's ShakeCast software downloads the parameters or maps needed to evaluate each facility. The local facility database is then used to determine the shaking levels at each site, likely damage outcomes are projected, and notifications proceed as configured. ***Critically, all these actions happen automatically 7x24, with no user intervention or action required. Likewise, the users' facility database, perhaps proprietary and often changing, remains under the control of the user, in house.***

ShakeCast can also be configured on the user's end to automatically transfer ShakeMap files (for example GIS maps) to a specified location and initiate additional software processing tools and actions, for example, starting up complex loss-estimation calculations via HAZUS or other loss estimation software applications. Such an action is simply another form of notification.

In addition to real-time notification, another important feature of the ShakeMap/ ShakeCast combination is its built-in capacity to generate and deliver Scenario Earthquakes for evaluating system performance and response capabilities under earthquake conditions. ShakeMap is now used routinely to generate earthquake scenarios for many users; ShakeCast will further allow these system tests to be performed with the same notification tools that will be available and in place when a real earthquake strikes.

Future Plans. While currently in development several aspects of the system will be further enhance based on user interaction and feedback.

The delivery of critical post-earthquake information is a serious business; however, it is a difficult problem. In the ShakeCast development, consideration of reliable, redundant transmission, security and authentication, and documentation and version control of product delivery have all been considered from the onset. In addition, Information Technology security concerns are paramount. However, is quite clear that ongoing IT security concern and changes within organizations will present ongoing development hurdles. These will be addressed as necessary through direct user interfacing in our prototype installations. We expect the solutions we develop through these initial installations to be robust and

therefore can be used for wider distribution of ShakeCast to other users. These tools will also be beneficial for other rapid post-disaster notification and alerting systems.

Further research and development needs to be focused on facilitating the development of fragility (vulnerability) relationships necessary to utilize shaking information in real-time. That is, each utility and transportation organization has an array of facility, building, and component types with varying degrees of knowledge of their individual vulnerability, and highly variable levels of in-house expertise needed to establish and properly use fragility relationships in order to assign likely damage states in real time. A systematic approach to this development, and formal guidelines on the use of such facility/fragility assignments would be more cost-effective than relying on individual organizations to develop their own strategies (a task many have not or may not otherwise take on). We will also begin to focus more on developing and promoting guidelines by which current and future fragility analyses can take advantage of the rapid post-earthquake information provided by ShakeMap and ShakeCast by considering the specific parametric values ShakeMap produces, mainly peak ground motion and spectral accelerations. By providing and promoting this need, ongoing and additional studies will provide fragility values suitable for post-earthquake response and will continue expansion of a fragility matrix from which new users can select appropriate values.

When and where will ShakeMap/ShakeCast be available? Ongoing software development and beta testing of ShakeCast are progressing rapidly. Availability of the fully operational system will be widely publicized through standard Lifeline, Emergency Response publications, newsletters, and web pages including the USGS and Regional Networks, the Applied Technology Council (ATC), the Earthquake Engineering Research Institute (EERI), and American Lifelines Alliance (ALA).

References:

Development of a ShakeMap-based, earthquake response system within Caltrans, 2003, D. Wald, P. Naecker, C. Roblee, and L. Turner, in *Advancing Mitigation Technologies and Disaster Response for Lifeline Systems*, J. Beavers, Ed., Technical Council on Lifeline Earthquake Engineering, Monograph No. 25, August 2003, ASCE.

ShakeMap—A Tool for Earthquake Response, 2003**.** , D. Wald, L. Wald, B. Worden, and J. Goltz, *U.S. Geological Survey Fact Sheet 087-03*, online at http://pubs.usgs.gov/fs/fs-087-0.

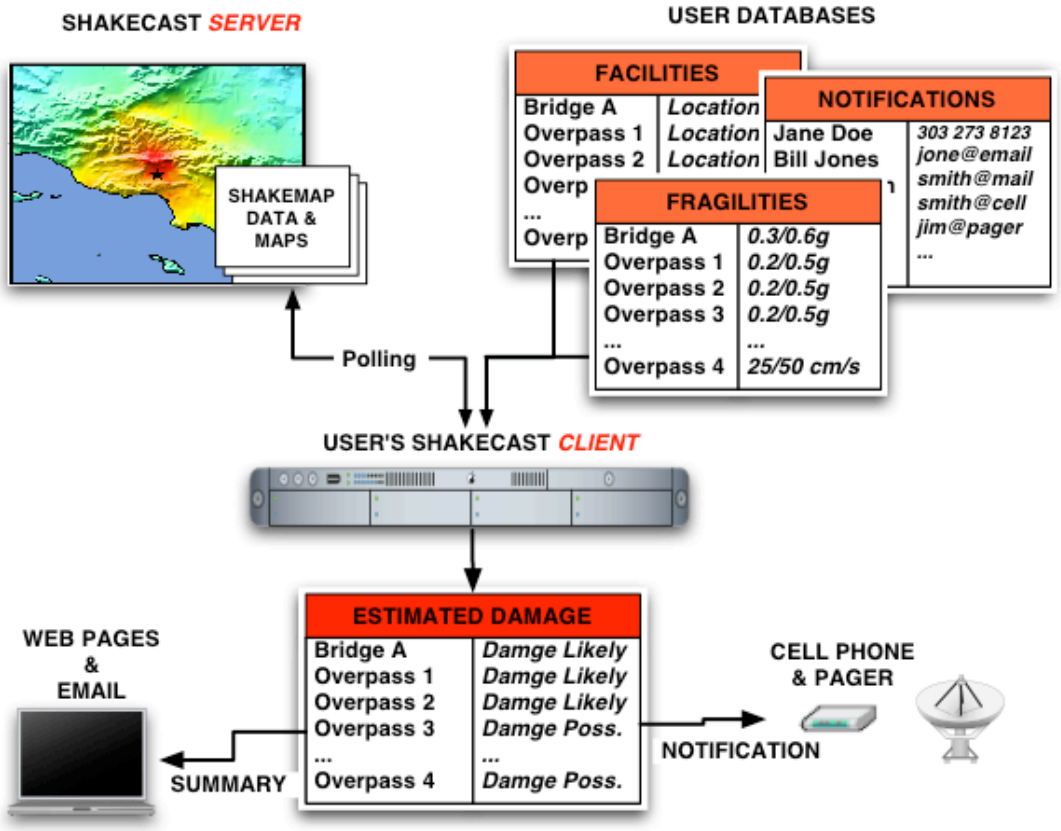
TriNet "ShakeMaps": Rapid Generation of Peak Ground Motion and Intensity Maps for Earthquakes in Southern California, 1999. D. J. Wald, V. Quitoriano, T. H. Heaton, H. Kanamori, Cr. W. Scrivner, and C. B. Worden (1999). *Earthquake Spectra*, Vol. 15, No. 3, 537-556.

ShakeMap Manual – Technical Manual, Users Manual and Software Guide, Wald, D. J., B. Worden, V. Quitoriano, Pankow, K. (2004). Manuscript in preparation.

Contact:

David Wald, Ph.D., U.S. Geological Survey, Golden, wald@usgs.gov, (303) 273-8441

Simplified ShakeCast Flowchart



DEVELOPERS:

USGS science for a changing world

GATEKEEPER SYSTEMS
Systems and Applications for the Internet

SUPPORT FROM:

AmericanLifelinesAlliance

USGS science for a changing world

PROTOTYPE TESTING:

PG&E Pacific Gas and Electric Company®

Caltrans

A.2 Disaster Resistant California Conference Proceedings, Sacramento, 2004.

2004 Disaster Resistant California Conference

New Tools from the California Integrated Seismic Network (CISN) and Advanced National Seismic System (ANSS)

The CISN and ANSS Integrated Products Team[1]

Abstract

To better serve the earthquake emergency management and response, engineering, and scientific communities, several agencies in California have formed the California Integrated Seismic Network (CISN). The CISN is one region of the Advanced National Seismic System (ANSS), a larger system being developed under the auspices of the U.S. Geological Survey (USGS) in cooperation with CISN and seismic network operators in other areas of the US. In this paper we highlight ongoing development of new CISN and ANSS products and web pages, with emphasis on the utility of these products as they pertain to disaster resistance. In additions to an improved web presence, the CISN partners are currently working on improving the robustness of statewide earthquake notification and developing applications to facilitate interpretation of rapid earthquake information and damaging ground motions. One such notification product is ***CISN Display***, an application for visualization of seismicity and ground shaking for critical users. The ANSS is complementing CISN activities by focusing ongoing development of ***ShakeMap***, a tool used to portray the extent of potentially damaging ground shaking following an earthquake, and ***ShakeCast***, an application for automating ShakeMap delivery and triggering established post-earthquake response protocols.

Background - CISN and the ANSS

A modern seismic system is vital for providing timely and accurate information about earthquake activity and earthquake effects and for reducing loss of life and property from earthquake disasters. The partners of the California Integrated Seismic Network (CISN), a

[1]Correspondence can be directed to David Wald, U. S. Geological Survey, Golden, CO, wald@usgs.gov. A list of contributors to this article can be found in the Acknowledgements.

component region of the United States Geological Survey's (USGS) Advanced National Seismic System (ANSS, U.S. Geological Survey, 1999), are committed to rapidly providing advanced earthquake information for disaster mitigation. The members of the CISN are the California Geological Survey, the California Institute of Technology Seismological Laboratory, the University of California, Berkeley Seismological Laboratory and the USGS offices in Menlo Park and Pasadena. The California Governor's Office of Emergency Services is an ex-officio participant in the CISN and is a primary recipient of the rapid post-earthquake information and hazard analyses provided by the CISN.

Advances in Web-based Information

A number of new developments and applications are available via our web sites. We discuss these advances in detail below, but since these and other applications are changing and improving rapidly, we provide more up to date information at the CISN website http://www.cisn.org, and at http://earthquake.usgs.gov, the website of the USGS Earthquake Program which is the umbrella organization for the ANSS. Part of our emphasis for web-based information delivery is a continued emphasis on bandwidth and redundancy: Both CISN and the ANSS have multiple, distributed servers in geographic distributed locations.

Recent Earthquake Maps. For most users, the seismicity maps (Figure 1) provide the starting point for earthquake information. Users typically select a specific region and "drill down" to more detailed maps and summary information about a specific earthquakes. The Recent Earthquake website provides general background as well as direct links to additional information and products. We are experimenting with automatic webpage refreshing of the Recent Earthquake pages, allowing users to see new events show up automatically within minutes of their occurrence. A new feature of the *Recent Earthquake Maps* is the ability to select a fault of interest and follow a link to a detailed USGS Fault Activity Database for that fault.

Earthquake in the News. For earthquakes that are strongly felt or that result in damage and casualties, we create a Special Report that appears on the front page of the CISN & ANSS homepages (Figure 1, M5.0 WYOMING earthquake). This webpage includes links to products, such as geographical and tectonic summary maps of the region, aftershock probabilities and maps, earthquake fault models, hazard maps, and links to news and other information. Informative background information summaries— called *Rapid Tectonic Summaries*—are now produced automatically in many areas of the country and the world, and are delivered through the *Earthquake in the News* mechanism.

Earthquake Summary Posters. For major or damaging events, we also produce a new GIS-based product (Figure?) within a day or two of the event, which summarizes figures from

Earthquake in the News in a poster form (web and downloadable PDF formats). These high quality event overviews provide the basis for briefings and post-earthquake evaluation.

Earthquake Notification Services. Automatic notification in near-real time is available in text messages for email, cell, and pagers. Sign-up can be done at the CISN Notification Services (Figure 2). Formerly, notification was made via an email *List Server*, which limited the range of possible notifications. These services are currently undergoing modifications to improve customization of the range of conditions under which automatic notification occurs (e.g., location, magnitude threshold, time of day, etc.). We are undergoing testing of a *Customized Notification Service*, which entails a database for associated specific users with more detailed notification conditions. We are also allowing users to sign up for *Really Simple Syndication (RSS),* a notification service that brings earthquake information to their screen automatically. Users may be familiar with *RSS* for getting delivery of breaking news headlines. Finally, for 7x24 operations centers we provide a fundamentally improved automatic notification product, *CISN Display*, as discussed below.

Did You Feel It? We create an online questionnaire for any felt earthquake, on which users indicate the perceived level of shaking at their location (specified by zipcode). The software uses this information to generate an intensity map that provides a rapid assessment of the extent of shaking from the human perspective. For regions with sparse seismic station coverage these maps provide a stand-in for ShakeMap (see below).

Strong Motion Data Center. The CISN Engineering Strong Motion Data Center is operated by CGS and provides seismograms for engineering applications typically for events above magnitude 4.0. These data are distributed via the *Internet Quick Report (IQR).* In addition, for more significant events a more complete *Internet Data Report (IDR)* is produced.

CISN Display

This near real-time, interactive Graphical User Interface (GUI, see Figure 3) provides seismicity, ground shaking information (via ShakeMap), and other earthquake products for the 24/7 operations center. Unlike web pages, earthquake information is pushed to *CISN Display* in near real-time that receives data over the Internet. It is a stand-alone software application (written in Java) and can be run on several operating systems. Central to the GUI is a GIS mapping engine that is capable of plotting user-customizable themes (facilities, roadways, etc.) that allows comparison of user's infrastructure along with seismic hazards to help evaluate the situation interactively. The primary application of CISN Display is real-time seismicity monitoring; the system automatically posts updates and can provide audible beeps to alert the user.

In addition, other products associated with the event, like ShakeMap, Did You Feel It? Maps, and aftershock probabilities, are provided by the *Display* as they become available. The system can automatically download ShakeMap overlays as GIS layers, which can be modified and joined with other layers to form a clear picture of the severity and distribution of shaking with respect to critical infrastructure. *CISN Display* is currently in beta-testing within several organizations and version 1.0 is expected to be released in Spring 2004. It performed well during the recent magnitude 6.5 San Simeon, California, earthquake sequence.

ShakeMap and ShakeCast

ShakeMap. ShakeMap is a tool used to portray the extent of potentially damaging shaking following an earthquake (see **http://earthquake.usgs.gov/shakemap**). This is but one end product of a modern seismic network capable of producing near real-time, magnitude, location, and ground-motion parameters. The rapid availability of these maps is of particular value to emergency response organizations, utilities, insurance companies, government decision-makers, the media, and the general public. The ANSS and CISN are continuing the development of the ShakeMap system. In particular, they are working to integrate the southern and northern California systems for statewide coverage.

We have finished a draft ShakeMap Manual, including a Technical Manual, Users' Guide, and Software Guide (Wald et al., 2004). The Applied Technology Council's *ATC-54: Guidelines for Using Strong Motion Data and ShakeMaps in Post-Earthquake Responses recent* (Applied Technology Council, 2003) provides examples of how ShakeMap can be used. We have also published a new four-page ANSS ShakeMap USGS Fact Sheet, for more general information and outreach (Wald et al., 2003).

ShakeCast. When a potentially damaging earthquake occurs, utility and other lifeline managers, emergency responders, and other critical users have an urgent need for information about the impact on their facilities so they can make appropriate decisions and take quick actions to ensure safety and restore system functionality. ShakeCast, short for *Shake*Map Broad*cast*, is a fully automated system for delivering specific ANSS ShakeMap products to critical users and triggering established post-earthquake response protocols. ShakeCast allows utilities, transportation agencies, and other large organizations to automatically determine the shaking value at their facilities, set thresholds for notification of damage states (typically, damage unlikely, moderate, or serious) for each facility, and then automatically notify (via pager, cell phone, or email) staff within their organizations who are responsible for those particular facilities so they can prioritize response. The system will initiate post-processing software applications automatically (for example, loss estimation routines). Currently, USGS "pushes" ShakeMap electronically (using ftp) to utilities and other critical users, but ShakeCast will allow this to be replaced with a subscriber service, avoiding firewall issues, and providing more robust delivery from redundant ShakeMap generation sites and distributed ShakeMap servers. A simplified flowchart of the ShakeMap/ShakeCast system is shown in Figure 4.

Prompt Assessment of Global Earthquakes (PAGE)

PAGE is a prototype system that monitors the U.S. Geological Survey's Advanced National Seismic System's (ANSS) near real-time global earthquake solutions and automatically identifies events that will be of societal importance, well in advance of ground-truth news accounts. Events that are likely to have caused human suffering and significant damage, or events that are widely felt and will therefore generate public and media attention, are so classified. *PAGE* makes this assessment within a few minutes of an earthquake's detection based on analysis of estimated shaking levels and evaluation of the vulnerability and the population at risk. For significant events, *PAGE* automatically generates and distributes a summary impact statement to emergency response teams, the media, and the general public. The basic concept for *PAGE* is straight-forward. However, the implementation, gathering of the necessary data sets, testing, and most importantly the effective use of our results will require significant system development and communication with potential users.

Acknowledgements

The ANSS Integrated Products Team consists of D. Wald, D. Oppenheimer, D. Given, H. Benz, W. Savage, R. Buland, and T. S. Yelin. Contributors to this article are: D. Wald, L Wald, D. Oppenheimer, H. Benz, W. Leith, J. McCarthy, R. Simpson, S. Schwarz, B. Worden, V. Quitoriano, H. Rico, E. Hauksson, L. Gee, P. Earle, and L. Lystoka.

References

Applied Technology Council (2003). *ATC-54:* Guidelines for Using Strong Motion Data and ShakeMaps in Post-Earthquake Responses, published by Calif. Geological Survey & Applied Technology Council. Available online at http://www.atccouncil.org/.

U. S. Geological Survey (1999). An Assessment of Seismic Monitoring in the United States Requirement for an Advanced National Seismic System, USGS Circular 1188.

Wald, D. J., L.Wald, B. Worden, and J. Goltz (2003). ShakeMap—A Tool for Earthquake Response, *U.S. Geological Survey Fact Sheet 087-03.* Available online: http://pubs.usgs.gov/fs/fs-087-0 or Contact L. Wald (lisa@usgs.gov) for copies.

Wald, D. J., B. Worden, V. Quitoriano, and K. Pankow, (2004). ShakeMap Manual: Technical Manual, Users Guide, And Software Guide, U.S. Geological Survey Open-File Report, in preparation.

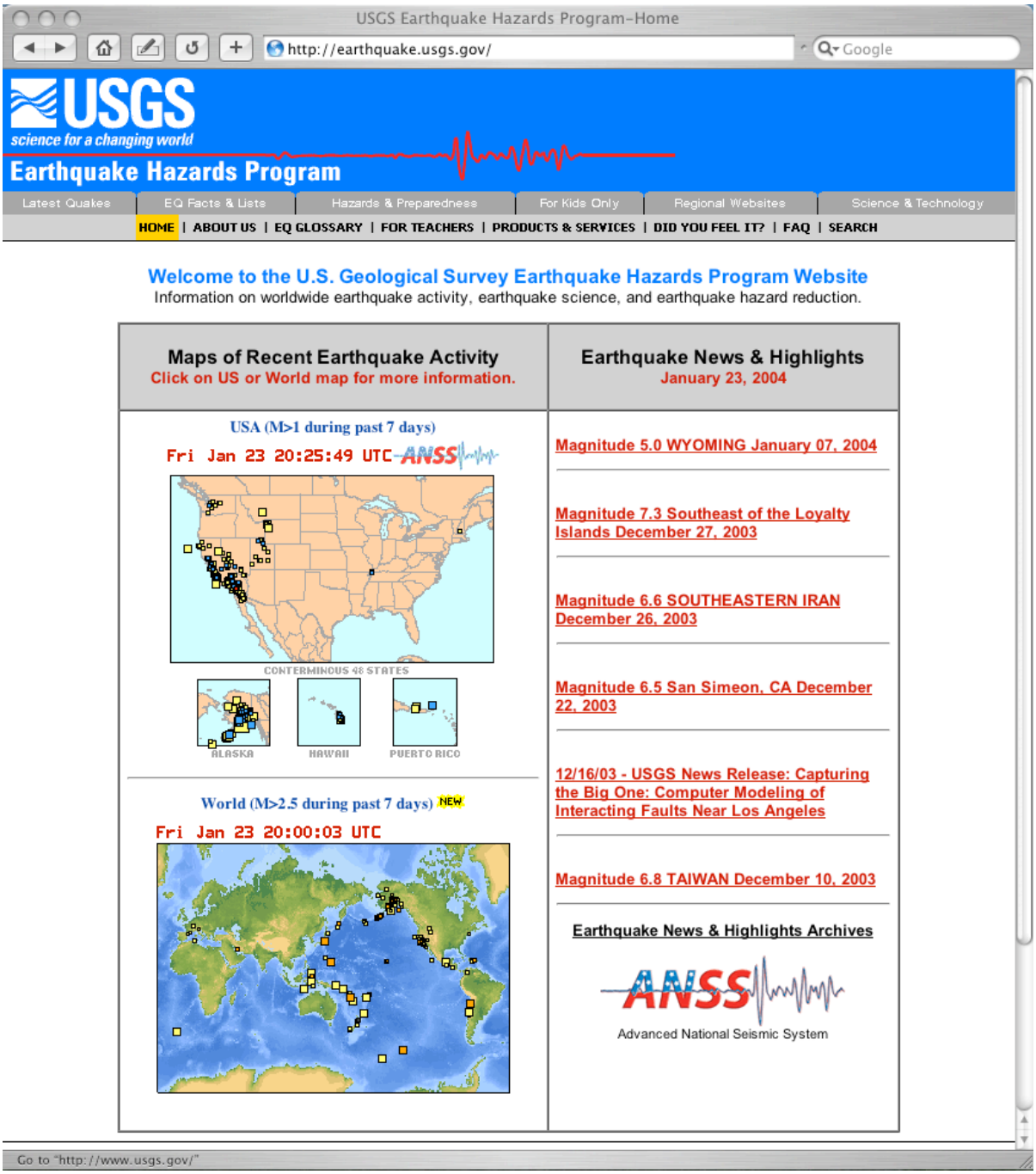


Figure 1. U. S. Geological Survey's Earthquake Program and Recent Activity page.

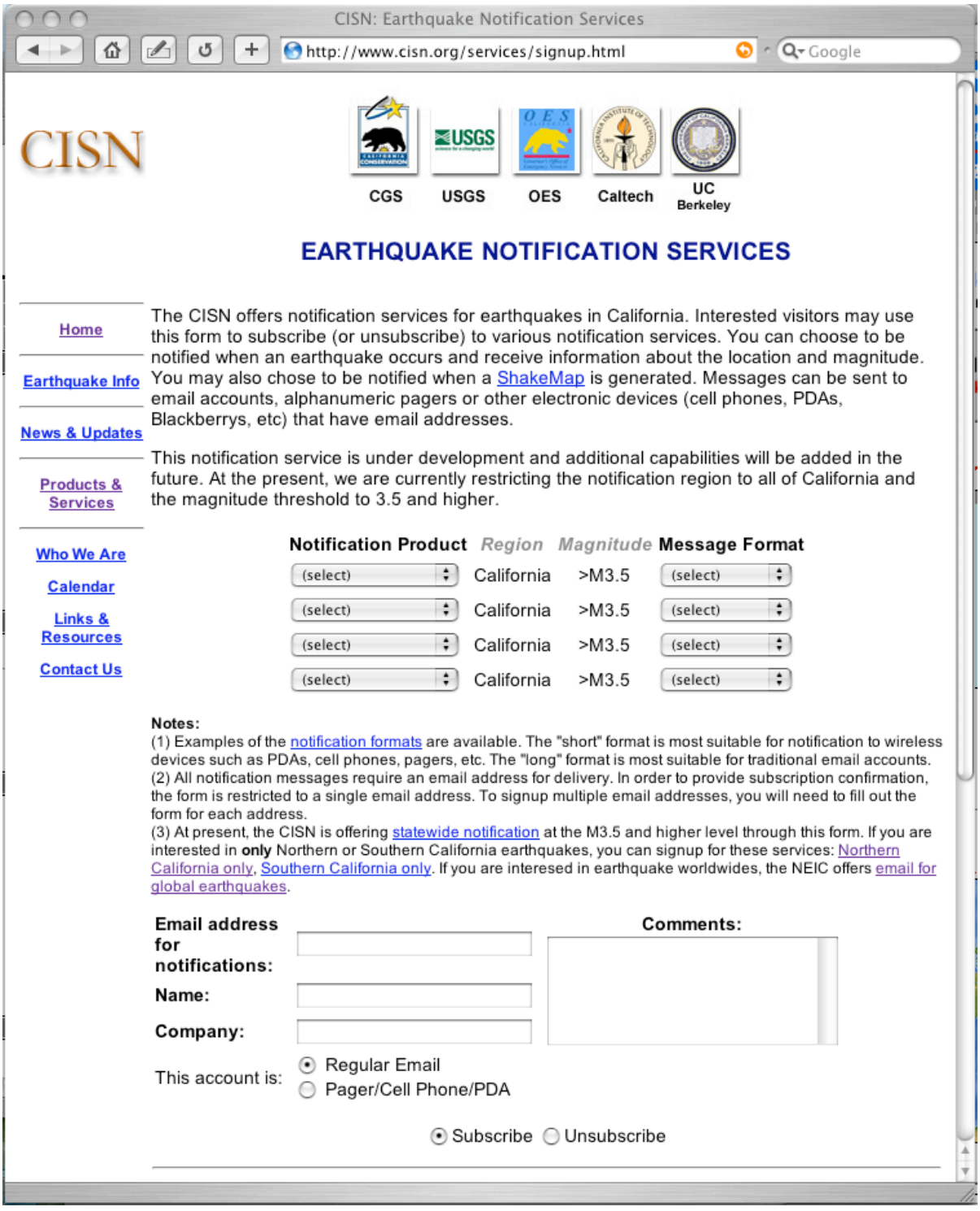


Figure 2. CISN Notification Services web page.

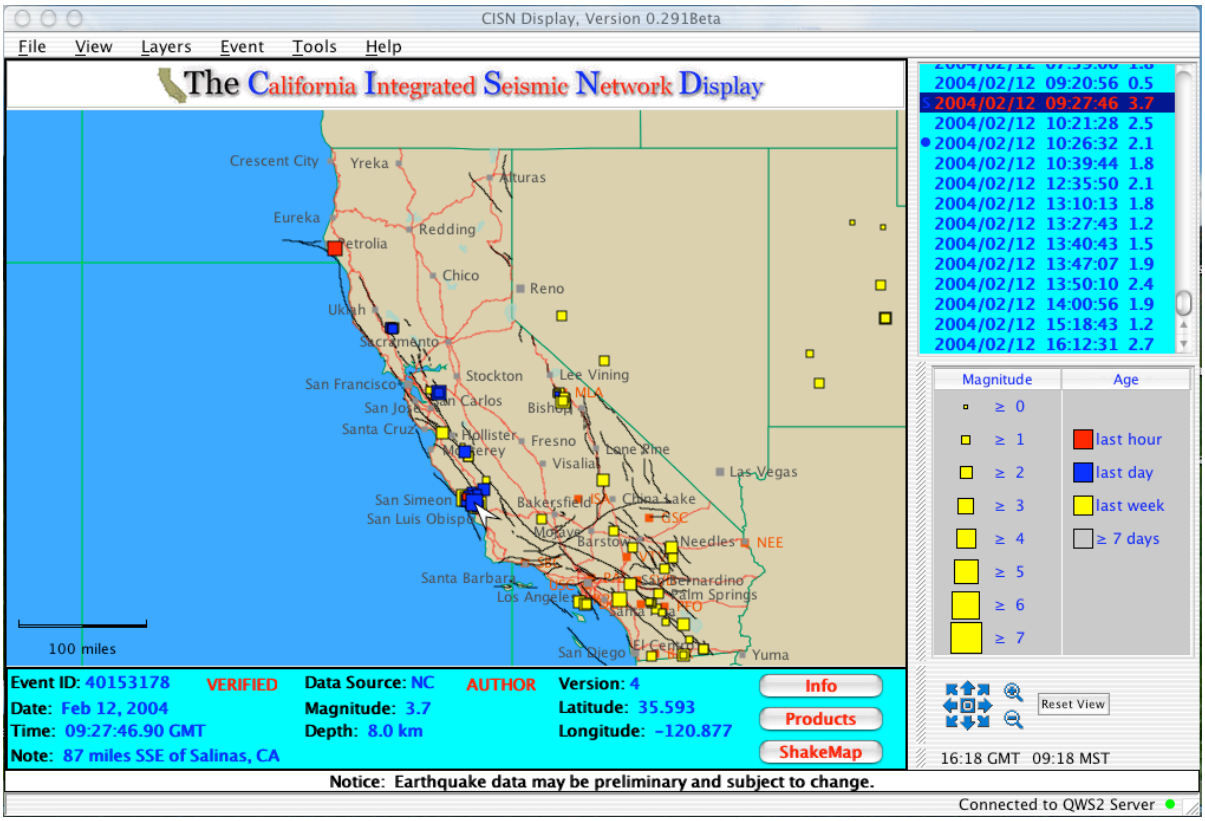


Figure 3. CISN Display's Graphical User Interface.

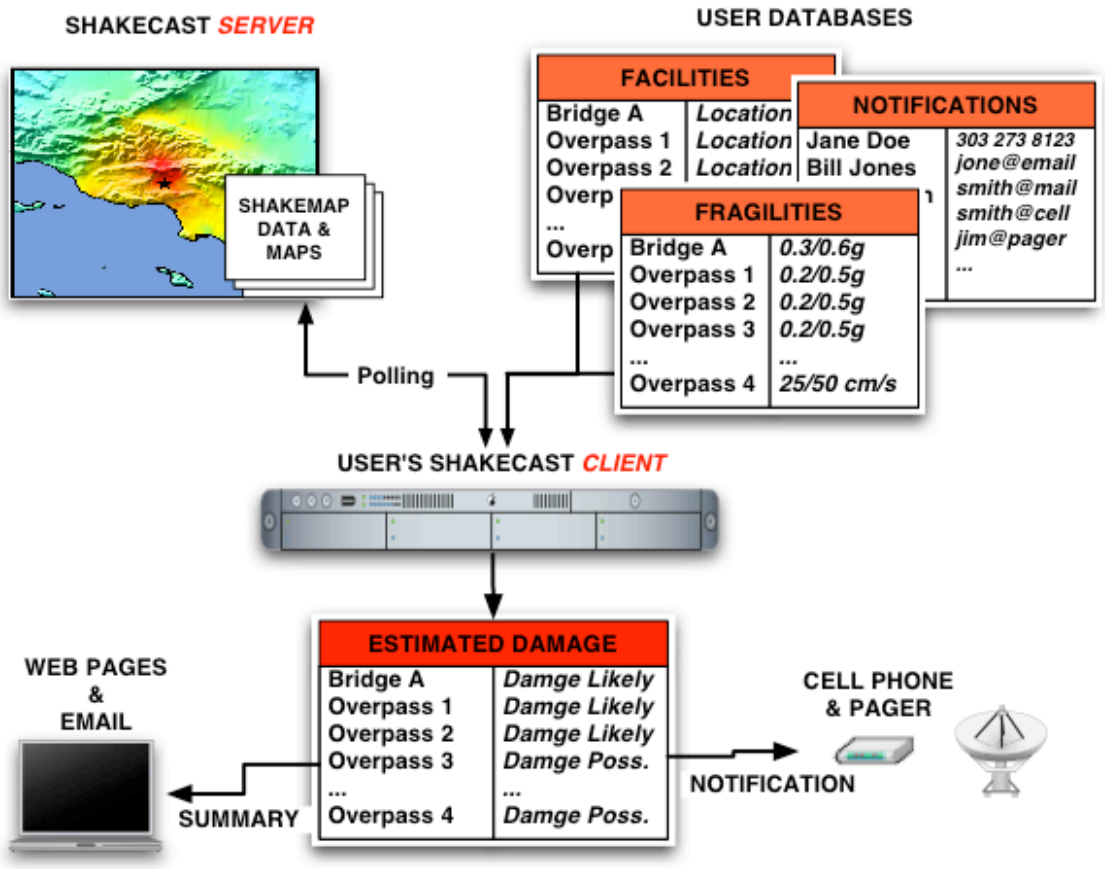


Figure 4. ShakeCast Flowchart.

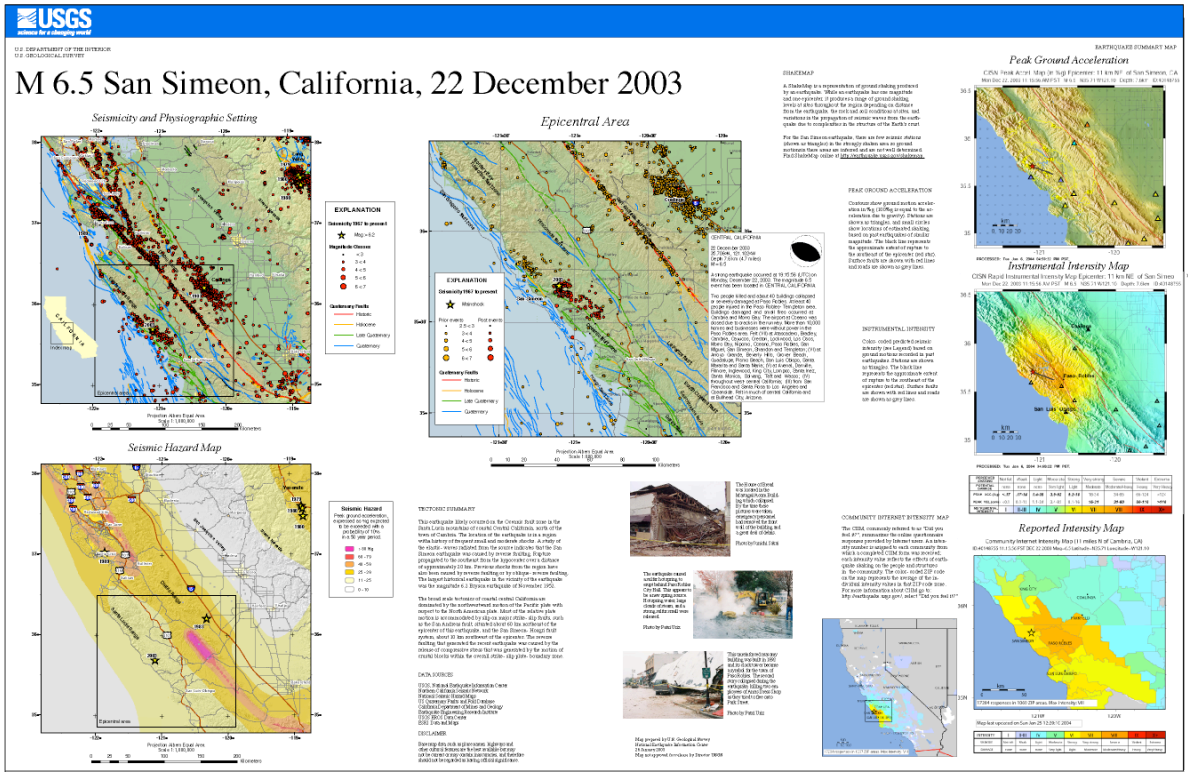


Figure 5. Earthquake Summary Poster example (for the magnitude 6.5 December 22, 2003, San Simeon, California, earthquake).

A.3 Abstract for the National Earthquake Conference, St. Louis, Sept., 2004.

What Just Happened?——Rapid Post-Earthquake Information from the Advanced National Seismic System (ANSS)

David J. Wald

U. S. Geological Survey, Golden, CO, wald@usgs.gov

To better serve the nation's emergency management and response, engineering, and scientific communities, more and better seismic data are being collected and more powerful and useful data products are being developed under the auspices of the Advanced National Seismic System (ANSS) by the U.S. Geological Survey (USGS) and its ANSS partners in academia, government, and industry. New, rapid analyses are available via our web sites, with distributed servers providing both bandwidth and redundancy during earthquake disasters. ***Recent Earthquake Maps***, ***Earthquakes in the News***, and ***Earthquake Summary Posters*** provide rapid post-earthquake summaries of seismicity, tectonic context, earthquake effects, and other important information for significant events, not only nationally but also globally. In addition, we are moving beyond enhanced earthquake web pages to push earthquake information to users who have a need for near real-time earthquake analysis. One such notification product is ***CISN Display***, an application for visualization of seismicity and ground shaking for critical users, developed primarily by the ANSS California Integrated Seismic Network (CISN) partners. The first release of **QWEmailer**, also now available from CISN, enables users to automatically issue earthquake notification (email and short text messages) specific to their needs. The ANSS is complementing CISN advances through ongoing development of ***ShakeMap***, a tool used to portray the extent of potentially damaging ground shaking following an earthquake, and with ***ShakeCast*** (http://www.shakecast.org/), an application for automating ShakeMap delivery to users and facilitating notification of shaking levels at user-selected facilities. Finally, we describe advancing uses of and development of ***Did You Feel It?***, a popular citizen-science, internet-based tool for rapidly mapping post-earthquake seismic intensities, which is particularly useful in areas of the U.S. not yet well covered by ANSS instrumentation. Development of these and other products will be discussed in detail. These applications are changing and improving rapidly; up-to-date earthquake information is available online at http://earthquake.usgs.gov/ and links to new software can be found at http://www.cisn.org/software/. These products rely fundamentally on magnitudes, hypocenters, and ground motion data, including urban arrays, collected and interpreted at regional and national seismic data centers under the ANSS. Improvement in accuracy and quality of these critical products depends directly on the growing national investment in more instruments in urban settings.

DRAFT

System Specification for

ShakeCast

"ShakeCast: Delivering Earthquake Shaking Data to the People Who Need It"

Software Version 1.0
Documentation Version 1.0

July 2004

Gatekeeper Systems

Applications and Systems for the Internet

1010 East Union Street
Suite 205
Pasadena, CA 91106-1756

Phone: +1 626 449 8135
Fax: +1 626 440 1742

E-Mail: info@gatekeeper.com
URL: http://www.gatekeeper.com/

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure of this document or of the software described herein is governed by the terms of a License Agreement or, in the absence of an agreement, is subject to the restrictions stated in subparagraph (c) (1) of the Commercial Computer Software –Restricted Rights clause at FAR 52.227-19 or subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, as applicable. Contractor/Manufacturer is Gatekeeper Systems, 1010 E. Union St, Pasadena CA 91106, 626 449 8135, 800 424 3070, Info@Gatekeeper.com, http://www.gatekeeper.com/.

Form Number: GKS 2003-6

Table of Contents

Chapter 1 Introduction

This document describes the ShakeCast System. It is intended for software developers, database administrators, and system administrators who work with the ShakeCast system software. A companion document, the *ShakeCast User's Guide*, describes the features and use of the ShakeCast System from the point of view of end users.

This Chapter introduces the function and purpose of ShakeCast This Chapter also introduces and defines some of the main concepts and terms used in the rest of the document.

1.1 Introduction to the ShakeCast Project

The rapid and reliable dissemination of detailed earthquake information is of great importance for public safety and emergency response. This information is needed by all kinds of facility owners, such as municipalities, utilities, building managers, schools, and many others. Such organizations would like to be "consumers" of earthquake information, but currently have no simple technology that they can use to readily access and make use of earthquake information.

The ShakeCast System is designed to be a simple, reliable, and widely deployable software tool that any modestly capable computer user can install on their computer to receive and make use of customized and personalized earthquake information. We call the system ShakeCast (short for "ShakeMap Broadcast") because its purpose is to broadcast ShakeMaps. ShakeCast consists of a receiver component (client) and a transmitter component (server). The information to be disseminated via ShakeCast is the output of the USGS ShakeMap system, which provides early estimates of the severity of shaking during an earthquake and thus is a good tool for estimating the likelihood of damage to structures and societal impact.

The ShakeCast software does much more than simply display maps of the areas affected by an earthquake. It will also:

- Automatically receive and process notifications of earthquakes
- Let you define locations (representing structures and facilities) of interest to you, and set shaking thresholds that will trigger automatic notification
- Provide you with electronic notification (pager, email, personal web pages, etc.) of events and projected shaking intensity at facilities you specify

- Reliably manage the receipt of updated shaking data from multiple ShakeCast servers distributed around the Internet, so that you have a excellent chance of receiving an uninterrupted and authenticated data feed even after a major event
- Easily integrate with in-house GIS systems, control systems, utility outage management systems, and other business systems in your organization
- Provide a mechanism for continual end-to-end testing of the ShakeCast system, so that you can be assured that the system is working properly when you eventually need it

The benefits of the ShakeCast system are substantial:

- ShakeCast allows individuals and facility owners to make widespread and immediate use of the beneficial information already in ShakeMap. ShakeCast takes advantage of the very substantial investment already made in ShakeMap and in the very large seismic monitoring infrastructure behind it.
- The ShakeCast system provides quantitative metrics on the use of ShakeMaps both before and after an earthquake. This data is then available for policy decisions on the future direction of the ShakeMap and ShakeCast systems. The ShakeCast system will return detailed information about these users that will help ShakeMap planners better understand how ShakeMap data is used.
- ShakeCast should help engage and involve managers and policy makers at a wide variety of institutions (e.g., state transportation departments, municipal governments, emergency responders, utilities, etc.) who are concerned about timely receipt of earthquake shaking data.

The next Section of this document provides a brief background discussion on the history and challenges of the broadcast of earthquake information and then describes how the ShakeCast system addresses these issues.

1.2 Background

Historically, the only objective data about possible damage available immediately after an earthquake has been an early estimate of "location and magnitude". Such a simple metric can easily be broadcast using simple communications technologies such as radio, television, email and pagers. For facility managers responsible for geographically dispersed facilities, however, a simple location and magnitude is of limited practical utility because it does not yield a meaningful estimate of the likelihood of damage at each facility with sufficient detail to guide facility managers in their initial emergency response activities.

A much more useful metric of damage likelihood has been devised: a "ShakeMap". This map estimates, with accuracy appropriate for use by ShakeCast consumers (utilities, school districts, municipalities, etc.), the "shaking" (intensity, peak ground acceleration, velocity, and spectral response) that structures were likely subjected to during an earthquake. A ShakeMap provides a good first-order estimate of the likelihood of damage, and when enhanced by data about the structural resilience of facilities, ShakeMap can provide extremely valuable information to facility owners directing early response to an earthquake.

The United States Geological Survey (USGS) has developed an automated system for computing ShakeMaps. This system is now generating ShakeMaps within a few minutes after an earthquake, which is early enough to be of considerable value for emergency response. This

system is now operating in four Western regions of the United States (Southern California, Northern California, the Pacific Northwest, and Utah). Other regions may come online in the future.

Once people know an earthquake has occurred, they can visit the ShakeMap web sites to obtain maps and data. However, few organizations have the technical means use this data other than by "just looking at it". Although a few companies have in the past received a "push" of ShakeMap data, there was not a reliable dissemination method for promptly broadcasting this data to the many thousands of individuals and organizations that need it. Nor is the information readily available in a form that those individuals and organizations can easily use.

Disseminating earthquake shaking information is a difficult problem, for a variety of technical reasons:

- Wide adoption of such a system will depend upon reliable transmission of data, even if communications networks have been damaged.
- There is a wide variety of data products that must be delivered for different users, including different metrics of shaking intensity (e.g., peak acceleration, velocity, etc.).
- The spatial variability of shaking intensity is critical, and most consumers of the data are likely interested only in data for specific locations where they own or manage facilities that might be damaged by earthquake shaking.
- There are difficult technical problems with triggering and alarms, to ensure that the proper people and systems are notified when specific events occur but that false alarms do not occur.
- It is important to ensure the authenticity of the earthquake data, so that organizations making operational decisions can be certain that the information came from a reliable source.
- It is important to keep track of updated versions of each data product so that as new information arrives it can be seamlessly related back to the original event and previous version of the data and to decisions that may have been made based on the prior data.
- There are important issues of event timing and timeliness, and it is critical that data from multiple sources relating to the same event or closely related events (e.g., aftershocks) be properly correlated.
- There is wide variance in the types, sophistication, and purposes of the organizations and computer systems that will consume the data.

Further major complications center on the infrequency of events. Historically, earthquake-related automated systems have not functioned flawlessly, because earthquakes are very infrequent, widely distributed, and extremely variable in their mechanism and damage characteristics. It is a significant technical challenge to build a network of interacting computer systems that is guaranteed to function when an earthquake actually happens. The system must be robust enough to "set it and forget it," possibly for years, yet flexible enough and simple enough to manage so that new data products and tools can be added without impacting reliability.

1.3 ShakeCast Software Design Philosophy

Fortunately, there exists in the Internet community most of the software tools and protocols for building a robust system that meets the technical requirements of an earthquake information dissemination system. What is required is small amount of new software, a software framework for the broadcast of ShakeMaps, and set of protocols that defines how the system utilizes already-available Internet tools. These elements can then be implemented in an open source reference system that can be easily installed by any modestly capable system administrator, and easily extended and customized for local applications by any developer with a modest knowledge of current Internet tools.

The design of ShakeCast takes full advantage of the software tools that are widely available and commonly used on the Internet. The ShakeCast system is intended as not a final software solution to be used identically by every site, but as a reference implementation: working software that others can either use or extend as they wish. The reference implementation is freely downloadable and will perform all of the basic functions needed by a consumer of earthquake shaking data, including:

- Automated notification (email, pager, etc.) of excessive "shaking" (using various definitions) at a list of specific locations
- Notification regarding events of various kinds internal to the ShakeCast system, including system activity above a specified threshold or failure of internal test procedures
- Reliable, coordinated, and non-duplicated receipt of data from multiple sources
- "Hooks" to incorporate this data in other corporate information systems, such as GIS systems, control systems, internal alarm systems, and the like
- Simple installation on Windows NT/2000/XP, Linux, FreeBSD, Solaris, and other operating systems, including automatic registration (“sign-up for data”) with ShakeCast broadcasters
- A robust quality assurance system that continually verifies that the entire ShakeCast system is functioning correctly end-to-end
- A complete system of usage monitoring, logging, and reporting, so that the administrators of individual ShakeCast receivers and the administrators of the ShakeCast servers will be able to assess how and how much ShakeMaps are being used.

Our design goal for the ShakeCast reference implementation is that a modestly knowledgeable personal computer user can install the entire system and begin receiving useful, reliable, authenticated, location-specific reports of earthquake shaking with about an hour of effort. A further goal is to create a community of software developers who work for enterprises who are consumers of ShakeMap data. This community of developers is then able to continue enhancement and extension of ShakeCast so that ShakeMap information can be made more readily usable by their organizations.

1.4 ShakeCast Users

ShakeCast receiver software is designed to be easy to install and configure, and will run on a wide variety of computer systems. It is intended for a wide variety of user organizations:

- *Public Utilities* may use ShakeCast to generate alarms indicating possible impacted facilities and to direct initial response and post-earthquake inspection efforts.
- *Public Safety Agencies* may use ShakeCast to help plan the deployment of emergency resources
- *Property Mangers* will use ShakeCast to prioritize the dispatch of inspection and repair personnel to their properties
- *Individuals* will use ShakeCast to ascertain the likelihood of damage or possible injury to loved ones, homes, or property

1.5 Technical Features of the ShakeCast Software System

The following tables and figures briefly describe some of the important technical features and design elements of the ShakeCast software and protocols.

ShakeCast Server Software Features	
Feature	**Description**
Multi-platform	Available on PCs and Unix systems
Easy installation and configuration	Installation and basic configuration in less than an hour in most cases
Automated registration	Automatic software registration with ShakeCast broadcast systems, including registration with servers in multiple regions
Integrated quality assurance and testing	The client software will participate in the ShakeCast system's comprehensive end-to-end testing procedures to provide high confidence in proper system function during an earthquake. Broadcast data will be checked for authenticity, correctness and completeness.
Automated notification	The system will notify a list of people of earthquake-related events via email, pager, and other mechanisms. Notification can be based on shaking intensity (e.g., "peak ground acceleration at Mom's house greater than 0.3g") using any of the shaking metrics of the current or future ShakeMap system. Users can "sign up" for notification via a Web page on their local ShakeCast system.
Personal web pages	Provide local ShakeCast users the ability to view shaking data (including maps, events, and alarms) on personalized web pages served from their local ShakeCast server without each user needing to access the main USGS ShakeMap systems.
Data version support	Revise and re-issue notifications as new data arrives. Maintain permanent record of the sequence of messages and notifications issued.
Locations and thresholds database	Maintain local list of locations of interest and notification thresholds.
External program integration	ShakeCast can trigger the execution of external programs for further event and data processing.
Basic GIS tools	Tools for working with GIS format ShakeMap data. Display your own facilities and ShakeMap data in a Web-based map generated

	locally on your receiver system.
Simple administration	Web-based configuration and administration interfaces
High quality documentation	Professionally developed documentation and support materials

System Reliability Features	
Feature	**Description**
Support ShakeCast from multiple servers	Protocol and tools will efficiently use multiple simultaneous broadcast servers. Each client can sign up with multiple servers on different networks to improve the likelihood of successful event notification and data delivery. Clients will efficiently use multiple servers, improving response-time for all ShakeCast users.
Cascading servers	ShakeCast servers can be cascaded (a tree of servers) to improve scalability and reliability. For example, a primary corporate server can be configured to feed multiple internal ShakeCast client systems, or a central State server can be configured to feed departmental servers in many departments or regions.
Integrated quality assurance and testing	Client software will participate in frequent end-to-end system tests, report errors to the local system administrator, and will log error types to the ShakeCast developers to aid in designing future software enhancements.
Comprehensive message logging	An integrated and comprehensive message logging and analysis system will aid the local system administrator in tracking system performance.
Support for Application Service Providers (ASPs)	The software will be suitable for customization by Application Service Providers who want to add further value to the system. ShakeCast could also be used by large “portal” sites (e.g., AOL, Yahoo) who want to provide their users with customized earthquake information the same way they currently provide customized weather information.

1.6 ShakeCast Software Development Plans

This document describes the capabilities of the ShakeCast Reference Implementation. The Reference Implementation is the most basic system that implements the core (required) features of the ShakeCast design. The Reference System is freely available software, in the public domain.

However, it is hoped that some software developers will take the Reference System and extend it, adding functions that their organization finds necessary or useful. The intent of the ShakeCast Project is to facilitate the development of such software extensions in an Open Source development environment, so that everyone in the extended ShakeCast community can take advantage of each other’s efforts.

The sharing of new software is also facilitated by the use of XML (the Extensible Markup Language) to express all ShakeCast metadata. All of the data exchanged in ShakeCast is described and internally documented in XML. By making the ShakeCast data formats easily readable,

public, and transparent, and by providing easy to use tools to define these data formats, the ShakeCast project intends to promote the sharing of shaking data products and the tools that operate on these data products.

Software Development Model	
Feature	**Description**
Open source development	Use an open source development model. Encourage shared development by external organizations so that ShakeCast improvements made by one organization can be incorporated back into the product.
Web-based transport	ShakeCast communication will be via HTTP/HTTPS.
XML-based metadata	Self-describing data products so that new data types, data formats, and sources can be added easily
Imbedded security and authentication	All external communication include support for public key signatures and manifests.

1.7 ShakeCast Terms and Terminology

The terminology used in ShakeCast may differ slightly from that used in other information systems.

1.7.1 ShakeCast Server

A *ShakeCast Server* is a computer that is running the ShakeCast Server Software. The server may or may not be acting as a server in the traditional sense: sending data downstream to another ShakeCast Server or a ShakeCast Client. Instead, a server may be only receiving data from another ShakeCast Server and making that data available on the web.

1.7.2 Upstream and Downstream

ShakeCast machines are more easily defined in terms of being *upstream* or *downstream.* An upstream machine sends ShakeMap data to a downstream machine. The request to send the data may originate on either machine.

1.7.3 Event

An *event* is a seismic event – an earthquake. All events have a globally unique and permanent identifying number, called an event ID. The event ID is assigned by the seismic monitoring systems, not by ShakeCast or ShakeMap. Once created, events may not be deleted, although they may be marked as "canceled" to indicate that an event is anomalous and should no longer be considered.

1.7.4 ShakeMap and ShakeMaps

ShakeMap is a software system for computing maps of shaking intensity. It uses data from networks of seismometers and other sources to estimate shaking intensity as measured by a variety of physical or instrumental metrics such as peak acceleration, velocity, spectral response,

instrumental intensity, and so on. There are very few ShakeMap systems, all of them operated by teams of seismologists and their professional support staff.

ShakeMap is also the name for the maps produced by the ShakeMap system.

1.7.5 Exchange

An *exchange* is when one ShakeCast server "talks to" another and sends information. An exchange can be as simple as "Hello, I'm still listening" or as complex as the transfer of a complete set of GIS files for an event.

1.7.6 Product

A *ShakeMap Product* (or just *Product*) is a result of ShakeMap processing. When ShakeMap processes a seismic event, it produces maps for many different metrics (i.e., peak acceleration, velocity, etc.). Each of these maps may be produced in many different data formats (i.e., as a grid of scalar values, as a GIS shapefile, as an image in JPEG format, as an image in PostScript format, etc.).

Each combination of event, metric, and format is a different *product.*

1.7.7 Product Metric

A *product metric* (sometimes just called a metric) is a measure of shaking such as acceleration, maximum velocity, and so on. Each metric for an event is provided by one or more different ShakeMap products.

1.7.8 Facility

A *facility* is a location that is to be monitored by ShakeCast. A facility is typically a building, bridge, highway, or similar man-made structure. The location of the facility must be known so that ShakeCast can attribute various levels of shaking at that location, and the facility may have associated fragility measures in one or more of the shaking metrics.

1.7.9 Fragility

Fragility is the measure of likely damage at a particular facility when a certain level of shaking is exceeded, as measured in a particular metric (e.g., "peak acceleration at the period of one second").

1.7.10 Damage Level

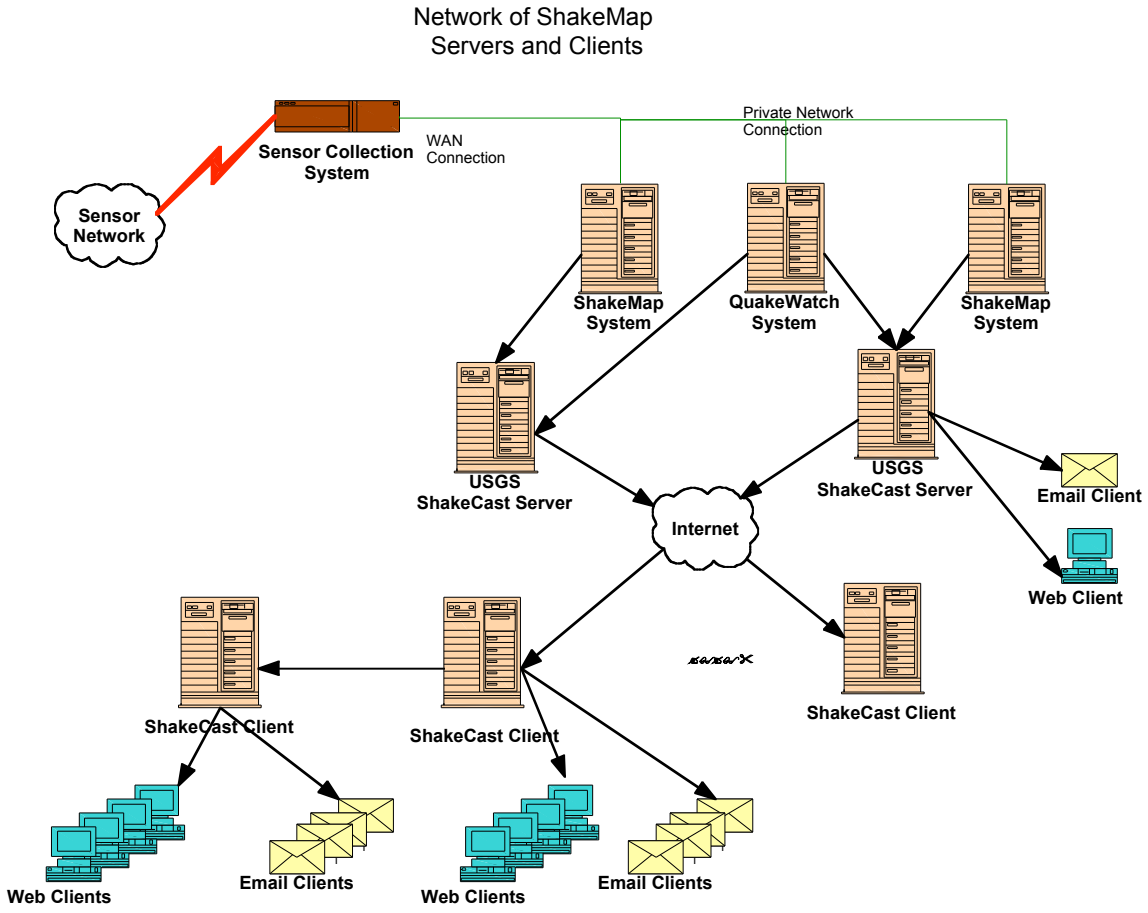
There are classes of fragility associated with each facility in each metric. The *damage level* is the class or category of shaking intensity experienced at a particular location. Damage levels are typically assigned as "No damage expected", "Some damage expected", and "Damage likely", or "green", "yellow", and "red". Damage levels are locally defined on each ShakeCast Server, and different organizations may use different categories or a different number of categories.

1.7.11 Notification

Notification is the process of electronically notifying a ShakeCast end user that a particular damage level is estimated at a particular facility from a certain event. Notifications can be delivered in a variety of electronic forms, including as an email message or an electronic pager message.

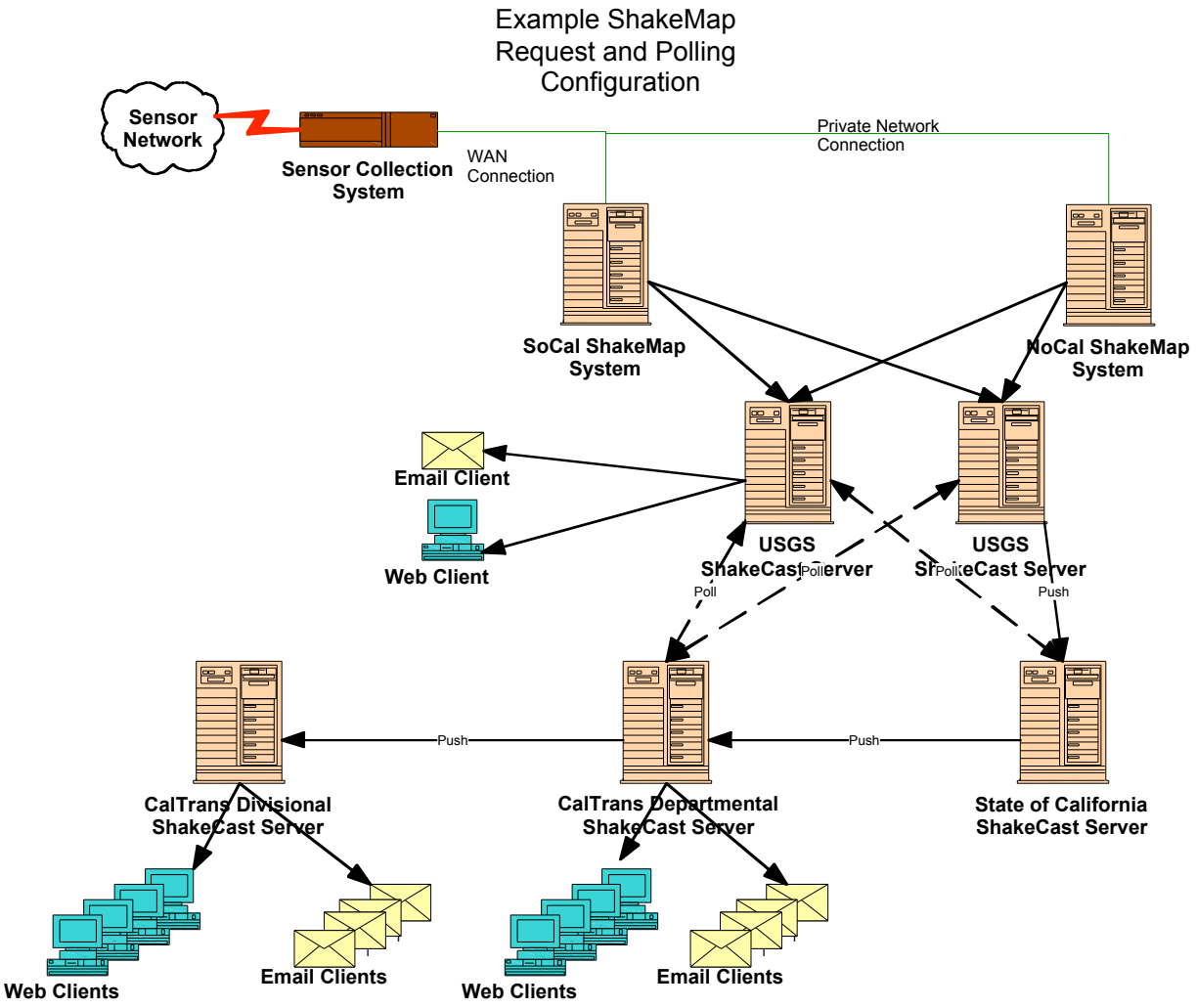
1.8 ShakeCast Network Topology

ShakeCast is may be visualized as a network of interconnected computer systems. The basic structure of the ShakeCast network is shown in the following figure.



ShakeCast servers can be configured to receive unsolicited input of ShakeCast events and products. They may also be configured to periodically poll upstream ShakeCast servers for information. An example configuration showing how servers might be configured for both polling and "push" is shown in the following figure. The figure shows how some servers might be configured to poll for data, some to receive a push of data, and some configured for both. The figure also shows how some servers can be configured to receive data via two or three paths, pro-

viding increased protection against failure of upstream servers or of the network connection to those servers.



Chapter 2 ShakeCast Request Format and Parameters

ShakeCast exchanges information between two computers in the form of "requests". This Chapter of the ShakeCast Specification describes the use, format, and parameters of these requests.

2.1 Introduction to ShakeCast Requests

ShakeCast Requests are simple "messages" exchanged using the HTTP protocol (the protocol that powers the World Wide Web). The requests have the following characteristics that make them a convenient mechanism for exchanging information in a system for communicating critical data about earthquakes:

- It is relatively easy to build software that uses HTTP. There are a great many tools, software libraries, and other software resources available for using HTTP.
- HTTP is well understood and easily managed by network administrators who are responsible for network security.
- The ShakeCast requests are always idempotent. That is, a ShakeCast request can be repeated, and repeated requests will have the same effect as the first request. This property is very important in situations where multiple computers are simultaneously communicating critical emergency information to many other computers, and the sequence of multiple requests cannot be assured or determined in advance.
- ShakeCast requests always contain information to ensure the integrity of the message. The HTTP protocol is designed so that applications can be assured that HTTP messages are not damaged in transmission. ShakeCast extends this assurance by identifying the parties in a message, by exchanging metadata ("data about data") that describes the message contents, and by sending extra information the receiver can use to easily verify that the message was received without being somehow damaged or truncated.

The following sections of this Chapter describe ShakeCast Requests.

2.2 The ShakeCast Request Format

All ShakeCast Requests are actually ordinary World Wide Web requests. Specifically, ShakeCast Requests are HTTP Gets, HTTP Posts, or HTTP File Upload requests. Generally, these

requests are satisfied by CGI (Common Gateway Interface) programs that run on the server that receives the request. The requests have the following general characteristics.

ShakeCast requests are all associated with a particular URL path (or directory). The requests are therefore prefixed with the /sc URL path. For example, a request to the ShakeCast *heartbeat* request on the ShakeCast Server named mercator.gatekeeper.com would be accessed as:

"http://mercator.gatekeeper.com/sc/heartbeat"

A request for the Event XML for event number 12345 might use the URL:

"http://mercator.gatekeeper.com/sc/xml/event/12345.xml"

The above request might be answered by the Web Server, or by a script, depending on the configuration of your ShakeCast server.

2.2.1 Request Parameters

Some ShakeCast Requests pass parameters. These parameters are generally used by the receiving server to construct query predicates that limit the results returned. ShakeCast Request Parameters are passed as ordinary HTTP request parameters.

2.2.2 HTTP Headers

ShakeCast servers communicate via the HTTP protocol. All requests that are properly processed by the ShakeCast software will return an HTTP Version 1.1 status header that indicates success:

HTTP/1.1 200 OK

An HTTP status of anything other than "200 Success" indicates that there is something wrong with the ShakeCast software, the software environment on either the upstream or downstream server, or the network (such as a problem with an intervening HTTP Proxy Server).

2.2.3 ShakeCast Status XML

ShakeCast processors also return a HTTP payload to represent the status of the request processing. The payload is in the form of an XML message. This message is described in more detail in *Chapter 5, XML Documents*.

[more tbs]

2.2.4 XML URLs in ShakeCast Requests

Many ShakeCast requests pass information that the receiving server can use to obtain more information. This information is passed in the form of callback addresses, or callback URLs, that provide an address where further information may be retrieved in the form of an XML message.

For example, the `new_event` request optionally passes the receiving server a URL. This URL is the full address of an XML message that fully describes the event. When a ShakeCast server is notified of a new event with the `new_event` request, the server then takes value of the `event_XML` parameter and uses it to request additional information about the event.

An XML URL in the ShakeCast system is always a fully-formed URL. It must include a protocol (e.g., http, https, or ftp), server address (either as DNS or IP), and the entire path to the passed object. Note that the passed URL does not have to be on the same server that responded to the request. However, the downstream server may not be able to access the address provided unless the address is on a server known to the downstream server and the server at the given address will honor that request with respect to access controls or other network configuration issues.

Also, note that the provided URL may be either a file or a script that dynamically generates the XML message. Since returning files is often less resource intensive than running a script to generate XML out of the ShakeCast database, the ShakeCast server often will generate a file, store that file in the server's filesystem, and then pass the address of that file as the XML URL.

2.2.5 Other Returned Data in ShakeCast Requests

[tbs]

2.3 ShakeCast Request Parameter Formats

The ShakeCast system exchanges data as HTTP request parameters. These parameters include simple items such as unique identifiers and server names, but may also include more complex datatypes such as dates, latitudes and longitudes. These parameters are always expressed as text strings, but the format of these strings is limited to the formats described in the following paragraphs.

2.3.1 Time Usage and Date and Time Formats

Dates and times in ShakeCast parameters and XML files are always defined in the following format:

```
yyyy-mm-dd:hh:mm:ss.hhZ
```

where the values of the symbols in the format are defined in the following table.

Date and Time Formats

Item	Description
yyyy	Four digit year
mm	Two digit month, leading zeros required
dd	Two digit day-of-month, leading zeros required
hh	Two digit hour, leading zeros required
mm	Two digit minutes, leading zeros required
ss	Two digit seconds, leading zeros required
hh	Two digit hundredths of seconds.
Z	Indicates that the time is "Zulu" (Greenwich Mean Time, or GMT).

The entire year-month-day is always required in a ShakeCast date and time. If a time is provided, the entire time string must be provided. Zero-fill the time if necessary. If a time is not provided, do not include the colon following the day column, nor the trailing "Z".

Dates and times in ShakeCast are always in Universal Time Coordinates (UTC), also known as Greenwich Mean Time (GMT). Daylight saving adjustments are never applied directly to the time, but are instead shown as offsets from GMT.

All ShakeCast machines must be synchronized with the US National Time Standard using the Network Time Protocol (NTP). Under normal conditions, the time skew between any two ShakeCast machines will be no more than a few hundredths of a second. When a ShakeCast server responds to a request that includes time as a query predicate, the processing server will "fudge" the time backward. This will guarantee that even if there is clock skew between the two servers, the request will find records been written since the time given. The amount of "fudging" is specified in the ShakeCast server configuration file, and defaults to 0.5 seconds.

2.3.2 Latitude, Longitude, and Bounding Boxes

Latitudes and longitudes in ShakeCast are always represented as decimal degrees. West longitudes are negative. South latitudes are negative.

A point location is represented using a latitude and longitude expressed as the latitude value followed by a colon followed by the longitude value. There should be no spaces or other characters in the latitude/longitude string. For example:

```
34.147689:-118.128808
```

A spatial bounding box is represented using two latitude/longitude pairs separated by a two colons. Again,, no spaces or other characters are allowed. For example:

```
34.147689:-118.128808::34.335439:-118.506447
```

2.3.3 Server IDs and Server Names

ShakeCast Server IDs are globally unique and assigned by a root ShakeCast Server. These Server IDs are configured into the ShakeCast system at the time that a ShakeCast system is installed. The Server ID is retrieved automatically during the installation process using a simple HTTP request. Note that it is therefore required that a ShakeCast system have network access at the time that the software is installed.

[tbs]

2.4 Upstream and Downstream ShakeCast Machines

As in many other information transfer protocols, ShakeCast messages involve both a "server" and a "client". However, a single ShakeCast computer is often both a server and a client. Therefore, instead of labeling a ShakeCast system as either server or client, it is convenient to think of ShakeCast requests as moving data from an upstream machine to a downstream machine.

A downstream ShakeCast server may request information from an upstream server: these are "upstream requests". Alternately, an upstream ShakeCast server may send information, unsolicited, to a downstream machine: these are "downstream requests".

[more tbs]

2.5 Security and Authentication

[tbs]

2.5.1 Introduction to Security and Authentication

[tbs]

2.5.2 Authorized ShakeCast Servers

A ShakeCast server will allow communication only with other servers that are known to it. A server is "known" if it is defined in the SERVER table of the ShakeCast database. Servers in the ShakeCast database are further defined as "upstream", "downstream", or both, and the ShakeCast server may restrict communications based on the type of server.

Servers are defined in the SERVER table in three ways:

- When you initially install a ShakeCast server, you provide information to identify an upstream ShakeCast server that will deliver shaking data to your new server. The ShakeCast installation automatically registers your server with the upstream server, and enters the upstream server in your system's SERVER table.
- A ShakeCast administrator can add to the SERVER table manually, or access a ShakeCast web page that creates and registers a new server.
- Another ShakeCast server can request access to your server. Such a request results in an email notification to the ShakeCast system administrator, and the administrator can later use a URL provided in that link to define the new server in the SERVER table and grant the right to communicate with that server.

2.5.3 ShakeCast Server Logins

ShakeCast servers communicate with one another by logging in to the Web server. The login username is the ShakeCast Server ID. The login password is given in the PASSWORD column of the SERVER table for the table row that has the *target server's Server ID*. That is, the password in the Server Table on the local database is always the password used when logging in to the remote server.

2.5.4 Secure Communications

ShakeCast uses standard HTTP requests for all communication with other servers, and standard SMTP requests for all email and pager notification. In both cases, your network administrator can provide a higher level of security for these communication protocols.

HTTP requests that require higher security can use HTTP over Secure Sockets Layer, SSL. [more TBS]

Email notification in ShakeCast uses MIME (Multipart Internet Mail Extensions) to encode messages. If you require a higher level of security for your email notification, you can substitute Secure MIME (S/MIME) to encrypt these messages. [more tbs]

2.5.5 ShakeCast use of Public Key Infrastructure

[tbs]

2.5.6 ShakeCast Use of IP Addresses and DNS

ShakeCast servers can be configured in the ShakeCast database using either Domain Name System (DNS) addresses (readable names, like shakecast.usgs.gov) or Internet Protocol addresses (numeric addresses, like 10.2.3.4). Most Internet software systems tolerate both name formats, but system administrators tend to use only the DNS form of addresses because these are easier to maintain and can be changed by the target (named) system administrator without notifying all the systems that might communicate with that system.

However, the DNS system depends on the operation of a large number of systems on the Internet, and in the event of an earthquake the DNS system may not be stable, reliable, or responsive. Therefore, ShakeCast uses DNS but is designed to tolerate the failure of DNS.

When ShakeCast servers are identified by a DNS name, ShakeCast requests all use the DNS name instead of the IP address. In addition, ShakeCast requests will translate the DNS name and store the translated IP address in the ShakeCast database. Subsequently, if a ShakeCast request fails to obtain a DNS translation, then ShakeCast retries the request using the previously-stored IP address. In this way, the ShakeCast system provides insulation from transient failures in the DNS system.

All ShakeCast requests follow the above protocol. Most importantly, the `heartbeat` request runs periodically, and will store in the ShakeCast database the IP address that was used in the request.

Chapter 3 ShakeCast Requests

The following paragraphs document the available ShakeCast requests, together with their behavior, required and optional arguments, returned status values, and returned data.

Request	**get_event_list**
Summary	Request a list of events that meet a given set of criteria.
Required Parameters	none
Optional Parameters	begin_time=<ts>: limit the response to events that have occurred since ts record_time=<ts>: limit the response to events that have been recorded in the up-stream server since ts loc_bb=<bb>: limit the response to events that have a event point location within the bounding box specified by bb (see *Section 2.3.2 Latitude*, Longitude, and Bounding Boxes for more information on formatting bounding box predicates) extent_bb=<bb>: limit the response to events that have an event extent that crosses the bounding box specified by bb (see *Section 2.3.2 Latitude*, Longitude, and Bounding Boxes for more information on formatting bounding box predicates) no_metadata=true: if specified, the request returns XML with only event numbers. If this parameter is false or is not specified, the request returns XML with both the event numbers and all known event metadata.
Returned XML	ShakeCast Status XML: status of the request response Server Status XML: status of the responding server (optional) Event XML: An Event XML message containing zero or more events
Description	The `get_event_list` request retrieves a list of events from an upstream server. You may specify any or all of the optional parameters as arguments to the request. If specified, those parameters (query predicates) qualify the request to potentially reduce the list of returned events. The list of query predicates may be null, which is equivalent to a request for all events known by the upstream server. Such a request may be used to initialize an empty event database. The result returned in the payload of the HTTP get includes an XML file that contains the definitions of one or more events. If the `no_metadata` parameter is specified, the file contains XML that specifies only the event_ids that match your request. If the `no_metadata` parameter is not specified, the file also contains event metadata such as event time, the server ID of the originating ShakeCast server for this event, and so on. (See *Section* 5.2, Event XML in Chapter 5 for more information on the metadata defined for ShakeCast Events.) Note that if your ShakeCast Server communicates with multiple upstream servers, there is no guarantee that all the servers will know about all events, or about the order of notification from those servers, or about the coherency of the data from those servers. Note also that you may obtain different answers if you issue the same query multiple times, as the event database on the upstream server may be updated in the intervening time between your requests. Therefore, if you want to be sure that you have obtained a complete list of events known to the upstream server, you should use the `record_ts` parameter and specify the timestamp of the last event you received from that server, or the server's birth timestamp, whichever is earlier. The Event XML file is empty if the query predicates, when applied to the EVENT table in the ShakeCast database on the upstream server, result in zero events that

	match the criteria given. The `begin_time` predicate is applied using a greater-than-or-equal Boolean expression. The bounding box predicates are applied using an SQL BETWEEN predicate, which is greater-than-or-equal for the lower bound and less-than-or-equal for the upper bound, on each of the latitude and longitude axes.
Example	http://target-server/sc/request/get_event_list? record_time= 2003-01-7:22:12:34.56; extent_bb=34.147689:-118.128808::34.335439:-118.506447
Requesting Server Actions	If request succeeds, parse the ShakeCast Status XML for status information. Update the target server's LAST_HEARD_FROM timestamp and IP address, and zero the error count for that server. Log the request and status in the Exchange Log. Check to see if this event is new (not yet known). If it is new, create an EVENT record in the ShakeCast database. Optionally record the event information from the optional request parameters in the EVENT record. If the event type indicates this is a ShakeCast Test Event, follow the test protocol described in *Section 10.4, 10.4Processing Standard Test Messages*. The downstream server may decide whether to pursue further processing based on the information in the Event XML.. Based on the contents of the Event XML, the server may optionally perform event processing steps described in *Chapter 8, Exchange Processing*, or *Chapter 7, Notification Processing*, and may also request product information about the event using the `get_product` request. If the `get_event_list` request fails, increment the error counter for the requesting server.
Responding Server Actions	Use the provided query arguments as predicates in a query against the local server's EVENT table in the ShakeCast Database. Return the `event_ids` of the events that match those predicates. Form these event ids into an Event XML file. If the user did not specify the `no_metadata` parameter in the request, also retrieve all available data on the event and create and return a fully-populated event XML record for each event. Update the requesting server's LAST_HEARD_FROM timestamp and reset the error counter for that server. Log the request in the Exchange Log.

Request	**get_metadata_list**
Summary	Request system metadata that is known to the upstream server.
Required Parameters	table_name =<name>: The name of a ShakeCast Database table
Optional Parameters	no_metadata=true: If specified, the server will return just the identifiers of the database rows (primary key values), not the data rows themselves. A downstream server can use this to verify that it is up-to-date without actually applying any metadata changes.
Returned XML	ShakeCast Status XML: status of the request response Server Status XML: status of the responding server (optional) Metadata XML: A Metadata XML message containing the metadata for zero or more rows in the requested database table.
Description	The `get_metadata_list` request retrieves a list of known ShakeCast Database metadata from an upstream server. There are no query predicates. The result returned in the payload of the HTTP get is an XML file that contains zero or more rows of metadata from the upstream server. See the *Section 5.7, ShakeCast Metadata XML* in Chapter 5 for more information on how the XML file is formed. Note that if your ShakeCast Server communicates with multiple upstream servers, there is no guarantee that all the servers will have knowledge of all the metadata in the global ShakeCast network.
Example	http://target-server/sc/request/get_metadata_list?table_name=METRIC
Requesting Server Actions	If the request succeeds, parse the ShakeCast Status XML for status information. Update the target server's LAST_HEARD_FROM timestamp and IP address, and zero the error count for that server. Log the request and status in the Exchange Log. Parse the Metadata XML file. If no_metadata=true was specified, just compare the list of metadata Primary Keys with those found in the corresponding table in the local database. If new keys are found, the local server may make further requests to get the metadata, or it may log a notification with the local System Administrator. If no_metadata is not specified or false, compare the metadata retrieved with the metadata rows in the corresponding table in the local database, and update the local database to match. See *Section 8.3, Metadata Exchanges* for more information on updating local ShakeCast metadata. Check to see if this shaking metric is new (not yet known). If it is new, create a METRIC record in the ShakeCast database. Use the metadata describing the shaking metric from the `metric_XML`. Enter the metadata in the METRIC record in the local ShakeCast database. If the `get_metadata_list` request fails, increment the error counter for the requesting server.
Responding Server Actions	Update the requesting server's LAST_HEARD_FROM timestamp. Log the request in the Exchange Log.

Request	get_product_list
Summary	Request a list of products that meet a particular set of criteria.
Required Parameters	None
Optional Parameters	event_id=<eid>: limit the response to products that are associated with the following comma-separated list of one or more events record_time=<ts>: limit the response to products that have been created since the time specified by ts product_type=<type>: limit the response to products that match the types given. Specify multiple types as a comma-separated list. no_metadata=true: if specified, the request returns XML with only product ID numbers. If this parameter is not specified, the request returns XML with both the product ID numbers and all known product metadata.
Returned XML	ShakeCast Status XML: status of the request response Server Status XML: status of the responding server (optional) Product XML: A Product XML message containing zero or more events ShakeMap XML: A ShakeMap XML message that contains zero or more ShakeMap XML messages for the ShakeMaps that are associated with the Products returned
Description	The `get_product_list` request retrieves a list of products from an upstream server. You may specify any or all of the optional parameters as arguments to the request. If specified, those parameters (query predicates) qualify the request to potentially reduce the list of returned products. If multiple predicates are present in the argument list, the predicates `event` and `record_time` are logically ORed, so products matching any of the predicates will be returned. The `product_type` predicate is The result returned in the payload of the HTTP get is an XML file that contains the definitions of one or more products. If the no_metadata parameter is specified, the file contains XML that specifies only the product_ids that match your request. If the no_metadata parameter is not specified, the file also contains product metadata such as product creation time, the version number of this product, and so on. (See *Section* ***Error! Reference source not found., Error! Reference source not found.*** in Chapter 5 for more information on the metadata available for products.) Note that if your ShakeCast Server communicates with multiple upstream servers, there is no guarantee that all the servers will have the same products. The `begin_time` predicate is applied using a greater-than-or-equal Boolean expression.
Example	http://target-server/sc/request/new_server?server=1234; server_xml=http://myself.mycompany.com/sc/xml/server/4567.xml
Requesting Server Actions	If the request succeeds, parse the ShakeCast Status XML for status information. Update the target server's LAST_HEARD_FROM timestamp and IP address, and zero the error count for that server. Log the request and status in the Exchange Log. If the `get_product_list` request fails, increment the error counter for the re-

	questing server.
Responding Server Actions	Update the requesting server's LAST_HEARD_FROM timestamp. Check to see if this server is new (not yet known). If it is new, create a SERVER record in the ShakeCast database. Optionally, retrieve the metadata describing the product type from the `product_XML`. Enter the metadata in the PRODUCT record. A new server may not be activated by simply receiving a `new_server` request. After the server metadata is entered in the ShakeCast database, the receiving server will notify the local system administrator of the presence of the new server using the notification methods described in *Chapter 7, Notification Processing*. That notification request provides a URL for the system administrator to run a script which will authorize or enable the new server record for use. Log the request in the Exchange Log.

Request	**get_server_list**
Summary	Request a list of ShakeCast servers that meet a particular set of criteria.
Required Parameters	None
Optional Parameters	update_time=ts: return only servers that have had their server record updated since ts [tbs] how to get peers? How to get others who will service me?
Returned XML	ShakeCast Status XML: status of the request response Server Status XML: status of the responding server (optional) Server XML: An Server XML message containing the metadata for zero or more servers
Description	The `get_server_list` request retrieves a list of known ShakeCast servers from an upstream server. You may specify any or all of the optional parameters as arguments to the request. If specified, those parameters (query predicates) qualify the request to potentially reduce the list of returned servers. The result returned in the payload of the HTTP get is an XML file that contains the definitions of one or more servers. If the no_metadata parameter is specified, the file contains XML that specifies only the server_ids that match your request. If the no_metadata parameter is not specified, the file also contains server metadata such as server birth time, the status of the server, and so on. (See the Server XML in Chapter 5 for more information on the metadata available for servers.) Note that if your ShakeCast Server communicates with multiple upstream servers, there is no guarantee that all the servers will have knowledge of all the servers in the global ShakeCast network.
Example	http://target-server/sc/request/get_server_list?[tbs]
Requesting Server Actions	If the request succeeds, parse the ShakeCast Status XML for status information. Update the target server's LAST_HEARD_FROM timestamp and IP address, and zero the error count for that server. Log the request and status in the Exchange Log. If the `get_server_list` request fails, increment the error counter for the upstream server. Check to see if this server is new (not yet known). If it is new, create a SERVER record in the ShakeCast database. Optionally, retrieve the metadata describing the product type from the `server_XML`. Enter the metadata in the SERVER record. A new server is not activated by simply receiving a `get_server_list` request. After the server metadata is entered in the ShakeCast database, the receiving server will notify the local system administrator of the presence of the new server using the notification methods described in *Chapter 7, Notification Processing*. That notification request provides a URL for the system administrator to run a script which will authorize or enable the new server record for use.

Responding Server Actions	Update the requesting server's LAST_HEARD_FROM timestamp. Log the request in the Exchange Log.

Request	heartbeat
Summary	Test a ShakeCast connection and report the presence of the sending server to the receiving server.
Required Parameters	server_id=<sid>: the assigned server id of the requesting (initiating) server uptime=<delta-time>: the number of seconds the requesting server has been up continuously system_generation=<generation>: the number of times the requesting ShakeCast server has restarted since original installation error_count=<count>: the number of times any message between these two servers (heartbeat or other message type) has failed since the last success software_version=<version-string>: the version number of the ShakeCast software being run on the requesting server
Optional Parameters	record_time=<ts>: <ts> is a full date and time in the ShakeCast time format system_xml=<URL>: <URL> will respond to HTTP Get requests with Server Status XML about the requesting server (the one generating the heartbeat request, not the server responding to the heartbeat request)
Returned XML	ShakeCast Status XML: a standard XML message containing status information ShakeCast Server Status XML: an XML message describing the status of the responding server
Description	The heartbeat request notifies receiver of the presence of the sender. It also allows the sender and receiver to verify that most of the ShakeCast software, network and hardware infrastructure is operating normally. Nominally, the error_count parameter will be zero, indicating that the requesting server has not failed to obtain heartbeat. A non-zero error_count indicates that the communication network between the two servers is not stable. When a sender makes a heartbeat request, it may optionally include a timestamp parameter. The receiver may use this timestamp to verify that the clocks of the sender and receiver are synchronized.
Example	http://target-server/sc/request/heartbeat?server_id=1234;uptime=86401; system_generation=123;error_count=0;software_version=V1.0.0
Requesting Server Actions	If request succeeds, parse the ShakeCast Status XML for status information. Update the target server's LAST_HEARD_FROM timestamp and IP address, and zero the error count for that server. Log the request and status in the Exchange Log. If the request fails, increment the error counter for the requesting server.
Responding Server Actions	Update the requesting server's LAST_HEARD_FROM timestamp. Optionally record the uptime, system generation, error count, and software version data in the requesting server's record in the SERVER table of the ShakeCast database. Log the request in the Exchange Log.

Request	**new_event**
Summary	Notify a server of a new ShakeCast event.
Required Parameters	None
Optional Parameters	None
Posted XML	Event XML
Returned XML	ShakeCast Status XML: status of the request response Server Status XML: status of the responding server (optional)
Description	The new_event request notifies a ShakeCast server of a new ShakeMap seismic event. The receiver will process the Event XML to examine the parameters of the event. The receiver may then ignore the event, or the receiver may enter the event in its database.
Example	http://target-server/sc/request/new_event
Requesting Server Actions	If request succeeds, parse the ShakeCast Status XML for status information. Update the target server's LAST_HEARD_FROM timestamp and IP address, and zero the error count for that server. Log the request and status in the Exchange Log. If the request fails, increment the error counter for the requesting server.
Responding Server Actions	Update the requesting server's LAST_HEARD_FROM timestamp. Process the Event XML. Check to see if this event is new (not yet known). If it is new, create an EVENT record in the ShakeCast database. If the event type indicates this is a ShakeCast Test Event, follow the test protocol described in *Section 10.4, Processing Standard Test Messages*. The downstream server may decide whether to pursue further processing based on the information in the event XML. Based on the contents of that XML message, the server may optionally perform event processing steps described in *Chapter 8, Exchange Processing*, or *Chapter 7, Notification Processing*. Log the request in the Exchange Log.

Request	**new_metadata**
Summary	Notify a server of new or changed ShakeCast metadata.
Required Parameters	None
Optional Parameters	None
Posted XML	Metadata XML: information to update the ShakeCast database
Returned XML	ShakeCast Status XML: status of the request response Server Status XML: status of the responding server (optional)
Description	The new_metadata request notifies a ShakeCast server of new or updated ShakeMap metadata. The notice may be ignored by the receiver, or the receiver may enter the metadata in its database. Generally, the receiver will enter the metadata update into the ShakeCast database and then issue an HTTP get request to the metadata_url URL to retrieve the XML file.
Example	http://target-server/sc/request/new_metadata
Requesting Server Actions	If request succeeds, parse the ShakeCast Status XML for status information. Update the target server's LAST_HEARD_FROM timestamp and IP address, and zero the error count for that server. Log the request and status in the Exchange Log. If the request fails, increment the error counter for the requesting server.
Responding Server Actions	Update the requesting server's LAST_HEARD_FROM timestamp. Retrieve and process the Metadata XML file, as described in *Section 8.3, Metadata Exchanges*. Log the request in the Exchange Log.

Request	**new_product**
Summary	Notify a server of a new ShakeCast product.
Required Parameters	None
Optional Parameters	None
Posted XML	Product XML: information about one or more new ShakeCast Products available
Returned XML	ShakeCast Status XML: status of the request response Server Status XML: status of the responding server (optional)
Description	The new_product request notifies a ShakeCast server of a new ShakeMap product. The notice may be ignored by the receiver, or the receiver may enter the product in its database. Generally, the receiver will enter the product definition into the ShakeCast database and then issue an HTTP get request to the product_xml to retrieve the product XML file and/or issue an HTTP get request to the product_url to get the actual product file itself.
Example	http://target-server/sc/request/new_product
Requesting Server Actions	If request succeeds, parse the ShakeCast Status XML for status information. Update the target server's LAST_HEARD_FROM timestamp and IP address, and zero the error count for that server. Log the request and status in the Exchange Log. If the request fails, increment the error counter for the requesting server.
Responding Server Actions	Update the requesting server's LAST_HEARD_FROM timestamp. Parse the Product XML message. Check to see if this product is new (not yet known). If it is new, create a PRODUCT record in the ShakeCast database. Optionally record the product information from the optional request parameters in the PRODUCT record. Check to see if this ShakeMap is new (not yet known). If it is new, create a SHAKEMAP record in the ShakeCast database. Optionally, retrieve the metadata describing the product from the `product_XML`. Enter the metadata in the PRODUCT record and other records in the ShakeCast Database. Optionally, retrieve the product file itself from the `product_URL`. Compare the size of the file (in bytes) and the MD5 hash with the parameters provided in the original request or in the product XML message. Log the request in the Exchange Log.

Request	**new_server**
Summary	Notify a server of a new ShakeCast server.
Required Parameters	None
Optional Parameters	None
Posted XML	Server XML: information about the new ShakeCast Server
Returned XML	ShakeCast Status XML: status of the request response Server Status XML: status of the responding server (optional)
Description	The `new_server` request notifies a ShakeCast server of a new ShakeMap server that has become known to the upstream server. The notice may be ignored by the receiver, or the receiver may enter the new server record in its database. Generally, the receiver will enter the server definition into the ShakeCast database and then issue an HTTP get request to the `server_xml` to retrieve the server XML file and store the parameters found there in the SERVER table.
Example	http://target-server/sc/request/new_server
Requesting Server Actions	If the request succeeds, parse the ShakeCast Status XML for status information. Update the target server's LAST_HEARD_FROM timestamp and IP address, and zero the error count for that server. Log the request and status in the Exchange Log. If the `new_server` request fails, increment the error counter for the requesting server.
Responding Server Actions	Update the requesting server's LAST_HEARD_FROM timestamp. Check to see if this server is new (not yet known). If it is new, create a SERVER record in the ShakeCast database. Optionally, retrieve the metadata describing the product type from the `server_XML`. Enter the metadata in the SERVER record. A new server may not be activated by simply receiving a `new_server` request. After the server metadata is entered in the ShakeCast database, the receiving server will notify the local system administrator of the presence of the new server using the notification methods described in *Chapter 7, Notification Processing*. That notification request provides a URL for the system administrator to run a script which will authorize or enable the new server record for use. Log the request in the Exchange Log.

Request	**register**
Summary	Register a ShakeCast server with an upstream server.
Required Parameters	name=<fully qualified host name>: a text string in the form of a fully quality host name (host name plus domain name)
Optional Parameters	None
Posted XML	Server XML: information about the server to be registered
Returned XML	ShakeCast Status XML: status of the request response Server Status XML: status of the responding server (optional) Server XML: An Server XML message containing the server ID of the newly registered server.
Description	The `register` request informs an upstream server about the presence of a downstream server. The upstream server will respond by computing a new globally unique Server ID and inserting a record for this server into the SERVER table of the upstream server's ShakeCast Database. [TBS – how to compute a global unique ID] Hosts must be named uniquely. The best way to do this is to use a host name qualified by your organization's DNS domain name. For example, if your organization is the Western Region of the United States Geological Survey, you might name your server as shakecast1.wr.usgs.gov. Your ShakeCast server name does not need to match a particular machine name. For example, the ShakeCast server named above might run on a machine called server23.wr.usgs.gov or mars.caltech.edu. Tthe upstream server will parse the posted ShakeCast Server XML file. The parameters found in that XML file will be added to the newly created record in the SERVER table, except that the Server ID may not be overwritten with values from the XML. The SERVER XML with the Server ID completed is then returned as the payload of the HTTP Get.
Example	http://target-server/sc/request/register?name=roll.gatekeeper.com
Requesting Server Actions	If the request succeeds, parse the ShakeCast Status XML for status information. Update the target server's LAST_HEARD_FROM timestamp and IP address, and zero the error count for that server. Log the request and status in the Exchange Log. If the `register` request fails, increment the error counter for the requesting server.
Responding Server Actions	Update the requesting server's LAST_HEARD_FROM timestamp. Log the request in the Exchange Log. [tbs – register the server]

Chapter 4 ShakeCast Interactions

ShakeCast messages may be sent one-at-a-time, or in combination. A number of combinations of requests are commonly used, and we term these "ShakeCast Interactions". This Section of the ShakeCast Specification documents these Interactions.

4.1 Poll and Request Interactions

In Poll and Request Interactions, a downstream ShakeCast Server periodically checks for new data on an upstream server. When the upstream server has no data available, the downstream server waits and tries again. If the upstream server has data, the downstream server requests additional objects.

4.1.1 Poll for Event and Get Resulting Products

In this interaction, a downstream server polls one or more upstream servers for new events. If a new event is returned, the get_event_list request also returns Event XML with the details of each event. This interaction is used by a downstream server that cannot receive a "push" of new events and event data from an upstream server.

Poll for Event and Get Resulting Products	
Request	**Description**
get_event_list? begin_time=<timestamp>; no_metadata=false	Request a list of new events since the last poll time. Return the event metadata in the body of the response. Update the local database with the list of new events and with the metadata about each event.
get_product_list?eid=<event_id>	Request a list of products available for this event. The product metadata is returned in the request. Included in the metadata is the URL of each product file. Loop over each event returned in the previous request.
HTTP get	Loop over each product for each event to get the actual product files.
Notification Loop	After all products have been received, loop through the data-base looking for notifications to be queued (see Chapter 7).

4.1.2 Poll for Product Update

If a server has recently processed ShakeCast products, it may periodically poll all upstream servers to check if any of those products have been updated or any new products added. Even if there is not a new event, new products may be created or existing products may be updated.

Poll for All Product Updates Get Resulting Products	
Request	**Description**
get_product_list? record_time=<timestamp>; no_metadata=false	Request a list of new products since the last poll time. Return the product metadata in the body of the response. Update the local database with the metadata returned.
HTTP get	Loop to get each product file.
Notification Loop	After all products have been received, loop through the database looking for notifications to be queued (see Chapter 7).

Alternately, the server may search for only those products on events that have occurred in some small recent time window. To do this, the server would build a list of events that have recently occurred or been updated, and then search for new products only on those recent events.

Poll for Product Updates on Recent Events and Get Resulting Products	
Request	**Description**
get_event_list? begin_time=<timestamp>; no_metadata=false	Request a list of new events since the last poll time. Return the event metadata in the body of the response. Update the local database with the returned metadata.
get_product_list? event_id=<eid_list>; no_metadata=false	Request a list of new products based on the event ID list returned in the previous request. Return the product metadata in the body of the response. Update the local database.
HTTP get	Loop to get each product file.
Notification Loop	After all products have been received, loop through the database looking for notifications to be queued (see Chapter 7).

4.1.3 Poll for Server and Metadata Updates

The ShakeCast Server is designed to allow downstream servers to "discover" new servers and new types of products. These may be "pushed" from an upstream server, or a downstream server may periodically poll for these updates. The following exchange shows a downstream server checking for new servers that can provide a data feed and.

Poll for All New Servers	
Request	**Description**
get_server_list?	Request a list of updated servers since the last poll time. Return the

update_time=<timestamp>; no_metadata=false	server metadata in the body of the response. Update the local database.
Notification Loop	Notify the system administrator if any of the new servers offer a data feed (based on permissions). Optionally, configure these new servers to feed data to this server (see Chapter 7).

The following exchange shows a downstream server polling for new product types. These new product types are entered into the local database, and become available to users to request and use in notification activity.

Poll for New Product Formats	
Request	**Description**
get_metadata	Request a list of updated product formats since the last poll time, using the default parameters in the HTTP Get request. Return the product format metadata in the body of the response. Update the local database with the new formats, if any.
Notification Loop	Notify the system administrator of any updated product formats (see Chapter 7).

4.1.4 Provide Heartbeat

ShakeCast servers can inform one another of their operating status by sending heartbeat messages. These messages include parameters that inform the receiving server of the general health and status of the requesting server.

Notify the Server of Our Status	
Request	**Description**
heartbeat?server_id=<sid>; uptime=<delta-time>; system_generation=<generation>; error_count=<count>; software_version=<version-string>	Notify the receiving server of the status of the requesting server. Request returns the ShakeCast software version running on the receiving server. The receiving server will update the SERVER table LAST_HEARD_FROM column and other columns as appropriate, depending on which optional parameters are passed.

4.2 Server Push Interactions

ShakeCast servers may send data to one another in an unsolicited fashion. This type of interaction is called “Push”. Push interactions may be single-step or multi-step.

In a single-step push interaction, all the information needed by the downstream server is sent in a single request.

Remember that a ShakeCast request may be composed of multiple HTTP requests in both directions. In a multi-step push interaction, the first interaction is an unsolicited push from an upstream server to notify the downstream server of new information. The remainder of the inter-

action consists of traditional "pull" or "get" operations, wherein the downstream server uses the new information provided in the push step to make targeted requests. Note that in a multi-step push, the Get operations may be executed against a different server than the one that generated the push (and, in practice, this will frequently be the case).

4.2.1 New Event Push from ShakeMap Server

The entire cascade of ShakeMap events is begun when a ShakeMap Server (not a ShakeCast Server) has detected a new significant event. A ShakeMap Server may send along every event it is notified about, but a more useful behavior will be to only send those events that have triggered the production of a ShakeMap.

In the following request, a ShakeMap Server simply sends the most basic of event information to one or more downstream ShakeCast Servers.

Notify a Server of a New Event (Only)	
Request	**Description**
new_event?event_id=<eid>; event_ts=<ts>	Notify the receiving server that a new event is known to the requesting server. The receiving server should enter the new event ID in the ShakeCast database, and may choose to send this information along to downstream servers. Note that the `event_xml` argument is not provided in the request, as a typical ShakeMap server does not respond to HTTP get requests, but only sends information unsolicited.

4.2.2 New Event Push from ShakeCast Server

In the following interaction, a ShakeCast Server is notified by another ShakeCast Server of a new event. This interaction might be used, for example, between two servers who believe they are both authoritative and complete for the list of events. If the receiving server is notified of a new event that it does not already "know about", then this is an indication that the ShakeCast network may have become partitioned. All of the information needed is exchanged in a single HTTP request.

Notify a Server of a New Event (Only)	
Request	**Description**
new_event?event_id=<eid>; event_xml=<URL>	Notify the receiving server that a new event is known to the requesting server. The receiving server should enter the new event ID in the ShakeCast database, and may choose to request further information about the event using the `event_XML` parameter.
New event notification	The receiving server will perform the New Event Notification processing steps described in *Chapter 7, Notification Processing*.

4.2.3 New Event and Product Push from ShakeCast Server

In the following interaction, more information is sent to the receiving server. In this case, the server is also provided a URL where additional information is available. If the receiving server is a downstream server, it will then use this URL to obtain detailed information.

Notify a Server of a New Event and New Products	
Request	**Description**
new_event?event_id=<eid>; event_xml=<URL>; event_location=<lat:lon>	Notify the downstream server that a new event is known to the upstream server. The downstream server should enter the new event ID in the ShakeCast database, and may choose to request further information about the event.
HTTP Get <event XML URL>	Based on the event location in the original `new_event` request, the downstream server decides it wants to know more about the event for which a notification was just received. All of this information is available by retrieving the Event XML. The server does so using an ordinary HTTP get, and then loads that information into the local database.
New event loop	Perform the processing steps required when an event is added or updated.
get_product_list?event_id=<eid>	Based on the information in the Event XML, the downstream server decides it needs to retrieve all available products for this event. The product XML is requested, but this request will generally fail because in most cases when a new event is created there are no products yet available.
Wait loop	Wait a pre-determined amount of time before checking again for new products for this event. Loop over failures a per-determined number of times.
new_product?product_id=<pid>; product_xml=<URL>; product_url=<URL>; size=<bytes>	While the downstream server is waiting to re-issue the failed get_product_list request, the upstream server pushes a new product downstream.
New product loop	The downstream server then performs the processing steps required when a product is added or updated.
HTTP Get product file	Retrieve the product file and product XML.
HTTP Get product XML	
New product loop	After receiving a new product file, perform the New Product Loop. When the file has been received, outstanding requests in the wait loop are canceled.
New product notification loop	Perform the notification steps associated with a new product.

4.2.4 More Interactions

[tbs]

4.3 Registration Interactions

ShakeCast servers may register with one another. They do so by making a request of the other server, and then offering a URL where the other server may obtain additional information in the form of a Server XML file.

4.3.1 Register New Downstream Server

The most common registration interaction occurs when a new server is brought online. As part of the server configuration process, the new server finds and registers with one or more upstream servers who can provide it with a feed of ShakeCast broadcasts.

Request	Description
register?name=<fqhn>; server_xml=<URL>	Request that the receiving server (now called the registration server) add the new server (now called the registering server) to the SERVER table. If the new server is successfully added, the receiving server will return a Server XML message containing a unique SERVER_ID to be used in subsequent interactions.
HTTP get <URL>	The registration server now turns around and executes an HTTP get against the URL provided in the register request. This URL should return a complete Server XML file. The registration will add the information in this XML file to the SERVER table.
heartbeat?server_id=<sid>; uptime=<delta-time>; system_generation=<generation>; error_count=<count>; software_version=<version-string>	Tell the upstream server hello.

4.3.2 Update Metadata for Existing Downstream Server

[tbs]

Request	Description

4.3.3 Request Complete Metadata Update

[tbs]

Request	**Description**

Chapter 5 XML Documents

Extensible Markup Language (known by the acronym XML) is a widely used and easily implemented method of exchanging data between disparate computer systems. The ShakeCast System uses XML to communicate all kinds of information between ShakeCast servers:

- Data about ShakeCast Servers and the ShakeCast software itself
- Data about events (earthquakes) and products (data files) available on the network
- Status information that helps the administrators of ShakeCast servers tell if their network is running smoothly

This Section documents the ShakeCast XML file formats.

5.1 Introduction to ShakeCast XML

[tbs]

5.1.1 [tbs]

[tbs]

5.2 Event XML

A ShakeCast Event is described by Event XML. A sample Event XML is shown in the following figure.

```
<event
    event_id="1234" event_version="2"
    event_status="ACTUAL"
    event_type="EARTHQUAKE"
    event_name="Northridge"
    event_location_description="1.2mi SSW of Northridge, CA"
    event_timestamp="1994-05-07 14:34:23" external_event_id="C123V4"
    magnitude="6.7" latitude="34.2345222" longitude="-118.123222"
/>
```

The Event XML is defined by the following XML Schema.

[tbs]

5.3 Product XML

A ShakeCast Product is described by Product XML. A sample Product XML is shown in the following figure.

```
<product
     product_type="TVMAP.PNG"
     product_status="RELEASED"
     generating_server_id="1"
     generation_timestamp="1994-05-07 14:34:23.00Z"
     bounding_box="34.2345678:-118.1234567::34.2345678:-
     118.1234567"
     product_id="1234" product_version="1"
     shakemap_id="1234" shakemap_version="1"
/>
```

The Product XML is defined by the following XML Schema.

[tbs]

5.4 ShakeMap XML

A ShakeCast ShakeMap is described by ShakeMap XML. A sample ShakeMap XML is shown in the following figure.

```
<shakemap
     shakemap_id="1234" shakemap_version="1"
     event_id="1234" event_version="1"
     generation_timestamp="1994-05-07 14:34:23.00Z"
     begin_timestamp="1994-05-07 14:29:05.01Z"
     end_timestamp="1994-05-07 14:30:01.10Z"
     bounding_box="34.234567:-118.123456::34.234567:-118.1234567"
    <metric metric_name="PSA03" min_value="0.0" max_value="70.1" />
    <metric metric_name="PSA10" min_value="0.0" max_value="70.1" />
    <metric metric_name="PSA30" min_value="0.0" max_value="70.1" />
    <metric metric_name="PGV" min_value="0.0" max_value="70.1" />
    <metric metric_name="PGA" min_value="0.0" max_value="70.1" />
  </shakemap>
```

The ShakeMap XML is defined by the following XML Schema.

[tbs]

5.5 Server Status XML

[tbs]

5.6 ShakeCast Status XML

[tbs]

5.7 ShakeCast Metadata XML

[tbs]

Chapter 6 The ShakeCast Database

ShakeCast servers store much of the data used by the server in a relational database. The ShakeCast database contains information needed to interact with other ShakeCast Servers, data that will be presented to users, configuration information needed to perform notifications, and various other kinds of data.

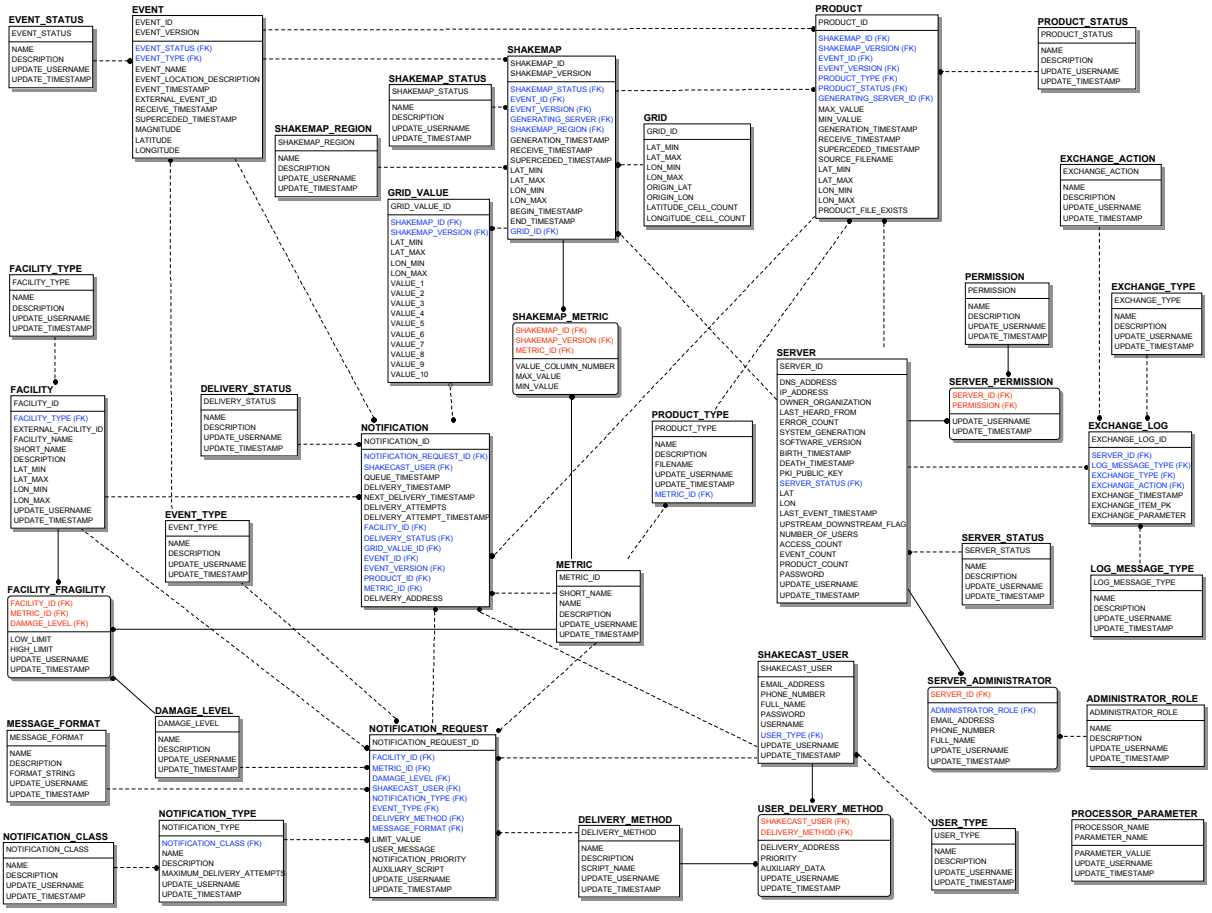
The following paragraphs document the structure of the ShakeCast database using a standard entity-relationship modeling syntax.

6.1 Introduction

The ShakeCast database is used in all elements of the ShakeCast system. Although some tables are used in many different stages of ShakeCast operation, the tables may be conveniently grouped as follows:

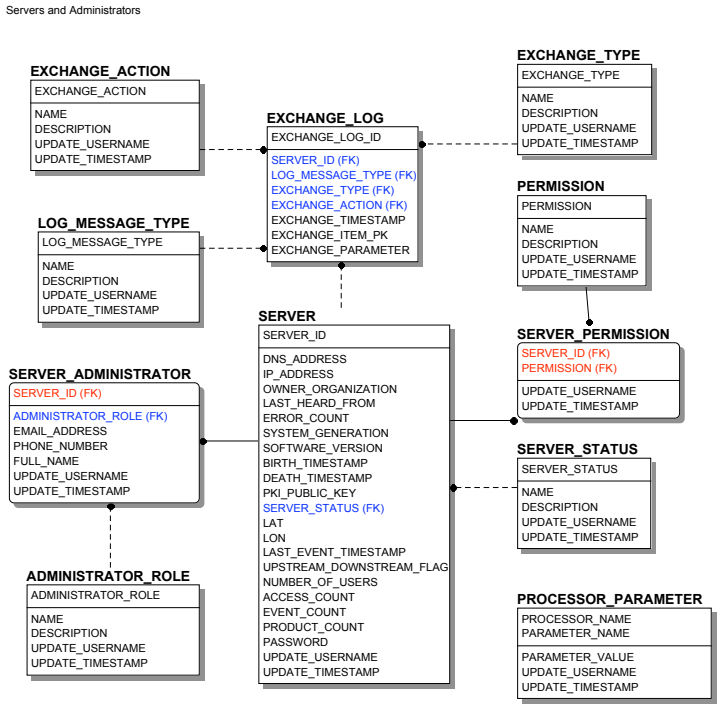
- Tables that define the ShakeCast system, the network of ShakeCast servers, exchanges between those servers, and the administrators who maintain that network
- Tables that define ShakeCast events (earthquakes), products, and related data
- Tables that are used to notify end users of earthquakes, shaking levels at particular facilities, and other ShakeCast events
- Miscellaneous operational tables

Each of these groups of tables is documented in the sections below.



6.2 Database Tables for Servers and System Administrators

The first group of tables defines ShakeCast servers and the organizations and users responsible for those servers. These tables also define the rights that each server has to communicate with the current server.



6.2.1 SERVER

The SERVER table contains information about every ShakeCast server that this server communicates with and every server that creates data used by this server.

Column	Datatype	Description
SERVER_ID	Integer	A globally unique identifier for the server
DNS_ADDRESS	String	DNS address (name) of the server
IP_ADDRESS	String	IP address (numeric address) of the server
LAST_HEARD_FROM	Datetime	The date and time (GMT) that the server was last heard from by this server
ERROR_COUNT	Integer	The number of errors recorded since the last time a successful exchange took place with this server.
SYSTEM_GENERATION	Integer	The number of times that this server has restarted the ShakeCast software. This value is incremented each time a server is restarted.
SOFTWARE_VERSION	String	The version string for the ShakeCast software this server is currently running.
OWNER_ORGANIZATION	String	The name of the organization that operates this server

		server
BIRTH_TIMESTAMP	Datetime	The earliest known date of operation of the server
DEATH_TIMESTAMP	Datetime	The last known date of operation of the server
STATUS	Integer	The current operational status of the server
LAT	Float	Latitude of this server
LON	Float	Longitude of this server
LAST_EVENT_TS	Datetime	The date and time of the last update to the event table on this server.
UPSTREAM_DOWNSTREAM_FLAG	String	The value "U" if the server in this record is upstream relative to this server. The value "D" if the server in this record is downstream relative to the local server. The value "B" if the server is both upstream and downstream. The value "S" if this is the server's own (self) record. The value "A" if this server is an authoritative source of data. Null if the relationship to this server is not known or undefined.
PKI_PUBLIC_KEY	String	The public key associated with this server.
NUMBER_OF_USERS	Integer	The number of unique user IDs that have access to ShakeCast data.
ACCESS_COUNT	Integer	The number of user accesses to ShakeCast data and reports.
EVENT_COUNT	Integer	The number of ShakeCast events stored in the ShakeCast server.
PRODUCT_COUNT	Integer	The number of ShakeCast products stored in the ShakeCast server.
PASSWORD	String	The password used by the local server to log into this remote server.
UPDATE_TIMESTAMP	Date	Last time this record was changed.
UPDATE_USERNAME	String	Local database user who performed the last update.

6.2.2 SERVER_STATUS

The SERVER_STATUS table defines the valid codes for the STATUS column of the SERVER table.

Column	Datatype	Description
SERVER_STATUS	String	Type code for server status
NAME	String	Status name
DESCRIPTION	String	Human readable meaning for this value

6.2.3 PERMISSION

The PERMISSION table defines the valid codes for the PERMISSION column of the SERVER table.

Column	Datatype	Description
PERMISSION	String	Type code for server permission
NAME	String	Status name
DESCRIPTION	String	Human readable meaning for this value

6.2.4 SERVER_PERMISSION

The SERVER_PERMISSION table defines which activities are valid when communicating with the each server in the ShakeCast network.

Column	Datatype	Description
SERVER_ID	Integer	A globally unique identifier for the server, and foreign key into the SERVER table.
PERMISSION	Integer	Type code for operations permitted in interactions between the local server and the server indicated.

6.2.5 SERVER_ADMINISTRATOR

The SERVER_ADMINISTRATOR table contains information about the people who administer ShakeCast servers. In most cases, a particular individual will be associated with only a single ShakeCast server.

Column	Datatype	Description
SERVER_ID	Integer	The globally unique ID for the server with which this administrator is associated.
ADMINISTRATOR_ROLE	Integer	The role of this individual for this server
EMAIL_ADDRESS	String	The email address of the individual
PHONE_NUMBER	String	The full phone number of the individual
FULL_NAME	String	The administrator's full name
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

6.2.6 ADMINISTRATOR_ROLE

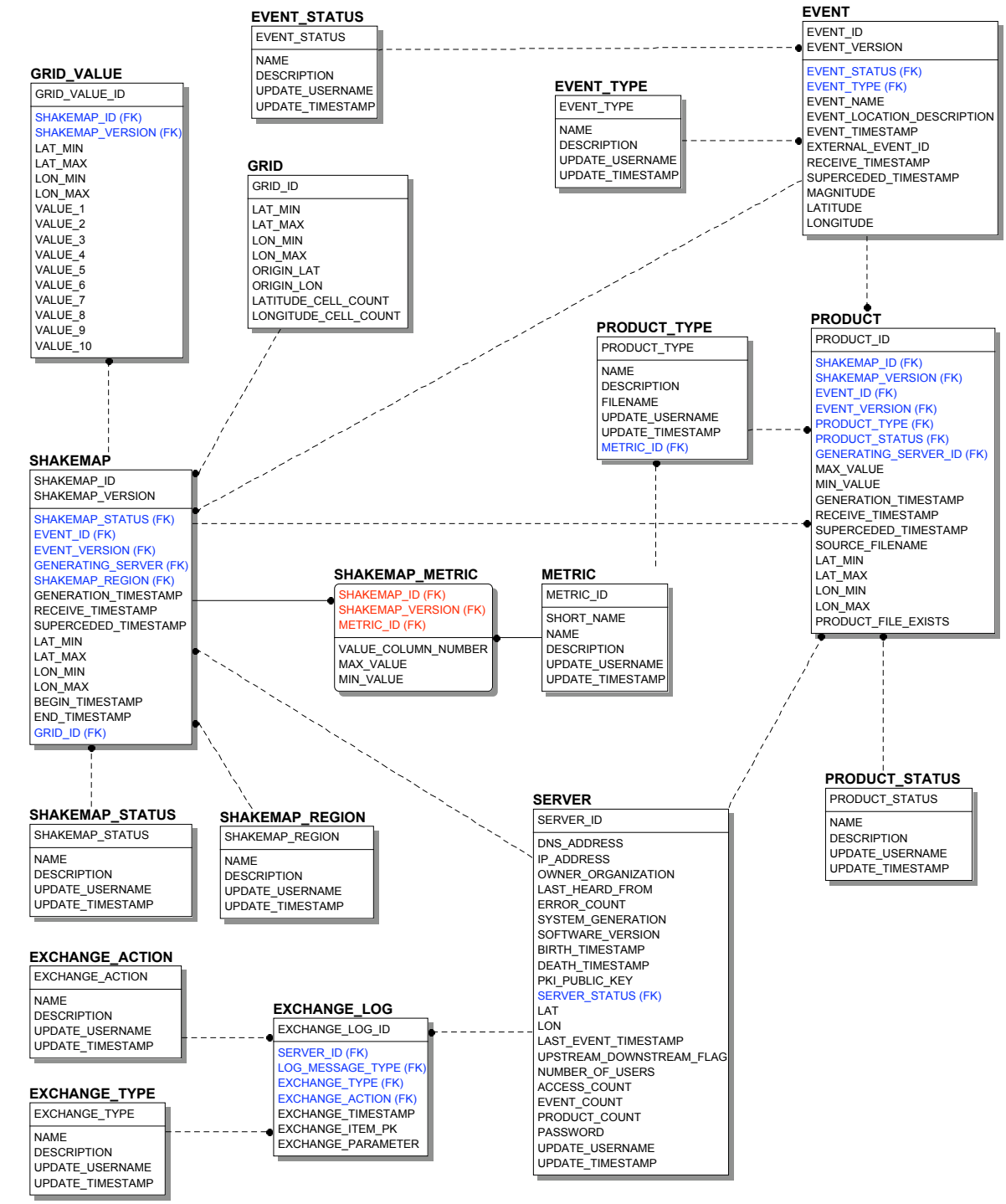
The ADMINISTRATOR_ROLE table defines the valid roles for administrators.

Column	Datatype	Description
ADMNISTRATOR_ROLE	Integer	Type code for the administrator role
NAME	String	Name of role
SHORT_NAME	String	Abbreviation for role name
DESCRIPTION	String	Human readable meaning for this value

6.3 Database Tables for Events, ShakeMaps, and Products

The following tables contain the data needed to represent Events, ShakeMaps, and ShakeMap Products.

Events and Products



6.3.1 EVENT

The EVENT table contains information about seismic events. The EVENT_ID is a globally unique, permanently assigned identifier associated with a single seismic event.

Column	Datatype	Description
EVENT_ID	Integer	A globally unique identifier for the event
EVENT_VERSION	Integer	A sequential version number. The latest version is the most current representation of the data about this event.
EVENT_TYPE	Integer	Foreign key to the EVENT_TYPE table
EVENT_TIMESTAMP	Datetime	The date and time (GMT) that the event occurred
EXTERNAL_EVENT_ID	String	Event ID in a locally-defined external server
EVENT_STATUS	Integer	The status of this event (active, cancelled, test, archive, etc.)
EVENT_NAME	String	Name of event (i.e., “Northridge”)
EVENT_LOCATION_DESCRIPTION	String	Human-readable location description (i.e., “8.1mi of Pasadena, CA”)
RECEIVE_TIMESTAMP	Datetime	The time this event information was first received on this server
LAT	Float	The latitude of the point representation of the event
LON	Float	The longitude of the point representation of the event
SUPERCEDED_TIMESTAMP	Datetime	The time this event was superceded by a newer version or by a different event.

6.3.2 EVENT_STATUS

The EVENT_STATUS table defines the valid codes for the STATUS column of the EVENT table. The event status is typically one of the following: active, cancelled, test, archive, or unknown.

Column	Datatype	Description
EVENT_STATUS	String	Type code for event status
NAME	String	Status name
DESCRIPTION	String	Human readable meaning for this value

6.3.3 EVENT_TYPE

The EVENT_TYPE table defines the valid codes for the EVENT_TYPE column of the EVENT table. The event type is typically one of the following: system test, local test, or active.

Column	Datatype	Description

EVENT_TYPE	Integer	Code for the event type
NAME	String	Event type name
DESCRIPTION	String	Human readable meaning for this value

6.3.4 SHAKEMAP

The SHAKEMAP table describes a single ShakeMap. A ShakeMap is produced outside of the ShakeCast system, by a ShakeMap Server. ShakeMaps are associated with zero or more events and with zero or more Products.

Column	Datatype	Description
SHAKEMAP_ID	Integer	Uniquely defines a ShakeMap.
VERSION	Integer	ShakeMap versions start with one and are increased each time the ShakeMap is updated. Only the latest version of a ShakeMap is correct. The primary key of this table is the SHAKEMAP_ID plus VERSION..
GENERATION_TIMESTAMP	Datetime	The time this ShakeMap was first created
SHAKEMAP_REGION	String	The ShakeMap region that generated this ShakeMap
GENERATING_SERVER	Integer	The unique ID of a ShakeCast server
SHAKEMAP_STATUS	Integer	The status of this ShakeMap (active, cancelled, test, archive, etc.)
RECEIVE_TIMESTAMP	Datetime	The timestamp of the last time this ShakeMap was received from an upstream server
BEGIN_TIMESTAMP	Datetime	The beginning date and time of the period covered by this ShakeMap.
END_TIMESTAMP	Datetime	The ending date and time of the period covered by this ShakeMap.
LAT_MIN	Float	Bounding box of the area covered by this ShakeMap
LAT_MAX	Float	
LON_MIN	Float	
LON_MAX	Float	
SUPERCEDED_TIMESTAMP	Datetime	Time this ShakeMap was superceded by a new version or by a different ShakeMap.
GRID_ID	Integer	Foreign key to the Grid Table, which defines the bounding box and cell size of the grid that applies to this ShakeMap

6.3.5 SHAKEMAP_STATUS

The SHAKEMAP_STATUS table defines the valid codes for the STATUS column of the SHAKEMAP table. The ShakeMap status is typically one of the following: active, cancelled, test, archive, or unknown.

Column	Datatype	Description
SHAKEMAP_STATUS	String	Type code for ShakeMap status
NAME	String	Status name
DESCRIPTION	String	Human readable meaning for this value

6.3.6 SHAKEMAP_REGION

The SHAKEMAP_REGION table defines the ShakeMap Regions. Current regions include Southern California, Northern California, and Utah.

Column	Datatype	Description
SHAKEMAP_REGION	String	Type code for ShakeMap Region
NAME	String	Region Name
DESCRIPTION	String	Human readable meaning for this value

6.3.7 PRODUCT

This table contains information about each ShakeMap product. A product is a single metric (PRODUCT_METRIC column) for a single ShakeMap (SHAKEMAP_ID column) in a single format (PRODUCT_FORMAT column).

Column	Datatype	Description
PRODUCT_ID	Integer	A globally unique identifier for this product.
VERSION	Integer	Product versions start with one and are increased each time the product is updated. Only the latest version of a product is valid and correct. The primary key of this table is the PRODUCT_ID plus VERSION.
PRODUCT_STATUS	Integer	The status of this product (active, cancelled, test, archive, etc.)
GENERATION_TIMESTAMP	Datetime	The time this ShakeMap was first created.
UPDATE_TIMESTAMP	Datetime	The last time this table or the ShakeMap itself was up-dated.
GENERATING_SERVER	Integer	The unique ID of a ShakeCast server

METRIC	Integer	The shaking metric represented in this product, such as “acceleration”, “instrumental intensity”, etc.
MAX_VALUE	Integer	The maximum value for METRIC contained within this product.
MIN_VALUE	Integer	The minimum value for METRIC contained within this product.
RECEIVE_TIMESTAMP	Datetime	The time this product information was first received on this server
UPDATE_TIMESTAMP	Datetime	The time this product information was last updated
LAT_MIN	Float	Bounding box of the area covered by this product
LAT_MAX	Float	
LON_MIN	Float	
LON_MAX	Float	
GRID_ID	Integer	A foreign key to the grid layout that is used when this data is represented relationally
SOURCE_FILENAME	String	The name of the file in the local filesystem that contains this product. This may be a single file, a directory name, or the name of an archive file containing multiple files (e.g., a ZIP file)
SUPERCEDED_BY	Integer	The PRODUCT_ID of a product that supercedes this one.
SUPERCEDES	Integer	The PRODUCT_ID of a product that this product supercedes. If this product supercedes more than one product, only one is listed here.

6.3.8 PRODUCT_STATUS

The PRODUCT_STATUS table defines the valid codes for the STATUS column of the PRODUCT table. The product status is typically one of the following: active, cancelled, test, archive, or unknown.

Column	Datatype	Description
PRODUCT_STATUS	String	Type code for product status
NAME	String	Status name
DESCRIPTION	String	Human readable meaning for this value

6.3.9 METRIC

The METRIC table defines the valid codes for the METRIC column of the PRODUCT table. Shaking metrics include “peak acceleration”, “maximum velocity”, “instrumental intensity”, and so on.

Column	Datatype	Description
METRIC_ID	Integer	Unique identifier for this metric.
NAME	String	Product type name
SHORT_NAME	String	Abbreviation for metric name
DESCRIPTION	String	Human readable meaning of this metric

6.3.10 PRODUCT_TYPE

The PRODUCT_TYPE table defines the valid codes for the PRODUCT_FORMAT column of the PRODUCT table. Product Types describe both the data in a product (e.g., "TV Map") and the file format (e.g., ".zip" or ".jpg").

Column	Datatype	Description
PRODUCT_TYPE	Integer	Format code for products
NAME	String	Product format name
DESCRIPTION	String	Human readable meaning for this value
FILENAME	String	The string used to construct the filename for this product type. This information is concatenated with the short name of the metric to construct a full filename to store the product.
METRIC_ID	Integer	Foreign key to the Metric table. Defines the metric contained in this Product Type. Not all Product Types have a metric (e.g., GRID_XYZ products contain a mix of products).

6.3.11 GRID

A ShakeMap Grid is a rectangular array of cells, each of fixed size (in terms of latitude and longitude). The GRID table describes this array.

Column	Datatype	Description
GRID_ID	Integer	Unique ID for this grid.
ORIGIN_LATITUDE	Float	Latitude of the origin of the grid
ORIGIN_LONGITUDE	Float	Longitude of the origin of the grid
LATITUDE_CELL_COUNT	Integer	Number of cells in the direction of latitude (north-south).
LONGITUDE_CELL_COUNT	Integer	Number of cells in the direction of longitude (east-west)
LAT_MIN	Float	Bounding box of the area covered by this product
LAT_MAX	Float	

LON_MIN	Float	
LON_MAX	Float	

6.3.12 GRID_VALUE

Each ShakeCast Grid File is also represented relationally as a set of rows in the GRID_VALUE table. The values are stored in a second-normal form instead of a third-normal form table for performance and efficiency. The values are stored in columns named VALUE_1, VALUE_2, and so on. The SHAKEMAP_METRIC table defines which metrics are stored in each value column.

Column	Datatype	Description
GRID_VALUE_ID	Integer	A unique ID for this Grid Value row.
SHAKEMAP_ID	Integer	Foreign key to the ShakeMap Table. Defines the ShakeMap for which this Grid Value applies.
SHAKEMAP_VERSION	Integer	
LAT_MIN	Float	Bounding box of the area covered by this product
LAT_MAX	Float	
LON_MIN	Float	
LON_MAX	Float	
VALUE_1	Float	The values for this grid cell for each metric generated by this ShakeMap are stored in these columns. The table SHAKEMAP_METRIC defines which column contains a particular metric. Additional columns may be added in the future. No assumption should be made about the order in which metrics appear in these columns.
VALUE_2	Float	
VALUE_3	Float	
VALUE_4	Float	
VALUE_5	Float	
VALUE_6	Float	
VALUE_7	Float	
VALUE_8	Float	
VALUE_9	Float	
VALUE_10	Float	

6.3.13 SHAKEMAP_METRIC

This table defines the metrics that are available from a particular ShakeMap. It also defines which column in the GRID_VALUE table contains the values for a particular metric for this ShakeMap.

Column	Datatype	Description

SHAKEMAP_ID	Integer	Foreign key to the ShakeMap Table. Defines the Shake-Map for which this Grid Value applies.
SHAKEMAP_VERSION	Integer	
METRIC_ID	Integer	Foreign key to the Metric Table
VALUE_COLUMN_NUMBER	Integer	The VALUE_x column number (i.e., VALUE_1, VALUE_2) in the GRID_VALUE table that contains this metric for this ShakeMap.
MIN_VALUE	Float	The minimum value of this metric in this ShakeMap (not including possible null values).
MAX_VALUE	Float	The maximum value of this metric in this ShakeMap (not including possible null values).

6.3.14 EXCHANGE_LOG

A product is moved between ShakeCast servers via an *exchange*. The EXCHANGE_LOG table records salient information about this activity

Column	Datatype	Description
EXCHANGE_LOG_ID	Integer	Locally unique identifier for each exchange operation
SERVER_ID	Integer	Globally unique ID of the server on the other end of the exchange. Foreign key to the SERVER table.
EXCHANGE_TYPE	String	Type of exchange activity. Foreign key to the EXCHANGE_TYPE table
EXCHANGE_ACTION	String	The action taken during this exchange. Foreign key to the EXCHANGE_ACTION table.
EXCHANGE_TIMESTAMP	Datetime	The date and time the exchange was initiated
EXCHANGE_ITEM_PK	Integer	A general parameter to hold the primary key value of the item exchanged. The interpretation of this value depends on the EXCHANGE_TYPE.
EXCHANGE_PARAMETER	String	A general parameter to contain additional data about the exchange. The interpretation of this column depends on the value of EXCHANGE_TYPE.

6.3.15 EXCHANGE_TYPE

The EXCHANGE_TYPE table defines the valid codes for the EXCHANGE_TYPE column of the EXCHANGE table. Valid exchange types include “Server Update”, “Product”, and “Miscellaneous”.

Column	Datatype	Description
EXCHANGE_TYPE	String	Type code for exchanges

NAME	String	Exchange type name
DESCRIPTION	String	Human readable meaning for this value

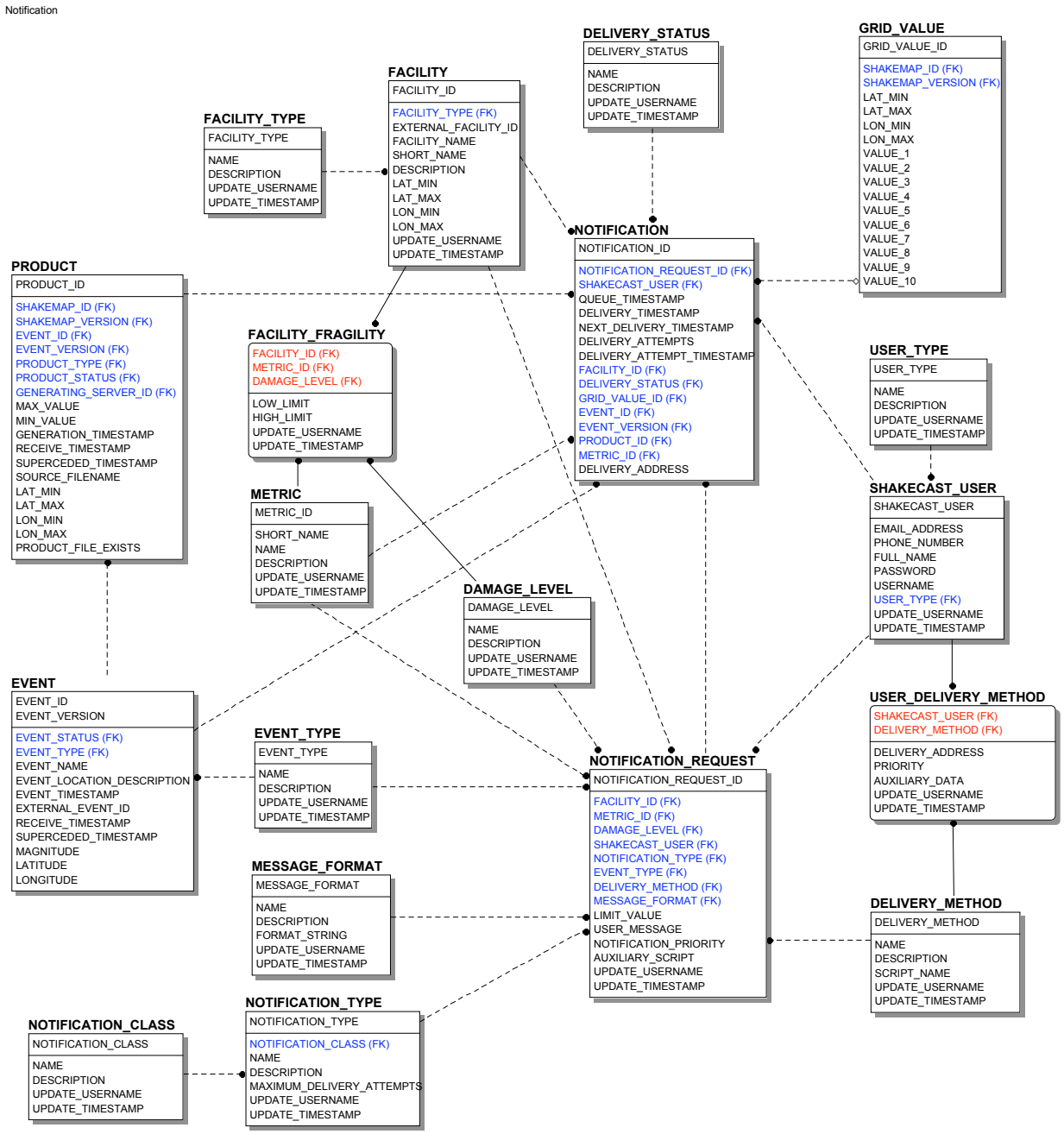
6.3.16 EXCHANGE_ACTION

The EXCHANGE_ACTION table defines the disposition of the exchange. Valid exchange action values include “Updated”, “Added”, “Ignored”, and “Rejected”.

Column	Datatype	Description
EXCHANGE_TYPE	Integer	Type code for exchanges
NAME	String	Exchange type name
DESCRIPTION	String	Human readable meaning for this value

6.4 Database Tables for Notification

In addition to tables defined elsewhere, the ShakeCast system uses the following tables to compute and execute notification operations.



6.4.1 FACILITY

A ShakeCast Facility is a specific structure (e.g., bridge, school, pumping station, etc.) at a specific location or a region. The location may be defined by a latitude/longitude for "point" facilities, or by a bounding box for non-point facilities. Note that the FACILITY table may be used to define a physical facility (e.g., a freeway overpass) or a region (e.g., a county). However, when used to specify a region, the FACILITY table must express the region as a rectangular bounding box.

Column	Datatype	Description
FACILITY_ID	Integer	A locally-unique primary key
NAME	String	Name of the facility
SHORT_NAME	String	Abbreviated name for facility
DESCRIPTION	String	A free text description or comment
UPDATE_TIMESTAMP	Datetime	The last time this table or the ShakeMap itself was updated.
LAT_MIN	Float	Bounding box of the area covered by this facility
LAT_MAX	Float	
LON_MIN	Float	
LON_MAX	Float	

6.4.2 NOTIFICATION_REQUEST

A ShakeCast notification event is generated for each NOTIFICATION_REQUEST where the value in a grid cell exceeds the corresponding value in the request.

Column	Datatype	Description
NOTIFICATION_REQUEST_ID	Integer	Locally generated primary key
SHAKECAST_USER	String	Foreign key to the USER table, defining which user is to be notified
FACILITY_ID	Integer	Foreign key to the FACILITY table. Used for facility-related notifications.
METRIC_ID	Integer	Foreign key to METRIC table. Defines the metric used in the notification computation.
DAMAGE_LEVEL	Integer	Foreign key to the DAMAGE_LEVEL table. Defines the damage level used in the notification computation.
NOTIFICATION_TYPE	Integer	Foreign key to NOTIFICATION_TYPE table
EVENT_TYPE	Integer	Foreign key to the EVENT_TYPE table. Define the type of events to which this notification request applies. For example, some notifications may apply only to “live” events, or just to scenarios, or just to test events.
DELIVERY_METHOD	Integer	Foreign key to the DELIVERY_METHOD table. Defines how the notification is to be delivered to the user.
MESSAGE_FORMAT	String	Foreign key to the MESSAGE_FORMAT table. The message format defines the layout of the message, such as which data items are to be included.

LIMIT_VALUE	Float	The value of this product in this cell.
METRIC_ID	Integer	Foreign key to METRIC table
NOTIFICATION_PRIORITY	Integer	Defines how this message is to be prioritized relative to other messages also to be sent to this user.
USER_MESSAGE	String	Arbitrary additional message the user wants sent when this notification request is executed
MAX_DELIVERY_ATTEMPTS	Integer	Maximum number of times the notification delivery is to be attempted
AUXILIARY_SCRIPT	String	Name of script to run to execute this notification request. (See [TBS] for more information on the calling of auxiliary scripts.)

6.4.3 NOTIFICATION

The NOTIFICATION table contains all current and historical notification requests that have been actually triggered by the ShakeCast system. The Notification table may be thought of as the queue of notification activity (for pending notifications) or the log of activity (for historical notifications).

Column	Datatype	Description
NOTIFICATION_ID	Integer	Locally generated primary key
NOTIFICATION_REQUEST_ID	Integer	Foreign key to NOTIFICATION_REQUEST table
SHAKECAST_USER	String	Foreign key to the USER table, defining which user is to be notified (denormalized)
QUEUE_TIMESTAMP	Datetime	Time the queue entry was created
NEXT_DELIVERY_TIMESTAMP	Datetime	Time the queue entry is next due to be processed
DELIVERY_TIMESTAMP	Datetime	Time the queue entry was successfully delivered
DELIVERY_ATTEMPTS	Integer	The number of times delivery has been attempted for this entry
DELIVERY_ATTEMPT_TIMESTAMP	Datetime	The last time delivery was attempted for this notification entry
DELIVERY_STATUS	Integer	Foreign key to DELIVERY_STATUS table. Contains last delivery status if delivery was attempted (may be success or errors), or completion or cancellation value.
GRID_VALUE_ID	Integer	Foreign key to the GRID_VALUE table. Contains the value for which this notification was triggered.
EVENT_ID	Integer	Foreign key to the EVENT table. Contains the event for which this notification was triggered.
EVENT_VERSION	Integer	
PRODUCT_ID	Integer	Foreign key to the PRODUCT table

PRODUCT_VERSION	Integer	
DELIVERY_ADDRESS	String	Actual user address (e.g., email address, pager ID, phone number, etc.) used for the message.

6.4.4 DELIVERY_STATUS

The DELIVERY_STATUS table defines the valid codes for the DELIVERY_STATUS column of the NOTIFICATION table. Valid types are locally defined.

Column	Datatype	Description
DELIVERY_STATUS	Integer	Status code
NAME	String	Status code name
DESCRIPTION	String	Description of the meaning of this status value.

6.4.5 DAMAGE_LEVEL

The DAMAGE_LEVEL table defines the valid levels of facility damage. Valid types are typically "Green", "Yellow", and "Red". Valid types are locally defined

Column	Datatype	Description
DAMAGE_LEVEL_ID	Integer	Type code for damage level, locally defined
NAME	String	Damage level name
SHORT_NAME	String	Abbreviation for damage level name
DESCRIPTION	String	Further descriptive information about the meaning of this damage level code

6.4.6 MESSAGE_FORMAT

The MESSGAE_FORMAT table defines the actual text and substitution directives for the message to be delivered. Messages may have varying lengths, be in various languages, or provide for substitution of various kinds of event and product data.

Column	Datatype	Description
MESSAGE_FORMAT	Integer	Type code for message format, locally defined
NAME	String	Notification type name
SHORT_NAME	String	Abbreviation for notification type name
DESCRIPTION	String	Description

FORMAT_STRING	String	A formatted string including substitution directives for event and product data.
TEMPLATE_FILE	String	Filename of a template to be used for constructing this message.

6.4.7 FACILITY_FRAGILITY

The FACILITY_FRAGILITY table defines the facility thresholds for a facility for each fragility level and each product metric.

Column	Datatype	Description
FACILITY_ID	Integer	Foreign key to the FACILITY table
PRODUCT_METRIC	Integer	Foreign key to the PRODUCT_METRIC table
FRAGILITY_LEVEL_ID	Integer	Foreign key to the FRAGILITY_LEVEL table
LOW_LIMIT	Float	Low limit of this fragility threshold at this facility
HIGH_LIMIT	Float	High limit of this fragility threshold at this facility
UPDATE_TIMESTAMP	Float	Last time this record was updated
UPDATE_USERNAME	String	Local username of the user who last updated this record

6.4.8 NOTIFICATION_TYPE

The NOTIFICATION_TYPE table defines the valid codes for the NOTIFICATION_TYPE column of the NOTIFICATION_REQUEST table. Valid types include “email”, “pager”, and “script”.

Column	Datatype	Description
NOTIFICATION_TYPE	String	Type code for notification requests
NAME	String	Notification type name
DESCRIPTION	String	Additional descriptive information about this notification type
MAXIMUM_DELIVERY_ATTEMPTS	Integer	Default value for maximum number of tries for this type of notification
NOTIFICATION_CLASS	String	Grouping value for notification types

6.4.9 NOTIFICATION_CLASS

The NOTIFICATION_CLASS table defines groups or classes of NOTIFICATION_TYPES.

Column	Datatype	Description
NOTIFICATION_CLASS	String	Type code for notification class
NAME	String	Notification class name
DESCRIPTION	String	Further descriptive information about the meaning of this notification class

6.4.10 SHAKECAST_USER

The USER table has a single record for each user who is to receive a notification.

Column	Datatype	Description
SHAKECAST_USER	String	Unique identifier for each user
FULL_NAME	String	Full name of the user
UPDATE_TIMESTAMP	Datetime	Time this record was last updated
PASSWORD	String	Hashed password of this user in the ShakeCast server
USER_TYPE	Integer	Foreign key to the USER_TYPE table
EMAIL_ADDRESS	String	Primary email address for this user
PHONE_NUMBER	String	Primary phone number for this user

6.4.11 USER_TYPE

The USER_TYPE table defines the valid codes for the USER_TYPE column of the USER table. Valid types are locally defined.

Column	Datatype	Description
USER_TYPE	String	Type code for user types
NAME	String	User type name
DESCRIPTION	String	Further description of the user type

6.5 Internal Operational Tables

A number of tables that define the operational configuration of the ShakeCast server. These tables are documented in the following paragraphs.

6.5.1 Processor Parameter Table

The Daemon Parameter Table stores parameters that control the behavior of the ShakeCast Processors such as the Exchange Processor, the Notification Processor, and the Message Processor.

Column	Datatype	Description
PROCESSOR_NAME	String	Name of the processor. Processors "know" their own name because it is passed to them as a parameter when they are invoked by the operating system.
PARAMETER_NAME	String	The name of the parameter for which the value is being defined.
PARAMETER_VALUE	String	The value to which the parameter is to be set. Both numeric and string values are stored as strings.

Chapter 7 Notification Processing

The ShakeCast system can notify system administrators and other users when "interesting" things happen. Notification messages are sent as SMTP (email) messages to one or more email addresses or text pager addresses. Almost any activity of a ShakeCast server can trigger a notification action.

Obvious notification messages include:

- When a particular facility has experienced a particular level of seismic shaking
- When a seismic event of a certain magnitude occurs
- When the shaking estimate for a previous event has been updated

Less obvious notification messages include:

- When a ShakeCast server has been started or stopped
- When a ShakeCast server encounters errors communicating with another server
- When a new ShakeCast system has joined the ShakeCast network
- When a new event occurs (irrespective of the effect on facilities)
- When a user notification activity reaches the maximum number of notification attempts without successful notification

The following sections in this Chapter describe how notification occurs in a ShakeCast Server.

7.1 Notification Processor Basics

The ShakeCast Notification Processor performs a number of basic functions on every kind of notification. The following paragraphs describe those basic functions.

7.1.1 Initiation of the Notification Processor by the Exchange Processor

Whenever the Exchange Processor (*Chapter 8, Exchange Processing*) completes an exchange, it does three things:

- The Exchange Processor writes key data to the ShakeCast Database. In the case of a new or updated event, only the EVENT table is written. In the case of a new or updated ShakeMap

product, the PRODUCT table is written. Other exchange activities might write to other tables.

- The Exchange Processor may initiates further exchange processing activities to get additional information associated with the recently-exchanged item. For example, if the exchanged item was an event, the Exchange Processor would start a processing loop to retrieve ShakeMap product files related to that event. This processing continues until the server has determined that all available supporting data has been received. (More detail on how the Exchange Processor works is found in *Chapter 8, Exchange Processing*).
- Next, the Exchange Processor also wakes the Notification Processor. The Notification Processor checks all registered notification requests to see if any new notification messages should be queued. After the Notification Processor successfully queues messages, it in turn wakes the Message Processor. The Message Processor continues to try to deliver messages until they have all been delivered or until the maximum number of tries has been exceeded for any messages that were undeliverable.

The following paragraphs describe the function of the Notification Processor and the Message Processor.

7.1.2 Notification Processor Environment

The ShakeCast Notification Processor runs as a separate process, independent of the Exchange Processor. The use of a separate process provides a degree of insulation from the scheduling issues associated with the Exchange Processor. Also, the Exchange Processor runs in the context of the Apache CGI environment, but the Notification Processor is a separate daemon (or service) and runs even if the CGI environment is busy or impacted by communication problems.

On most systems, the Notification Processor also runs at a higher scheduler priority than the Exchange Processor. Notifications are the important end goal of the ShakeCast System, and so it is appropriate that these activities take precedence over retrieving or sending additional data.

7.1.3 The Notification Request Table

The Notification Request table is the hub of the notification process. Each notification that a user requests is recorded as an entry in this table.

Users enter Notification Requests by filling in a Web-based form. The ShakeCast System also creates some Notification Requests as part of the installation and configuration process. The Notification Request table structure is shown in the following figure:

NOTIFICATION_REQUEST Table Structure		
Column	**Datatype**	**Description**
NOTIFICATION_REQUEST_ID	Integer	Locally generated primary key
GRID_CELL_ID	Integer	Grid cell to use for the computation
NOTIFICATION_TYPE	Integer	Foreign key to NOTIFICATION_TYPE table
LIMIT_VALUE	Float	The value of this product in this cell.

METRIC_ID	Integer	Foreign key to METRIC table
EVENT_TYPE	Integer	Foreign key to the EVENT_TYPE table
SHAKECAST_USER	String	Foreign key to the USER table, defining which user is to be notified
USER_MESSAGE	String	Arbitrary additional message the user wants sent when this notification request is executed
MAX_TRIES	Integer	Maximum number of times the delivery is to be attempted
AUXILIARY_SCRIPT	String	Name of script to run to execute this notification request. (See [TBS] for more information on the calling of auxiliary scripts.)

When the Notification Processor starts up, it uses the data in the Notification Request table to decide what needs to be done. There are three basic ways the Processor uses the Notification Request table:

- If the Processor was instructed to perform a particular type of notification, the Processor checks just that single type of notification in the Notification Request table. It will limit the SQL query against the table to just that single type of notification request.
- The Notification Processor may also be instructed to check for notifications related to a specific data instance. For example, the Processor may be instructed to check for notifications that are related to a specific event, to a specific ShakeCast data product, or to a specific facility.
- If the Processor received no specific instructions to limit the requests, then the Processor uses an SQL query that will check every notification request in the NOTIFICATION_REQUEST table.

All of the above queries can be satisfied by inspecting the NOTIFICATION_REQUEST table and a few related tables. This is why the NOTIFICATION_REQUEST table can be thought of as the center of the ShakeCast notification process.

7.1.4 Simple Event Notification

The following paragraphs will use as an example the particular case of notifying users about new Events. Similar scenarios can be drawn for other kinds of notifications, such as shaking at a facility or ShakeCast Server activity. These other scenarios will be expanded upon later.

For example, if the Notification Processor is instructed to notify users about new events only, it will query the Notification Request table with SQL that looks something like this:

```
select * from NOTIFICATION_REQUEST where NOTIFICATION_TYPE=3
```

where the type code “3” indicates EVENT notifications.

As the Processor reads notification requests, each kind of request is joined with the data that triggers that request. For example, if the user requests to be notified when a new event arrives, the Exchange Processor would trigger the Notification Processor with the specific event ID (say, event 12345), and the Notification Processor would issue SQL something like this:

```
select * from
      NOTIFICATION_REQUEST NR,
      EVENT E,
where
      NR.NOTIFICATION_TYPE=3 and
      E.EVENT_ID = 12345
```

Generally, users do not want to be notified of every single event. Let's consider a user who wants to only be notified when the event has a Richter magnitude greater than 5.0 and is located in the Los Angeles area. To specify this notification, the user will provide values for the LIMIT_VALUE and bounding box columns of the FACILITY table. The above query then becomes:

```
select * from
      NOTIFICATION_REQUEST NR,
      EVENT E,
      FACILITY F
where
      NR.NOTIFICATION_TYPE=3 and
      E.EVENT_ID = 12345 and
      E.RICHTER_MAGNITUDE >= NR.LIMIT_VALUE and
      NR.FACILITY_ID = F.FACILITY_ID and
      E.LAT between F.LAT_MIN and F.LAT_MAX and
      E.LON between F.LON_MIN and F.LON_MAX
```

The actual SELECT statement issued by the Notification Processor is somewhat more complicated than that indicated above. The Processor must handle a variety of cases, such as when the Event magnitude is null (not specified), or the user provided no limit value, or the user provided no bounding box, or the location of the event is not known, or some combination of these situations. However, these situations are simply extensions of the basic idea shown above.

Similar SELECT statements can be created for each of the kinds of notifications supported by ShakeCast. Each of these will be described in further detail later in this Chapter.

7.1.5 The Notification Queue Table

The Notification Processor does not simply retrieve the notification records as is shown in the SQL Select statement used above. Instead, the Processor stores the results of the join operation in the Notification Queue table. Whereas the Notification Request table records the user's requests for notification, the Notification Queue is a record of all the specific notification activities that the ShakeCast Server has attempted to perform. The Notification Queue table structure is shown in the following figure:

NOTIFICATION Table Structure		
Column	**Datatype**	**Description**
NOTIFICATION_ID	Integer	Locally generated primary key
NOTIFICATION_REQUEST_ID	Integer	Foreign key to NOTIFICATION_REQUEST table
SHAKECAST_USER	String	Foreign key to the USER table, defining which user is to be notified (denormalized)

QUEUE_TIMESTAMP	Datetime	Time the queue entry was created
SCHEDULED_TIMESTAMP	Datetime	Time the queue entry is next due to be processed
DELIVERY_TIMESTAMP	Datetime	Time the queue entry was successfully delivered
DELIVERY_STATUS	Integer	Foreign key to DELIVERY_STATUS table. Contains last delivery status if delivery was attempted (may be success or errors), or completion or cancellation value.
TRIES	Integer	Number of times the delivery was attempted
GRID_VALUE_ID	Integer	Foreign key to the GRID_VALUE table
FACILITY_ID	Integer	Foreign key to the FACILITY table
EVENT_ID	Integer	Foreign key to the EVENT table

As the Notification Processor finds notification requests that are to be processed, it makes an entry in the NOTIFICATION table. Returning to the example in the previous section, the SQL Select statement becomes an SQL Insert statement, as below:

```
insert into NOTIFICATION
      (NOTIFICATION_REQUEST_ID, SHAKECAST_USER, QUEUE_TIMESTAMP,
      EVENT_ID, FACILITY_ID)
select NOTIFICATION_REQUEST_ID, SHAKECAST_USER, NOW(),
      EVENT_ID, FACILITY_ID
from
      NOTIFICATION_REQUEST NR,
      EVENT E,
      FACILITY F
where
      NR.NOTIFICATION_TYPE=3 and
      E.EVENT_ID = 12345 and
      E.RICHTER_MAGNITUDE >= NR.LIMIT_VALUE and
      NR.FACILITY_ID = F.FACILITY_ID and
      E.LAT between F.LAT_MIN and F.LAT_MAX and
      E.LON between F.LON_MIN and F.LON_MAX
```

Other columns of the NOTIFICATION table are filled in automatically by the database or the processor. For example, the NOTIFICATION_ID is automatically created, and the DELIVERY_STATUS is set to the initial value of "QUEUED".

One further refinement is required before the above query form is used by the Notification Processor. The statement shown above will create a new entry on the Notification Queue for every matching Notification Request, every time the statement is executed. However, we do not want to issue a notification request if there is already a request outstanding for this event. Therefore, we want to limit the query with a further predicate:

```
insert into NOTIFICATION
(NOTIFICATION_REQUEST_ID, SHAKECAST_USER, QUEUE_TIMESTAMP, EVENT_ID,
FACILITY_ID)
select NOTIFICATION_REQUEST_ID, SHAKECAST_USER, NOW(), EVENT_ID,
FACILITY_ID
from
      NOTIFICATION_REQUEST NR,
      EVENT E,
      FACILITY F
```

```
where
      NR.NOTIFICATION_TYPE=3 and
      E.EVENT_ID = 12345 and
      E.RICHTER_MAGNITUDE >= NR.LIMIT_VALUE and
      NR.FACILITY_ID = F.FACILITY_ID and
      E.LAT between F.LAT_MIN and F.LAT_MAX and
      E.LON between F.LON_MIN and F.LON_MAX and
      not exists (select * from NOTIFICATION N where
                  N.NOTIFICATION_REQUEST_ID =
NR.NOTIFICATION_REQUEST_ID and
                  N.EVENT_ID = E.EVENT_ID)
```

The `not exists` predicate will prevent the Notification Processor from inserting queue items that match items that already exist. Further refinements on this idea are described later.

7.2 Message Delivery Processor

Once messages are in the NOTIFICATION, the Notification Processor "wakes up" the Message Delivery Processor. This Processor, which also runs in a process context separate from both the Notification Processor and the Exchange Processor, runs essentially continuously. The Message Processor attempts to deliver messages in the NOTIFICATION table.

The Notification Processor does not need to tell the Message Processor the specifics of what it needs to do. It simply wakes the Message Processor. Each time it runs, the Message Processor always tries to deliver every undelivered message in the Notification Queue.

7.2.1 Selecting Messages to Send

To determine which messages need to be sent, the Message Processor uses a basic SQL Select statement similar to this one:

```
select *
from
      NOTIFICATION N,
      NOTIFICATION_REQUEST NR,
      SHAKECAST_USER U
where
      N.NOTIFICATION_REQUEST_ID = NR.NOTIFICATION_REQUEST_ID and
      N.SHAKECAST_USER = U.SHAKECAST_USER and
      N.NEXT_DELIVERY_TS <= now() and
      N.DELIVERY_STATUS = 1    /* Delivery pending */
order by
      N.SHAKECAST_USER
```

7.2.2 Message Aggregation

The Message Delivery Processor loops over every record returned by the above SQL statement. Messages to the same user for the same facility are grouped together. For example, if a single facility exceeds the shaking limits for four separate shaking metrics, the Message Processor aggregates those four entries in the NOTIFICATION into a single message. This process is termed Message Aggregation.

[more on message aggregation tbs]

7.2.3 Completion of Notification Queue Entries

The Processor then attempts to deliver the message. As each message is delivered, the processor updates the columns in the NOTIFICATION table for each component of the delivered message, as shown below.

Message Processor Updates to NOTIFICATION	
Column	**Description**
SCHEDULED_TIMESTAMP	If the delivery fails, this column is updated with the time when the delivery should be re-tried. Parameters in the ShakeCast configuration tables determine how rapidly failed deliveries are rescheduled. The general algorithm is that the delivery is attempted several times in rapid succession. If all those tries fail, the Processor then starts to decay the retry time by doubling the time between retries. Eventually, the retry time reaches a maximum time between tries, and the Processor continues to retry delivery with the maximum delay until the maximum number of notification attempts is reached. The maximum number of notification attempts is defined for each NOTIFICATION_TYPE. (Reports of ShakeCast Server errors, for example, retry essentially without bound.)
DELIVERY_TIMESTAMP	If the message is successfully delivered, this column is updated with the time the delivery succeeded.
ATTEMPT_TIMESTAMP	The time the delivery was last attempted.
DELIVERY_STATUS	The Message Processor sets this status value to SUCCESS when the delivery completes successfully. It is set to FAILURE when a delivery attempt fails. The status may also be set to CANCELLED by an administrator via a maintenance web page.
DELIVERY_ATTEMPTS	This number is incremented on each delivery attempt.

7.2.4 Iterative Message Processing

Each time the Message Processor completes a loop through all the current message records, it repeats the process, again setting the query predicate `N.NEXT_DELIVERY_TIMESTAMP <= now()`. If no records are returned (all the delivery times are in the future), the Message Processor tries to determine how long to wait. It does this using the following SQL query:

```
select min (NEXT_DELIVERY_TIMESTAMP)
from
      NOTIFICATION N
where
      N.DELIVERY_STATUS = 1    /* Delivery pending */
```

The Message Processor compares the Next Delivery Time with the maximum wait time set in the system configuration, and waits until whichever time is earliest. It then wakes itself when that time arrives, and repeats the process.

Note that the Message Processor attempts to deliver all outstanding messages that have reached their assigned scheduled time, every time it sweeps the Notification Queue. It attempts delivery

in Priority order, but it will try even low priority messages before it re-attempts delivery of higher priority messages. Although messages may take only a second or two to deliver, there are a variety of message delivery failure modes where it can take many seconds to detect that a message is undeliverable. Therefore, it is important that exceedingly large numbers of undelivered and undeliverable messages not be allowed to build up in the Notification Queue. The MAXIMUM_NOTIFICATION_ATTEMPTS column of the NOTIFICATION_TYPE table and the other system configuration parameters that control the delivery of messages should therefore be adjusted only with some degree of caution.

7.3 Notification Signup Web Page

[tbs]

7.4 Event Notification

When a ShakeCast Event is received by the Exchange Processor, the Exchange Processor completes the current exchange and then wakes the Notification Processor. The message to wake the Notification Processor includes all Notification Types that are in the Event Notification Class. The Notification Processor then executes a series of SQL statements that relate the EVENT table and the NOTIFICATION_REQUEST table, of the form:

```
insert into NOTIFICATION
      (NOTIFICATION_REQUEST_ID, SHAKECAST_USER, QUEUE_TIMESTAMP,
EVENT_ID)
select NOTIFICATION_REQUEST_ID, SHAKECAST_USER, NOW(), EVENT_ID
from
      NOTIFICATION_REQUEST NR,
      EVENT E,
      NOTIFICATION_TYPE NT
where
      NR.NOTIFICATION_TYPE = NT.NOTIFICATION_TYPE and
      NT.NOTIFICATION_CLASS = 1 and /*Event Class*/
      E.EVENT_ID = <eid> and
      not exists (select * from NOTIFICATION N where
            N.NOTIFICATION_REQUEST_ID = NR.NOTIFICATION_REQUEST_ID and
            N.EVENT_ID = E.EVENT_ID and
            N.DELIVERY_STATUS = 1 /*Pending*/
            ) and
<further predicates>
```

The various combinations of further predicates are listed in the following table, and may all be ORed together in a single SQL Where Clause.

Event Notification Query Predicates

Predicate	Description
`NR.LIMIT_VALUE is null and F.FACILITY_ID = NR.FACILITY_ID and E.LAT between F.LAT_MIN and F.LAT_MAX and E.LON between F.LON_MIN and F.LON_MAX`	A limit is not specified, but a facility is specified, and the specified facility's bounding box overlaps the location of the event. Note that a facility in this case actually represents a region such as a utility service area, not a physical facility like a reservoir.

NR.FACILITY_ID is null and NR.LIMIT_VALUE is not null and E.MAGNITUDE >= NR.LIMIT_VALUE	No facility (location) is provided, but the event magnitude exceeds the requested limit. The limit value must not be null in this case. Users can use this case to receive notification of every event, irrespective of the event's location.
NR.LIMIT_VALUE is not null and NR.FACILITY_ID is not null and E.MAGNITUDE >= NR.LIMIT_VALUE E.LAT between F.LAT_MIN and F.LAT_MAX and E.LON between F.LON_MIN and F.LON_MAX	A limit is defined and the facility is defined, and the magnitude of the event exceeds the defined limit, and the event is within the defined bounding box.

7.5 Product Notification

When a ShakeCast Product is received by the Exchange Processor, the Exchange Processor completes the current exchange and then wakes the Notification Processor. The message to wake the Notification Processor includes all Notification Types that are in the Product Notification Class. The Notification Processor then executes a series of SQL statements that relate the PRODUCT table and the NOTIFICATION_REQUEST table.

7.5.1 Product Notification Without Facility

This type of Notification Request is used when the user wants to be notified whenever a ShakeMap Product is created that reports values above a predetermined limit, irrespective of the location covered by the ShakeMap.

In some cases, the user will want to be notified whenever products of a certain type arrive, or when those Product records report maximum values (for their respective Metric) above a specified limit. The following SQL statement is used in this case:

```
insert into NOTIFICATION
(NOTIFICATION_REQUEST_ID, SHAKECAST_USER, QUEUE_TIMESTAMP, EVENT_ID,
PRODUCT_ID)
select NOTIFICATION_REQUEST_ID, SHAKECAST_USER, NOW(), EVENT_ID,
PRODUCT_ID
from
      NOTIFICATION_REQUEST NR,
      PRODUCT P,
      NOTIFICATION_TYPE NT
where
      NR.NOTIFICATION_TYPE = NT.NOTIFICATION_TYPE and
      NT.NOTIFICATION_CLASS = 2 and /*Product Class*/
      P.PRODUCT_ID = <pid> and
      NR.METRIC_ID  = P.METRIC_ID and
      ((NR.LIMIT_VALUE is null) or (NR.LIMIT_VALUE < P.MAX_VALUE))
      not exists (select * from NOTIFICATION N where
                  N.NOTIFICATION_REQUEST_ID =
                        NR.NOTIFICATION_REQUEST_ID and
                  N.PRODUCT_ID = P.PRODUCT_ID and
                  N.DELIVERY_STATUS=1 /*Pending*/)
```

7.5.2 Product Notification With Facility

This type of Notification Request is used when the user wants to be notified whenever a ShakeMap Product is created that overlaps one or more Facilities of interest, without regard to the level of shaking that might actually have occurred in the immediate vicinity of the Facility.

When the user also specifies a Facility in the Notification Request, the SQL executed by the must include a comparison of the bounding box of the Facility and the bounding box of the Product. To include a comparison of the Facility, the above SQL statement is extended to include a reference to the FACILITY table and a predicate based on the bounding box of the Facility and the bounding box of the Product.

```
insert into NOTIFICATION
(NOTIFICATION_REQUEST_ID, SHAKECAST_USER, QUEUE_TIMESTAMP, EVENT_ID,
PRODUCT_ID)
select NOTIFICATION_REQUEST_ID, SHAKECAST_USER, NOW(), EVENT_ID,
PRODUCT_ID
from
      NOTIFICATION_REQUEST NR,
      PRODUCT P,
      NOTIFICATION_TYPE NT,
      FACILITY F
where
      NR.NOTIFICATION_TYPE = NT.NOTIFICATION_TYPE and
      NT.NOTIFICATION_CLASS = 2 and /*Product Class*/
      P.PRODUCT_ID = <pid> and
      NR.METRIC_ID  = P.METRIC_ID and
      ((NR.LIMIT_VALUE is null) or (NR.LIMIT_VALUE < P.MAX_VALUE)) and
      not ( (F.LON_MAX < P.LON_MIN) or (P.LON_MAX < F.LON_MIN) or
            (F.LAT_MAX < P.LAT_MIN) or (P.LAT_MAX < F.LAT_MIN) ) and
      not exists (select * from NOTIFICATION N where
                  N.NOTIFICATION_REQUEST_ID =
                        NR.NOTIFICATION_REQUEST_ID and
                  N.PRODUCT_ID = P.PRODUCT_ID and
                  N.DELIVERY_STATUS=1 /*Pending*/ )
```

7.6 Grid Notification

When a ShakeCast Product that includes a Grid is received by the Exchange Processor, the Exchange Processor completes the Grid Data Exchange and then wakes the Notification Processor. The message to wake the Notification Processor includes all Notification Types that are in the Grid Notification Class.

Grid Notification Requests are designed to generate Notification Queue entries based on the value of a particular metric at a particular location (one or more grid cells). The Grid Cells involved in the comparison are located by their overlap with a Facility. Therefore, a Facility (with a locating bounding box) is required for a Grid Notification Request.

Grid Notification Requests are of two sub-forms:

- The user may specify a limit value in the Notification Request. When the value of the metric exceeds the limit value for any Grid Cell that overlaps the Facility, then the Notification Request is matched.

- The user may instead specify a Damage Level. The Damage Level is used together with estimates of the "fragility" of the facility (resistance to shaking). When the Shaking estimated at any Grid Cell associated with the Facility exceeds the limit associated with the specific Damage Level for that facility, then the Notification Request is matched.

Each of these kinds of Grid Notification Requests is described below.

7.6.1 Outstanding Grid Notification Records

To start the process of creating new Notification Queue entries, the Notification Processor finds existing notifications for this User on this Facility. These may be notifications that are any currently outstanding or have already been completed. These existing notifications will be used later to filter the new notifications, so that multiple notifications are not issued for a particular user for a particular facility, unless the new notification suggests a higher Damage Level estimate than the previous notification.

The following query is used to identify potential existing notification activity in the Notification Queue.

```
select
      NOTIFICATION_REQUEST_ID, SHAKECAST_USER, QUEUE_TIMESTAMP,
      EVENT_ID, PRODUCT_ID, MAX_VALUE
from  NOTIFICATION N,
      PRODUCT P
where DELIVERY_STATUS = 1 and /*Pending delivery*/
      N.METRIC = P.METRIC
```

New notification entries will not be created that match the above entries. Instead, the existing entries will be modified with additional data from the new entries.

7.6.2 First Level Filter on Product Maximum Value

The Notification Processor next executes an SQL query that relates the PRODUCT table, the FACILITY_FRAGILITY table, the FACILITY table, and the NOTIFICATION_REQUEST table. This statement is a first level filter that identifies any NOTIFICATION_REQUEST records that might possibly be triggered by this Product. The query predicate is based on the comparison of the column PRODUCT.MAX_VALUE to the low-limit and high-limit values of the FACILITY_FRAGILITY table. It also filters records based on the comparison between the bounding box of the PRODUCT and the bounding box or location of the FACILITY.

```
select NOTIFICATION_REQUEST_ID
from  NOTIFICATION_REQUEST NR,
      FACILITY F,
      FACILITY_FRAGILITY FF,
      PRODUCT P
where P.PRODUCT_ID = <pid> and
      P.PRODUCT_VERSION = <version>
      NR.NOTIFICATION_TYPE = NT.NOTIFICATION_TYPE and
      NT.NOTIFICATION_CLASS = 3 and /*Grid Class*/
      P.METRIC_ID = NR.METRIC_ID and P.METRIC_ID = FF.METRIC_ID and
      NR.DAMAGE_LEVEL_ID = FF.DAMAGE_LEVEL_ID and
      NR.FACILITY_ID = F.FACILITY_ID and
      NR.FACILITY_ID = FF.FACILITY_ID and
      (P.MAX_VALUE between FF.LOW_LIMIT and FF.HIGH_LIMIT or
```

```
        (P.MAX_VALUE >= FF.LOW_LIMIT and FF.HIGH_LIMIT is null)) and
      not ( (F.LON_MAX < P.LON_MIN) or (P.LON_MAX < F.LON_MIN) or
            (F.LAT_MAX < P.LAT_MIN) or (P.LAT_MAX < F.LAT_MIN) )
```

The above query can be executed fairly quickly. It returns all the possible Notification Requests that *may* be triggered by the given product. However, it also returns false positive Notification Requests: requests that might have been triggered based on the coarse maximum value given in the PRODUCT record, but that are not actually requested when the higher-resolution GRID_VALUE table is also examined.

7.6.3 Fine Granularity Notification Request Processing

Using the list of request IDs returned from the previously-defined coarse filter, the Notification Processor next examines the Grid Values themselves, and compares these values to the limits set in the FACILITY_FRAGILITY table.

```
select
      NOTIFICATION_REQUEST_ID, SHAKECAST_USER, NOW(),
      EVENT_ID, PRODUCT_ID, MAX(GV.VALUE)
from
      NOTIFICATION_REQUEST NR,
      GRID_VALUE GV,
      GRID_CELL GC,
      FACILITY_FRAGILITY FF
where
      NR.NOTIFICATION_REQUEST_ID in (first-level-list) and
      GV.PRODUCT_ID = <pid> and
      GV.PRODUCT_VERSION = <version> and
      GV.GRID_ID = GC.GRID_ID and
      NR.METRIC_ID = FF.METRIC_ID and
      NR.METRIC_ID = GV.METRIC_ID and
      FF.METRIC_ID = GV.METRIC_ID and /*reflexive (redundant)*/
      FF.DAMAGE_LEVEL_ID = NR.DAMAGE_LEVEL_ID and
      F.FACILITY_ID = NR.FACILITY_ID and
      ((GV.VALUE between FF.LOW_LIMIT and FF.HIGH_LIMIT) or
       (GV.VALUE >= FF.LOW_LIMIT and FF.HIGH_LIMIT is null)) and
      (not ((F.LON_MAX < GC.LON_MIN) or (GC.LON_MAX < F.LON_MIN) or
            (F.LAT_MAX < GC.LAT_MIN) or (GC.LAT_MAX < F.LAT_MIN) )
group by
      NOTIFICATION_REQUEST_ID, SHAKECAST_USER, EVENT_ID, PRODUCT_ID
```

The above query also compares the Grid Cell bounding box with the Facility bounding box, and includes only Grid Values for Grid Cells that overlap with the Facility in the Notification Request.

7.7 Metadata Update Notification

A ShakeCast Server depends on a great many different kinds of metadata (data about data):

- Data about the various ShakeMap product metrics that are available
- Data about the various ShakeMap product formats that are available
- Data about system usage

The ShakeCast System Administrator can configure the ShakeCast server to automatically poll other servers for these various metadata elements, and automatically post this information to the server. As part of the Exchange Processing that takes place to retrieve this data, the ShakeCast scripts also post entries to the Notification Request Queue so that administrators can be notified of the changes that have been made. These notification requests are created using an SQL statement similar to the following:

```
insert into NOTIFICATION
      (NOTIFICATION_REQUEST_ID, SHAKECAST_USER, QUEUE_TIMESTAMP, URL)
select
      NOTIFICATION_REQUEST_ID, SHAKECAST_USER, NOW(), '<url>'
from  NOTIFICATION_REQUEST NR
where NOTIFICATION_TYPE=5 /*Metadata change notification type*/
```

The URL in the above statement is provided by the Exchange Processor when the queue entry is written, and is usually a pointer to a ShakeCast report page that describes the new metadata.

[more tbs]

7.8 Registration and New System Notification

The ability to make critical ShakeCast system configuration changes with a minimum of effort is an essential component of the "set it and forget it" nature of the ShakeCast System. A ShakeCast server is not a standalone computer system. Even if your ShakeCast Server is never changed, it is awash in a sea of changes in the ShakeCast network as new servers are added, old servers stop working, and new types of ShakeMaps are created. The functions described in this Chapter make it possible for a system administrator to take maintain a ShakeCast system without ever taking any pro-active steps. Instead of the administrator having to know "what to do", the ShakeCast system is designed to run unattended, and ask the administrator for "approval" in those few cases where approval is needed before a configuration change is applied.

New ShakeCast servers can be added to the ShakeCast network with minimal intervention by an administrators at all the many machines in the network. In addition, new connections can be created between existing ShakeCast servers, allowing information to flow between these machines. These two processes are described in the following paragraphs.

7.8.1 System Registration and Notification

When the ShakeCast software is installed, as one of the final steps in the configuration process the server registers itself with at least one upstream ShakeCast machine. This is accomplished by issuing a `register` request to the upstream server (see *Section Chapter 3, ShakeCast Requests* and *Section 2.4, Upstream and Downstream ShakeCast Machines*).

The upstream server logs information about the registering server in the SERVER table of the ShakeCast database. However, the new server is not automatically enabled for data exchanges. Instead, the `register` request places entries in the Notification Queue so that New System Notification messages are delivered to system administrators.

The Register Request uses an SQL query like the following to generate New System Notification Requests.

```
insert into NOTIFICATION
```

```
      (NOTIFICATION_REQUEST_ID, SHAKECAST_USER, QUEUE_TIMESTAMP, URL)
select
      NOTIFICATION_REQUEST_ID, SHAKECAST_USER, NOW(), '<url>'
from  NOTIFICATION_REQUEST NR
where NOTIFICATION_TYPE=4 /*System registration request*/
```

The URL argument in the above query is constructed by the `register` request, and consists of a pointer to administrative CGI script `approve_registration_request`. This URL is formed as follows:

```
/sc/admin/approve_registration_request?system_id=<sid>
```

The function of this request is described in *Section 7.8.3 Approving Registration Requests*.

7.8.2 Registering Other New Systems

ShakeCast servers can learn about other systems in the network in two other ways:

- Another server already known to our ShakeCast Server can "push" new server information toward our server.
- Our server can "poll" other servers, periodically asking about new servers that have joined the network and that may require service from us.

When a server already known to ours wants to inform us of a new server, it does this by issuing a `new_server` request. This request is processed in the context of an Exchange (*Chapter 8, Exchange Processing*), and updates the ShakeCast database with information about the new server. When the Exchange is complete and the database has been updated, the Exchange Processor writes a Notification Request Queue entry to inform the system administrator about the new system, and then wakes the Message Processor to deliver that request to the system administrator.

Similarly, a ShakeCast System Administrator can configure his system to poll all adjacent (connected) servers and request a list of servers that offer download service. This is achieved by executing the `get_system_list` request (see *Chapter 3, ShakeCast Requests*). When the exchange initiated by the `get_system_list` request is completed and the database has been updated, the Exchange Processor writes a Notification Request Queue entry to inform the system administrator about the new system, and then wakes the Message Processor to deliver that request to the system administrator.

7.8.3 Approving Registration Requests

Both the New System Registration process and the New Server request result in a new server being added to the ShakeCast database. Before the new system is activated, however, the ShakeCast System Administrator must invoke the `approve_registration_request` script via a Web page.

The `approve_registration_request` script performs the following functions.

- Check that the user invoking the script is registered in the SHAKECAST_USER database with a User Type of "Administrator".

- Generate a Web page presenting the configuration information for the server that has requested registration:
  - Server name, address, organization, and so on
  - Version of the ShakeCast software the system is running
  - The Administrative Contact Information for the server's administer, as defined in the System XML that is exchanged between the two servers.
  - A hash value representing the current state of this server's record in the ShakeCast database. This value will be sent back to the server with the update request.
- The generated page also has a series of checkboxes for the administrator to use to configure the characteristics of the new server.
- When the administrator approves the request, the same script is invoked again, this time instructing the script to apply the changes to the ShakeCast Database.

7.9 ShakeCast System Activity Notification

ShakeCast can be configured to notify System Administrators or others when a variety of important system events or system activity occur. System Administrators (or other users) request notification on the ShakeCast Notification Signup Web Page (see *Section 7.3, Notification Signup Web Page*).

The kinds of events and activity that can trigger notification are summarized in the following table.

System Activity Notification Types		
Class	**Activity**	**Description**
Error	System Restart	The ShakeCast System has been restarted
	Communication Error Limit Reached	A ShakeCast partner system has encountered communication errors exceeding the limit set in the ShakeCast Database SERVER table.
	Delivery Attempts Limit Reached	A ShakeCast message could not be delivered because the maximum number of delivery attempts was reached.
	Self-Test Failed	The ShakeCast System failed some or all of the ShakeCast Self-Test
Configuration	New Server Added	A new ShakeCast Server has been added to the list of servers known to this server.
Usage	Daily Usage Report	A report summarizing of the number of email messages sent and web pages viewed today.
	Daily Exchange Report	A report summarizing of the number of ShakeCast Exchanges completed today.
	Monthly Usage Report	A report summarizing of the number of email messages sent and web pages viewed this month.

	Monthly Exchange Report	A report summarizing of the number of ShakeCast Exchanges completed this month.
	User Summary	A report summarizing of the number of ShakeCast users by user type, user organization (department), and activity or usage level.

The following sections describe each of these notification mechanisms in more detail.

7.9.1 Error Notification

[tbs]

7.9.1.1 System Restart

[tbs]

7.9.1.2 Communication Error Limit

[tbs]

7.9.1.3 Delivery Attempts Limit

[tbs]

7.9.2 System Configuration Notification

[tbs]

7.9.2.1 New System Added

[tbs]

7.9.3 Usage Notification

[tbs]

7.9.3.1 Daily and Monthly Usage Report

[tbs]

7.9.3.2 Daily and Monthly Exchange Report

[tbs]

7.9.3.3 User Summary Report

[tbs]

Chapter 8 Exchange Processing

A ShakeCast Exchange is the process of moving data or metadata from one ShakeCast Server to another. An Exchange can be performed when an upstream ShakeCast server sends unsolicited data to a downstream server, or when a downstream server "polls" or requests data from an upstream server. Some exchanges are combinations of these two styles of communication; an upstream server sends a brief notification to a downstream server, which then "decides" whether or not it needs additional data, and, if it does, makes a request for that information.

A ShakeCast Exchange may move data (for example, a ShakeMap Grid), or it may move metadata (for example, a Product Format XML file containing information about new ShakeMap product formats). The exchange process for these two kinds of exchange are described separately below.

The following sections in this Chapter describe the methods that ShakeCast Servers use to accomplish these data exchanges.

8.1 Creating and Updating ShakeCast Databases

In ShakeCast, all requests to exchange data are idempotent. That is, a request can be repeated any number of times and, as long as the data in the request is identical, the resulting database is the same as if the request had been issued only once. For example, a new ShakeCast event is created by the `new_event` request. A ShakeCast Server can receive the `new_event` request several times, from the same server and/or other severs, and the server will define the event in the ShakeCast Database only one time; it will not define a different event for each time the request was received.

In conjunction with the idempotent nature of ShakeCast requests is a principle of "last update wins". If a ShakeCast Server receives two requests, and the data in the requests is in conflict, then the latest request to arrive controls the value that is stored in the ShakeCast Database. For example, if a `new_event` request arrives and defines Event 1234 to be at the location "34.147689:-118.128808", but a subsequent request defines the same event at location "34.335439:-118.506447", then the second value is the value retained in the ShakeCast database.

8.2 The ShakeCast Exchange Log

Every request that exchanges data is also recorded in the ShakeCast Exchange Log of both the upstream server and the downstream server. The Exchange log is a time-indexed record of all exchanges, and is stored as the EXCHANGE_LOG table in the ShakeCast Database. The Exchange Log contains the following data:

Exchange Log	
Column	**Description**
EXCHANGE_TIMESTAMP	The time (GMT, from the local system's clock) that the exchange was completed.
SERVER_ID	The globally unique system ID of the ShakeCast Server on the other side of the exchange.
EXCHANGE_TYPE	The type of exchange that took place. Values include "Metadata", "Data", and "Data File".
EXCHANGE_ACTION	For the downstream server, the action taken as a result of the values given in the exchange. The action may be "Insert", "Update", "Rejected due to errors", or "No change required".

8.3 Metadata Exchanges

The ShakeCast System provides a mechanism to update all the Servers in the network with ShakeCast metadata. For example, when the ShakeMap developers create a new kind of ShakeMap Metric, this information needs to be distributed to all the servers in the network. By entering a new Metric in the database, users can begin to define Facility Fragilities that are based on that product. Or, the ShakeMap developers may decide that a new Event Status value is required to handle a new kind of test event that has been created.

All ShakeCast Servers accept metadata updates. These updates insert or update new values in the ShakeCast database tables.

Metadata Exchanges always occur via the exchange of a metadata XML file. The Metadata XML format provides for the following data items.

Metadata XML	
Data Item	**Description**
Table Name	The name of the table that is to be updated.
Primary Key Value	The value of the primary key in that table that is to be inserted or updated.
Column Value Pairs	An array of column-value pairs. On insert, each of the columns listed is set to the value given. On update, only the listed columns are updated; the values of unnamed columns in the database are not changed.

Since the Metadata XML format can describe any relational table, it can be used to update any of the ShakeCast metadata such as Product Format, Metric, Event Status, and so on. Refer to *Section5.7, ShakeCast Metadata XML*, for the details of the Metadata XML format.

To update ShakeCast Metadata, the upstream server creates (or retrieves) XML describing the update to be performed. The downstream server then reads that XML and performs the update.

[more tbs]

8.4 Data Exchanges

When a ShakeCast Server processes a request to store data in the ShakeCast Database, the request processor must perform a number of steps before writing the data to the ShakeCast database. It does this using the following algorithm:

- Check first to see if the object defined in the request is already in the database. If it is, then the subsequent operations are SQL Updates instead of SQL Inserts. The object is located by the primary key column only; if the primary key (which much be supplied as a request parameter) is matched in the database, then the matched record is updated. If the primary key is not matched, then the request is treated as defining a new record.
- The parameters are matched to database table columns using code and logic that is unique to the code in each request processor.
- Depending on the particular request, the script may apply various validity tests against the data.
- The record is then update or inserted.
- If the update or insert fails for any reason (for example, because a database constraint is violated, or because a validity test failed), then the script writes the error message into a ShakeCast Status XML message and returns without updating the database. If the update succeeds, then the script writes a success message into the ShakeCast Status XML and commits the changes to the database.

Data is exchanged between ShakeCast Servers in two ways:

- As parameters to ShakeCast Requests
- As XML

How a ShakeCast server extracts data from each of these mechanisms is described below.

8.4.1 Data Exchanged as Request Parameters

A ShakeCast Request can include parameters that refer to data found in the ShakeCast Database. For example, the `get_product` request can refer to an Event ID to ask for products associated with a single Event. In these cases, the data is used as a query predicate on the upstream server.

A ShakeCast Request can also include parameters that are not used as query predicates, but instead are used to set or define values for a ShakeCast database object, such as an Event or Product. For example, the ShakeCast `new_event` request always includes the `event_id` parameter, because it is creating a new event. The request can also include the `event_location` parame-

ter to pass a ShakeCast server the location of the earthquake. Requests sometimes include additional parameters (such as the `event_location` parameter in the above example) so that downstream servers do not have to ask for data that they may not want.

Data in requests is always text, but the values stored in the ShakeCast database may be text or binary (integer, float, or date-time). A server that is storing data in the database attempts to cast the parameters from text into the datatype defined for the column in the database. If the data does not successfully convert into that format, then the script returns an error, which is written to the ShakeCast Status XML and returned to the requesting server.

8.4.2 Data Exchanged as XML

A ShakeCast Request can include data that is stored in an XML message. For example, if a downstream server issues a `get_event_list` request, the upstream server can store all of the event data in an Event XML message and return all of the data in the payload of the request response.

When data is provided in an XML message, the ShakeCast server parses the XML using [tbs – how to handle DTDs]. The parsed data is then stored. Data in XML is always text, but the values stored in the ShakeCast database may be text or binary (integer, float, or date-time). A server that is storing data in the database attempts to cast the parameters from text into the datatype defined for the column in the database. If the data does not successfully convert into that format, then the script returns an error, which is written to the ShakeCast Status XML and returned to the requesting server.

8.5 Product File Exchanges

Some ShakeCast products consist of a single file; for example, an image of the Peak Ground Acceleration for an Event might be contained in a single .PNG file. Others consist of multiple files, such as a ShapeFiles set (a file set that contains GIS data), or a georeferenced TIFF file (which consists of a .TIF file and a .TFW file that contains geo-referencing information). Some ShakeMap products may also have an accompanying "manifest" or information file that provides further supporting information, in either a machine-readable or human-readable form.

Because many ShakeCast products consist of multiple parts, all products are always contained in a ZIP archive. The ZIP file preserves the file metadata (creation timestamp, for instance) and also serves to compress the file contents to improve file transfer times. Therefore, when a ShakeCast Server retrieves a ShakeCast Product File, it always expects a .ZIP file, and it always unzips its contents before storing it and handing it off to other processors.

All ShakeCast products are stored in a directory hierarchy, the root of which is specified as a configuration parameter when the ShakeCast system is initially installed.

[details tbs on how products are organized in the directories. By product? By shakemap?]

8.6 Grid File Exchanges

Grid Files are a special case of Product File Exchanges. The processing described in the previous section is applied to Grid Files, but then the data is extracted from the Grid File into the ShakeCast Database.

This processing is accomplished by the script `store_grid_records`. This script does the following operations:

- A Grid File may be based on an existing grid, or it may define a new grid. The script determines if this is a new grid or an existing grid by examining the grid geometry. The grid geometry is defined by parameters in the first record of a grid file. If the geometry matches that of an existing grid in the GRID table, then the grid number of the existing grid is used for subsequent steps. If the geometry does not match any existing grid, then a new grid is created.
- The values in a Grid File are stored in second normal form (a list of values for separate metrics at each grid cell). The values in the ShakeCast Database are stored in third normal form (each grid value for each metric is in a separate row). When a grid file is processed, the processor breaks the file into separate third normal form records and stores those records.
- If an error occurs on storing any one of the records, then the entire grid file is rejected (and not stored) for that one metric, but other metrics are stored.

Chapter 9 Invoking External Procedures from ShakeCast

[tbs]

9.1 Overview

[tbs]

9.2 External Script Environment

[tbs]

9.3 Environment Variables and Parameters

[tbs]

9.4 Returning Status Values

[tbs]

Chapter 10Testing ShakeCast

[tbs]

10.1 Overview

[tbs]

10.2 Server Self-Test Functions

[tbs]

10.3 Communication Tests

[tbs]

10.4 Processing Standard Test Messages

[tbs]

10.5 Test Logs and Test Error Reporting

[tbs]

Chapter 11ShakeCast Administration User Interface

[tbs]

11.1 Overview

[tbs]

11.2 Notification Requests and Notification Pages

[tbs]

11.3 User Administration Pages

[tbs]

11.4 System Configuration Pages

[tbs]

INDEX

Database Description for

ShakeCast

"ShakeCast: Delivering Earthquake Shaking Data to the People Who Need It"

Software Version 1.0
Documentation Version 1.0

July 2004

Gatekeeper Systems

Applications and Systems for the Internet

1010 East Union Street
Suite 205
Pasadena, CA 91106-1756

Phone: +1 626 449 8135
Fax: +1 626 440 1742

E-Mail: info@gatekeeper.com
URL: http://www.gatekeeper.com/

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure of this document or of the software described herein is governed by the terms of a License Agreement or, in the absence of an agreement, is subject to the restrictions stated in subparagraph (c) (1) of the Commercial Computer Software –Restricted Rights clause at FAR 52.227-19 or subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, as applicable. Contractor/Manufacturer is Gatekeeper Systems, 1010 E. Union St, Pasadena CA 91106, 626 449 8135, 800 424 3070, Info@Gatekeeper.com, http://www.gatekeeper.com/.

Form Number: GKS 2004-9

Table of Contents

Chapter 1 Introduction

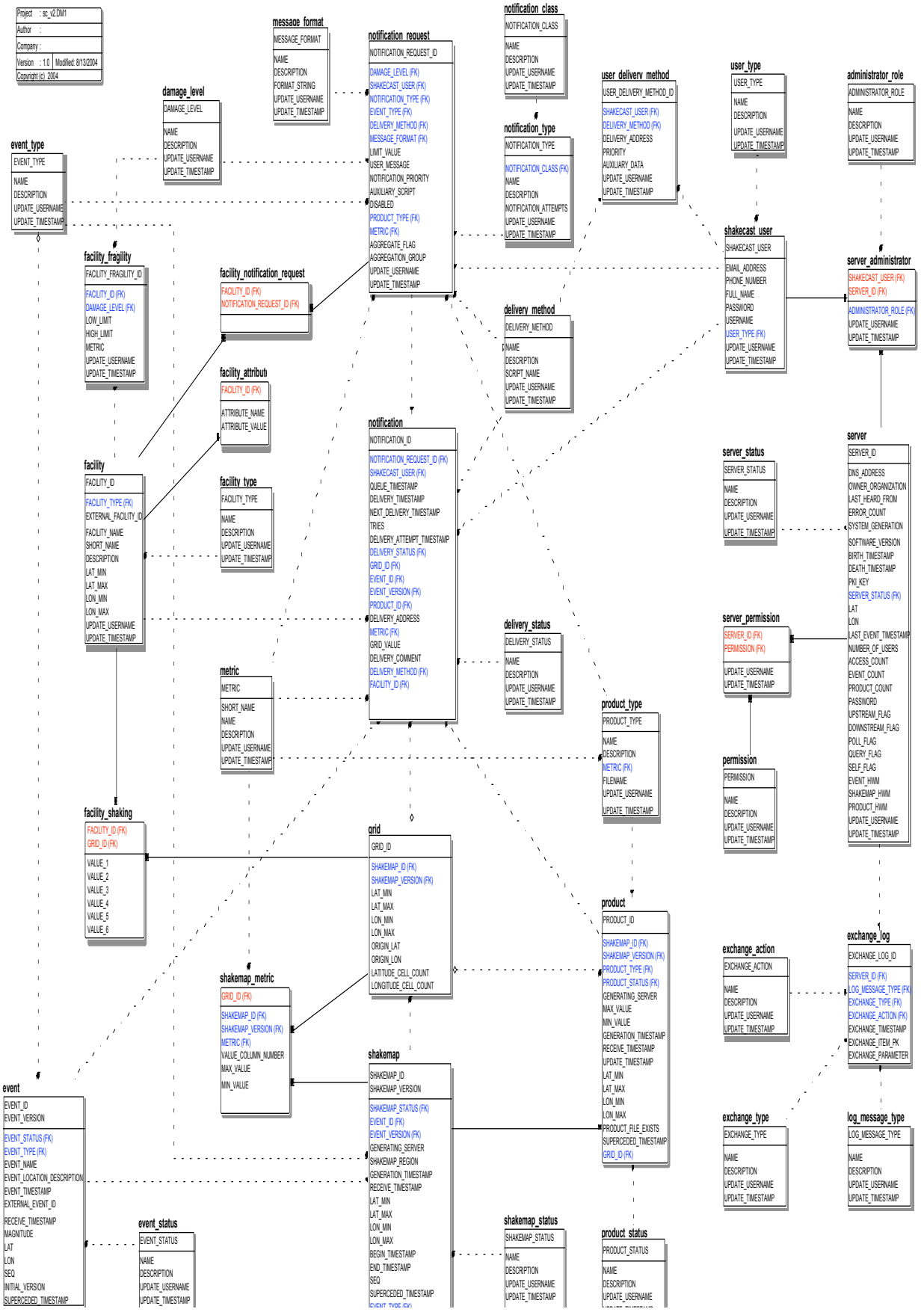
ShakeCast servers store much of the data used by the server in a relational database. The ShakeCast database contains information needed to interact with other ShakeCast Servers, data that will be presented to users, configuration information needed to perform notifications, and various other kinds of data.

The following paragraphs document the structure of the ShakeCast database using a standard entity-relationship modeling syntax.

The ShakeCast database is used in all elements of the ShakeCast system. Although some tables are used in many different stages of ShakeCast operation, the tables may be conveniently grouped as follows:

- Tables that define the ShakeCast system, the network of ShakeCast servers, exchanges between those servers, and the administrators who maintain that network
- Tables that define ShakeCast events (earthquakes), products, and related data
- Tables that are used to notify end users of earthquakes, shaking levels at particular facilities, and other ShakeCast events
- Miscellaneous operational tables

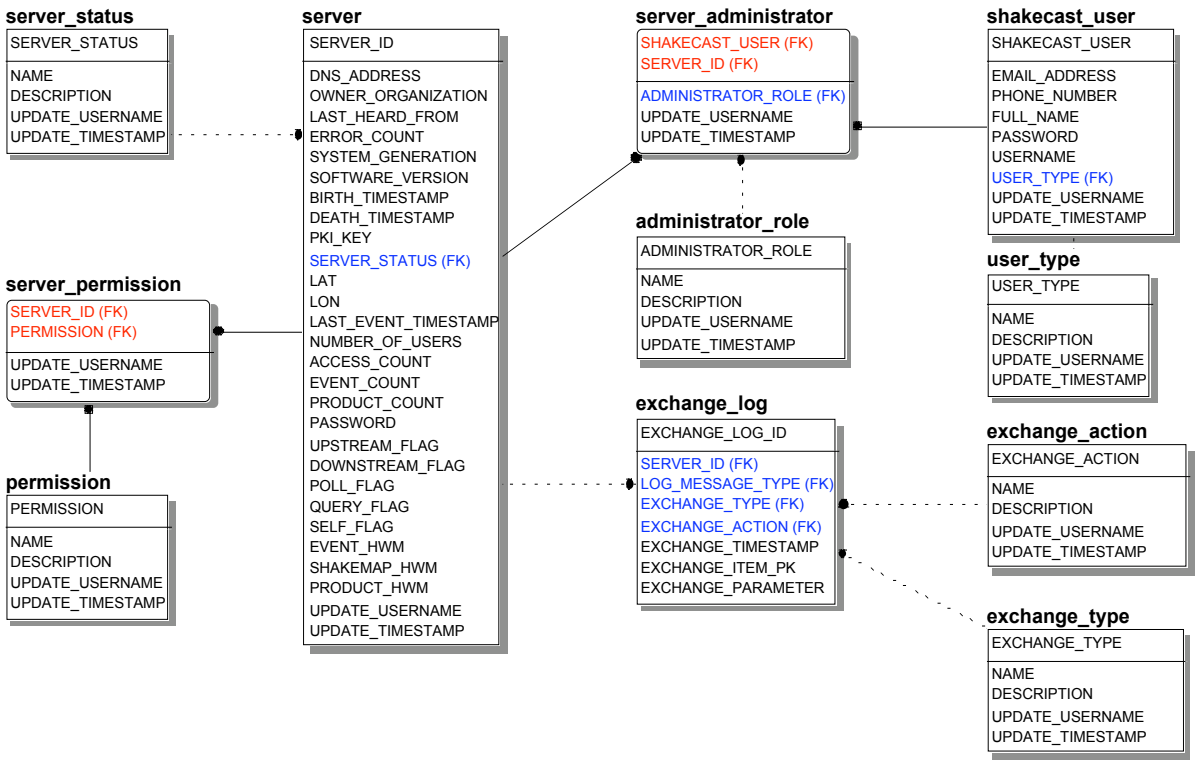
Each of these groups of tables is documented in the sections below.


Project : ss_v2.DM1
Author :
Company :
Version : 1.0 Modified: 8/13/2004
Copyright (c) 2004
damage_level
DAMAGE_LEVEL
NAME
DESCRIPTION
UPDATE_USERNAME
UPDATE_TIMESTAMP
messaae_format
MESSAGE_FORMAT
NAME
DESCRIPTION
FORMAT_STRING
UPDATE_USERNAME
UPDATE_TIMESTAMP
notification_request
NOTIFICATION_REQUEST_ID
DAMAGE_LEVEL (FK)
SHAKECAST_USER (FK)
NOTIFICATION_TYPE (FK)
EVENT_TYPE (FK)
DELIVERY_METHOD (FK)
MESSAGE_FORMAT (FK)
LIMIT_VALUE
USER_MESSAGE
NOTIFICATION_PRIORITY
AUXILIARY_SCRIPT
DISABLED
PRODUCT_TYPE (FK)
METRIC (FK)
AGGREGATE_FLAG
AGGREGATION_GROUP
UPDATE_USERNAME
UPDATE_TIMESTAMP
notification_class
NOTIFICATION_CLASS
NAME
DESCRIPTION
UPDATE_USERNAME
UPDATE_TIMESTAMP
notification_type
NOTIFICATION_TYPE
NOTIFICATION_CLASS (FK)
NAME
DESCRIPTION
NOTIFICATION_ATTEMPTS
UPDATE_USERNAME
UPDATE_TIMESTAMP
user_deliverv method
USER_DELIVERY_METHOD_ID
SHAKECAST_USER (FK)
DELIVERY_METHOD (FK)
DELIVERY_ADDRESS
PRIORITY
AUXILIARY_DATA
UPDATE_USERNAME
UPDATE_TIMESTAMP
user_type
USER_TYPE
NAME
DESCRIPTION
UPDATE_USERNAME
UPDATE_TIMESTAMP
administrator_role
ADMINISTRATOR_ROLE
NAME
DESCRIPTION
UPDATE_USERNAME
UPDATE_TIMESTAMP
event_type
EVENT_TYPE
NAME
DESCRIPTION
UPDATE_USERNAME
UPDATE_TIMESTAMP
facility_fragility
FACILITY_FRAGILITY_ID
FACILITY_ID (FK)
DAMAGE_LEVEL (FK)
LOW_LIMIT
HIGH_LIMIT
METRIC
UPDATE_USERNAME
UPDATE_TIMESTAMP
facility_notification_request
FACILITY_ID (FK)
NOTIFICATION_REQUEST_ID (FK)
facility_attributi
FACILITY_ID (FK)
ATTRIBUTE_NAME
ATTRIBUTE_VALUE
delivery_method
DELIVERY_METHOD
NAME
DESCRIPTION
SCRIPT_NAME
UPDATE_USERNAME
UPDATE_TIMESTAMP
shakecast_user
SHAKECAST_USER
EMAIL_ADDRESS
PHONE_NUMBER
FULL_NAME
PASSWORD
USERNAME
USER_TYPE (FK)
UPDATE_USERNAME
UPDATE_TIMESTAMP
server_administrator
SHAKECAST_USER (FK)
SERVER_ID (FK)
ADMINISTRATOR_ROLE (FK)
UPDATE_USERNAME
UPDATE_TIMESTAMP
facility
FACILITY_ID
FACILITY_TYPE (FK)
EXTERNAL_FACILITY_ID
FACILITY_NAME
SHORT_NAME
DESCRIPTION
LAT_MIN
LAT_MAX
LON_MIN
LON_MAX
UPDATE_USERNAME
UPDATE_TIMESTAMP
facility_type
FACILITY_TYPE
NAME
DESCRIPTION
UPDATE_USERNAME
UPDATE_TIMESTAMP
notification
NOTIFICATION_ID
NOTIFICATION_REQUEST_ID (FK)
SHAKECAST_USER (FK)
QUEUE_TIMESTAMP
DELIVERY_TIMESTAMP
NEXT_DELIVERY_TIMESTAMP
TRIES
DELIVERY_ATTEMPT_TIMESTAMP
DELIVERY_STATUS (FK)
GRID_ID (FK)
EVENT_ID (FK)
EVENT_VERSION (FK)
PRODUCT_ID (FK)
DELIVERY_ADDRESS
METRIC (FK)
GRID_VALUE
DELIVERY_COMMENT
DELIVERY_METHOD (FK)
FACILITY_ID (FK)
server_status
SERVER_STATUS
NAME
DESCRIPTION
UPDATE_USERNAME
UPDATE_TIMESTAMP
server
SERVER_ID
DNS_ADDRESS
OWNER_ORGANIZATION
LAST_HEARD_FROM
ERROR_COUNT
SYSTEM_GENERATION
SOFTWARE_VERSION
BIRTH_TIMESTAMP
DEATH_TIMESTAMP
PKI_KEY
SERVER_STATUS (FK)
LAT
LON
LAST_EVENT_TIMESTAMP
NUMBER_OF_USERS
ACCESS_COUNT
EVENT_COUNT
PRODUCT_COUNT
PASSWORD
UPSTREAM_FLAG
DOWNSTREAM_FLAG
POLL_FLAG
QUERY_FLAG
SELF_FLAG
EVENT_HWM
SHAKEMAP_HWM
PRODUCT_HWM
UPDATE_USERNAME
UPDATE_TIMESTAMP
delivery_status
DELIVERY_STATUS
NAME
DESCRIPTION
UPDATE_USERNAME
UPDATE_TIMESTAMP
server_permission
SERVER_ID (FK)
PERMISSION (FK)
UPDATE_USERNAME
UPDATE_TIMESTAMP
metric
METRIC
SHORT_NAME
NAME
DESCRIPTION
UPDATE_USERNAME
UPDATE_TIMESTAMP
product_type
PRODUCT_TYPE
NAME
DESCRIPTION
METRIC (FK)
FILENAME
UPDATE_USERNAME
UPDATE_TIMESTAMP
permission
PERMISSION
NAME
DESCRIPTION
UPDATE_USERNAME
UPDATE_TIMESTAMP
facility_shaking
FACILITY_ID (FK)
GRID_ID (FK)
VALUE_1
VALUE_2
VALUE_3
VALUE_4
VALUE_5
VALUE_6
grid
GRID_ID
SHAKEMAP_ID (FK)
SHAKEMAP_VERSION (FK)
LAT_MIN
LAT_MAX
LON_MIN
LON_MAX
ORIGIN_LAT
ORIGIN_LON
LATITUDE_CELL_COUNT
LONGITUDE_CELL_COUNT
product
PRODUCT_ID
SHAKEMAP_ID (FK)
SHAKEMAP_VERSION (FK)
PRODUCT_TYPE (FK)
PRODUCT_STATUS (FK)
GENERATING_SERVER
MAX_VALUE
MIN_VALUE
GENERATION_TIMESTAMP
RECEIVE_TIMESTAMP
UPDATE_TIMESTAMP
LAT_MIN
LAT_MAX
LON_MIN
LON_MAX
PRODUCT_FILE_EXISTS
SUPERCEDED_TIMESTAMP
GRID_ID (FK)
exchange_log
EXCHANGE_LOG_ID
SERVER_ID (FK)
LOG_MESSAGE_TYPE (FK)
EXCHANGE_TYPE (FK)
EXCHANGE_ACTION (FK)
EXCHANGE_TIMESTAMP
EXCHANGE_ITEM_PK
EXCHANGE_PARAMETER
exchange_action
EXCHANGE_ACTION
NAME
DESCRIPTION
UPDATE_USERNAME
UPDATE_TIMESTAMP
shakemap_metric
GRID_ID (FK)
SHAKEMAP_ID (FK)
SHAKEMAP_VERSION (FK)
METRIC (FK)
VALUE_COLUMN_NUMBER
MAX_VALUE
MIN_VALUE
shakemap
SHAKEMAP_ID
SHAKEMAP_VERSION
SHAKEMAP_STATUS (FK)
EVENT_ID (FK)
EVENT_VERSION (FK)
GENERATING_SERVER
SHAKEMAP_REGION
GENERATION_TIMESTAMP
RECEIVE_TIMESTAMP
LAT_MIN
LAT_MAX
LON_MIN
LON_MAX
BEGIN_TIMESTAMP
END_TIMESTAMP
SEQ
SUPERCEDED_TIMESTAMP
exchange_type
EXCHANGE_TYPE
NAME
DESCRIPTION
UPDATE_USERNAME
UPDATE_TIMESTAMP
log_message_type
LOG_MESSAGE_TYPE
NAME
DESCRIPTION
UPDATE_USERNAME
UPDATE_TIMESTAMP
event
EVENT_ID
EVENT_VERSION
EVENT_STATUS (FK)
EVENT_TYPE (FK)
EVENT_NAME
EVENT_LOCATION_DESCRIPTION
EVENT_TIMESTAMP
EXTERNAL_EVENT_ID
RECEIVE_TIMESTAMP
MAGNITUDE
LAT
LON
SEQ
INITIAL_VERSION
SUPERCEDED_TIMESTAMP
event_status
EVENT_STATUS
NAME
DESCRIPTION
UPDATE_USERNAME
UPDATE_TIMESTAMP
shakemap_status
SHAKEMAP_STATUS
NAME
DESCRIPTION
UPDATE_USERNAME
UPDATE_TIMESTAMP
product_status
PRODUCT_STATUS
NAME
DESCRIPTION
UPDATE_USERNAME

Chapter 2 Database Tables for Servers and System Administrators

The first group of tables defines ShakeCast servers and the organizations and users responsible for those servers. These tables also define the rights that each server has to communicate with the current server.

Server Tables



2.1 SERVER

The SERVER table contains information about every ShakeCast server that this server communicates with and every server that creates data used by this server.

Column	Datatype	Description
SERVER_ID	Integer	A globally unique identifier for the server. Server IDs are currently assigned by contacting the ShakeCast Registrar, at register@shakecast.org .
ACCESS_COUNT	Integer	The number of user accesses to ShakeCast data and reports. (Not currently used.)
BIRTH_TIMESTAMP	Datetime	The earliest known date of operation of the server. (Not currently used.)
DEATH_TIMESTAMP	Datetime	The last known date of operation of the server. (Not currently used.)
DNS_ADDRESS	String	DNS address (name) of the server. This may also be the IP address, if necessary.
DOWNSTREAM_FLAG	Integer	This value is non-null if the server is downstream of this server; that is, if the local server feeds data to the server defined in this record of the SERVERS table.
ERROR_COUNT	Integer	The number of errors recorded since the last time a successful exchange took place with this server. (Not currently used.)
EVENT_COUNT	Integer	The number of ShakeCast events stored in the ShakeCast server.
EVENT_HWM	Integer	The latest (greatest) EVENT.SEQ sent to the server listed in this record.
LAST_EVENT_TS	Datetime	The date and time of the last update to the event table on this server.
LAST_HEARD_FROM	Datetime	The date and time (GMT) that the server was last heard from by this server
LAT	Float	Latitude of this server
LON	Float	Longitude of this server
NUMBER_OF_USERS	Integer	The number of unique user IDs that have access to ShakeCast data. (Not currently used.)
OWNER_ORGANIZATION	String	The name of the organization that operates this server
PASSWORD	String	The password used by the local server to log into this remote server.
PKI _KEY	String	The public key associated with this server. (Not currently used.)
POLL_FLAG	Integer	This value is non-null if the given server should be polled by the local server. (See also

		QUERY_FLAG.)
PRODUCT_COUNT	Integer	The number of ShakeCast products stored in the ShakeCast server. (Not currently used.)
PRODUCT_HWM	Integer	The latest (greatest) PROCUCT.PRODUCT_ID sent to the server listed in this record.
QUERY_FLAG	Integer	This value is non-null if the given server should be allowed to poll the local server. (See also POLL_FLAG.)
SELF_FLAG	Integer	This value is non-null if the record refers to the local server. This is how the local server stores and locates its server characteristics. Only one record may have the SELF_FLAG set.
SHAKEMAP_HWM	Integer	The latest (greatest) SHAKEMAP.SEQ sent to the server listed in this record.
SOFTWARE_VERSION	String	The version string for the ShakeCast software this server is currently running.
STATUS	Integer	The current operational status of the server
SYSTEM_GENERATION	Integer	The number of times that this server has restarted the ShakeCast software. This value is incremented each time a server is restarted.
UPDATE_TIMESTAMP	Date	Last time this record was changed.
UPDATE_USERNAME	String	Local database user who performed the last update.
UPSTREAM_FLAG	Integer	This value is non-null if the server is upstream of this server; that is, if the local server feeds data to the server defined in this record of the SERVERS table.

2.2 SERVER_STATUS

The SERVER_STATUS table defines the valid codes for the STATUS column of the SERVER table. Typical values for the SERVER_STATUS column include “ALIVE”, “UNKNOWN”, and “REMOVED”.

Column	Datatype	Description
SERVER_STATUS	String	Type code for server status
NAME	String	Status name
DESCRIPTION	String	Human readable meaning for this value
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

2.3 SERVER_PERMISSION

The SERVER_PERMISSION table defines the valid codes for the PERMISSION column of the SERVER table. (This table is not currently used.)

Column	Datatype	Description
SERVER_ID	Integer	ID of the server for which permission is to be established.
PERMISSION	Integer	Code for the permission granted to this server.
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

2.4 PERMISSION

The PERMISSION table defines which activities are valid when communicating with the each server in the ShakeCast network. (This table is not currently used.)

Column	Datatype	Description
PERMISSION	Integer	Type code for operations permitted in interactions between the local server and the server indicated.
NAME	String	Status name
DESCRIPTION	String	Human readable meaning for this value
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

2.5 SERVER_ADMINISTRATOR

The SERVER_ADMINISTRATOR table contains links to the ShakeCast users who administer ShakeCast servers. In most cases, a particular individual will be associated with only a single ShakeCast server. (This table is not currently used.)

Column	Datatype	Description
SERVER_ID	Integer	The globally unique ID for the server with which this administrator is associated.
SHAKECAST_USER	String	The user ID of the ShakeCast user who is an administrator for this server.
ADMINISTRATOR_ROLE	Integer	The role that this person serves in the administration of this server.
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

2.6 ADMINISTRATOR_ROLE

The ADMINISTRATOR_ROLE table defines the valid roles for administrators. (This table is not currently used.)

Column	Datatype	Description
ADMNISTRATOR_ROLE	Integer	Type code for the administrator role
NAME	String	Name of role
DESCRIPTION	String	Human readable meaning for this value
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

2.7 EXCHANGE_LOG

A product is moved between ShakeCast servers via an *exchange*. The EXCHANGE_LOG table records salient information about this activity. (This table is not currently used.)

Column	Datatype	Description
EXCHANGE_LOG_ID	Integer	Locally unique identifier for each exchange operation
SERVER_ID	Integer	Globally unique ID of the server on the other end of the exchange. Foreign key to the SERVER table.
EXCHANGE_TYPE	String	Type of exchange activity. Foreign key to the EXCHANGE_TYPE table
EXCHANGE_ACTION	String	The action taken during this exchange. Foreign key to the EXCHANGE_ACTION table.
EXCHANGE_TIMESTAMP	Datetime	The date and time the exchange was initiated
EXCHANGE_ITEM_PK	Integer	A general parameter to hold the primary key value of the item exchanged. The interpretation of this value depends on the EXCHANGE_TYPE.
EXCHANGE_PARAMETER	String	A general parameter to contain additional data about the exchange. The interpretation of this column depends on the value of EXCHANGE_TYPE.

2.8 EXCHANGE_TYPE

The EXCHANGE_TYPE table defines the valid codes for the EXCHANGE_TYPE column of the EXCHANGE table. Valid exchange types include “Get”, “Put”, and “Heartbeat”.

Column	Datatype	Description
EXCHANGE_TYPE	String	Type code for exchanges
NAME	String	Exchange type name
DESCRIPTION	String	Human readable meaning for this value
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

2.9 EXCHANGE_ACTION

The EXCHANGE_ACTION table defines the disposition of the exchange. (This table is not currently used.)

Column	Datatype	Description
EXCHANGE_TYPE	Integer	Type code for exchanges
NAME	String	Exchange type name
DESCRIPTION	String	Human readable meaning for this value
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

2.10 LOG_MESSAGE_TYPE

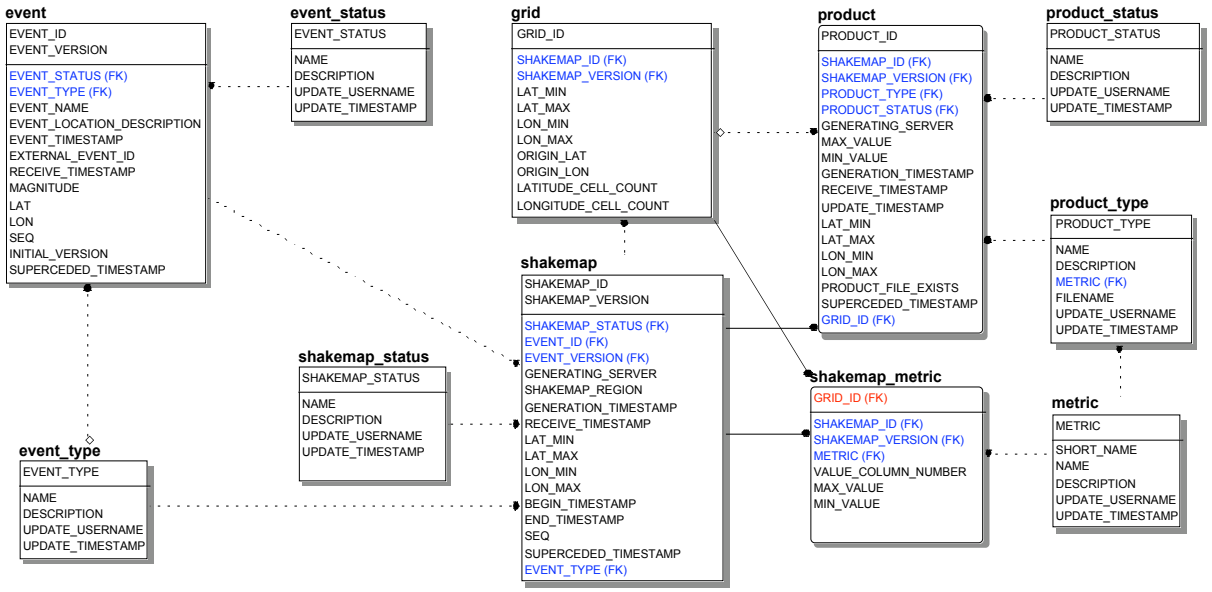
The LOG_MESSAGE_TYPE table defines the type of record stored in the ECHANGE_LOG table. (This table is not currently used.)

Column	Datatype	Description
LOG_MESSAGE_TYPE	Integer	Type code for log records.
NAME	String	Exchange type name
DESCRIPTION	String	Human readable meaning for this value
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

Chapter 3 Database Tables for Events, ShakeMaps, and Products

The following tables contain the data needed to represent Events, ShakeMaps, and ShakeMap Products.

Events, ShakeMaps and Products



3.1 EVENT

The EVENT table contains information about seismic events. The EVENT_ID is a globally unique, permanently assigned identifier associated with a single seismic event.

Column	Datatype	Description
EVENT_ID	Integer	A globally unique identifier for the event
EVENT_LOCATION_DESCRIPTION	String	Human-readable location description (i.e., “8.1mi of Pasadena, CA”)

		Pasadena, CA")
EVENT_NAME	String	Name of event (i.e., "Northridge")
EVENT_STATUS	Integer	The status of this event (active, cancelled, test, archive, etc.)
EVENT_TIMESTAMP	Datetime	The date and time (GMT) that the event occurred
EVENT_TYPE	Integer	Foreign key to the EVENT_TYPE table
EVENT_VERSION	Integer	A sequential version number. The latest version is the most current representation of the data about this event.
EXTERNAL_EVENT_ID	String	Event ID in a locally-defined external server
LAT	Float	The latitude of the point representation of the event
LON	Float	The longitude of the point representation of the event
RECEIVE_TIMESTAMP	Datetime	The time this event information was first received on this server
SUPERCEDED_TIMESTAMP	Datetime	The time this event was superceded by a newer version or by a different event.
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

3.2 EVENT_STATUS

The EVENT_STATUS table defines the valid codes for the STATUS column of the EVENT table. The event status is typically one of the following: "normal", "cancelled", "incomplete", or "released".

Column	Datatype	Description
EVENT_STATUS	String	Type code for event status
NAME	String	Status name
DESCRIPTION	String	Human readable meaning for this value
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

3.3 EVENT_TYPE

The EVENT_TYPE table defines the valid codes for the EVENT_TYPE column of the EVENT table. The event type is typically one of the following: "actual", "test", or "scenario".

Column	Datatype	Description
EVENT_TYPE	Integer	Code for the event type
NAME	String	Event type name
DESCRIPTION	String	Human readable meaning for this value
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

3.4 SHAKEMAP

The SHAKEMAP table describes a single ShakeMap. A ShakeMap is produced outside of the ShakeCast system, by a ShakeMap Server. ShakeMaps are associated with zero or more events and with zero or more Products.

Column	Datatype	Description
SHAKEMAP_ID	Integer	Uniquely defines a ShakeMap.
SHAKEMAP_VERSION	Integer	ShakeMap versions start with one and are increased each time the ShakeMap is updated. Only the latest version of a ShakeMap is correct. The primary key of this table is the SHAKEMAP_ID plus VERSION.
EVENT_ID	Integer	The EVENT_ID and EVENT_VERSION of the event for which this ShakeMap was created.
EVENT_VERSION	Integer	
GENERATION_TIMESTAMP	Datetime	The time this ShakeMap was first created
GENERATING_SERVER	Integer	The unique ID of a ShakeCast server
GRID_ID	Integer	Foreign key to the Grid Table, which defines the bounding box and cell size of the grid that applies to this ShakeMap
BEGIN_TIMESTAMP	Datetime	The beginning date and time of the period covered by this ShakeMap. (Not currently used.)
END_TIMESTAMP	Datetime	The ending date and time of the period covered by this ShakeMap. (Not currently used.)
LAT_MIN	Float	Bounding box of the area covered by this ShakeMap
LAT_MAX	Float	
LON_MIN	Float	
LON_MAX	Float	
RECEIVE_TIMESTAMP	Datetime	The timestamp of the last time this ShakeMap was received from an upstream server

SEQ	Integer	The unique local sequence number of this event on this server. This sequence number is used to keep track of the high water mark for event records.
SHAKEMAP_REGION	String	The ShakeMap region that generated this ShakeMap. (Not currently used.)
SHAKEMAP_STATUS	Integer	The status of this ShakeMap (active, cancelled, test, archive, etc.)
SUPERCEDED_TIMESTAMP	Datetime	Time this ShakeMap was superceded by a new version or by a different ShakeMap.
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

3.5 SHAKEMAP_STATUS

The SHAKEMAP_STATUS table defines the valid codes for the STATUS column of the SHAKEMAP table. The ShakeMap status is typically one of the following: “normal”, “cancelled”, “incomplete”, “released”, and “reviewed”.

Column	Datatype	Description
SHAKEMAP_STATUS	String	Type code for ShakeMap status
NAME	String	Status name
DESCRIPTION	String	Human readable meaning for this value
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

3.6 SHAKEMAP_REGION

The SHAKEMAP_REGION table defines the ShakeMap Regions. Regions might include Southern California, Northern California, and Utah. (This table is not currently used.)

Column	Datatype	Description
SHAKEMAP_REGION	String	Type code for ShakeMap Region
NAME	String	Region Name
DESCRIPTION	String	Human readable meaning for this value
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

3.7 PRODUCT

This table contains information about each ShakeMap product. A product is a single metric (PRODUCT_METRIC column) for a single ShakeMap (SHAKEMAP_ID column) in a single format (PRODUCT_FORMAT column).

Column	Datatype	Description
PRODUCT_ID	Integer	A globally unique identifier for this product.
GENERATING_SERVER	Integer	The unique ID of a ShakeCast server
GENERATION_TIMESTAMP	Datetime	The time this ShakeMap was first created.
GRID_ID	Integer	A foreign key to the grid layout that is used when this data is represented relationally
LAT_MAX	Float	Bounding box of the area covered by this product
LAT_MIN	Float	
LON_MAX	Float	
LON_MIN	Float	
MAX_VALUE	Integer	The maximum value for METRIC contained within this product.
METRIC	Integer	The shaking metric represented in this product, such as “acceleration”, “instrumental intensity”, etc.
MIN_VALUE	Integer	The minimum value for METRIC contained within this product.
PRODUCT_FILE_EXISTS	Integer	A flag that indicates that the file has been successfully transferred to this server. (The product record may exist before the product is actually available on this server.)
PRODUCT_TYPE	Integer	The type of this product, e.g., “GRID”, “HAZUS”, “CONT_PGA”, “INTEN_JPG”, etc.
PRODUCT_STATUS	Integer	The status of this product (active, cancelled, test, archive, etc.)
RECEIVE_TIMESTAMP	Datetime	The time this product information was first received on this server
SHAKEMAP_ID	Integer	The ID and VERSION of the ShakeMap associated with this product.
SHAKEMAP_VERSION	Integer	
SOURCE_FILENAME	String	The name of the file in the local filesystem that contains this product. This may be a single file, a directory name, or the name of an archive file containing multiple files (e.g., a ZIP file)
SUPERCEDED_BY	Integer	The PRODUCT_ID of a product that supercedes this one.
SUPERCEDES	Integer	The PRODUCT_ID of a product that this product supercedes. If this product supercedes more than one product,

		only one is listed here.
UPDATE_TIMESTAMP	Datetime	The last time this table or the ShakeMap itself was updated.
VERSION	Integer	Product versions start with one and are increased each time the product is updated. Only the latest version of a product is valid and correct. The primary key of this table is the PRODUCT_ID plus VERSION.

3.8 PRODUCT_STATUS

The PRODUCT_STATUS table defines the valid codes for the STATUS column of the PRODUCT table. The product status is typically one of the following: “released”, “reviewed”, and “cancelled”.

Column	Datatype	Description
PRODUCT_STATUS	String	Type code for product status
NAME	String	Status name
DESCRIPTION	String	Human readable meaning for this value
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

3.9 METRIC

The METRIC table defines the valid codes for the METRIC column of the PRODUCT table. Shaking metrics include “peak spectral acceleration”, “maximum velocity”, “instrumental intensity”, and so on.

Column	Datatype	Description
METRIC_ID	Integer	Unique identifier for this metric.
NAME	String	Product type name
SHORT_NAME	String	Abbreviation for metric name
DESCRIPTION	String	Human readable meaning of this metric
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

3.10 PRODUCT_TYPE

The PRODUCT_TYPE table defines the valid codes for the PRODUCT_FORMAT column of the PRODUCT table. Product Types describe both the data in a product (e.g., "TV Map") and the file format (e.g., ".zip" or ".jpg"). For example, the PRODUCT_TYPE might include "CONT_PGA" for shapefile contours of PGA, "CONT_PGV" for shapefile contours of PGV, GRID for a ShakeMap grid file, HAZUS for a HAZUS-format file, "INTEN_JPG" for a Shaking Intensity JPG image and "INTEN_PS" for a ShakeMap Intensity PostScript image file.

Column	Datatype	Description
PRODUCT_TYPE	Integer	Format code for products
NAME	String	Product format name
DESCRIPTION	String	Human readable meaning for this value
FILENAME	String	The string used to construct the filename for this product type. This information is concatenated with the short name of the metric to construct a full filename to store the product.
METRIC_ID	Integer	Foreign key to the Metric table. Defines the metric contained in this Product Type. Not all Product Types have a metric (e.g., GRID_XYZ products contain a mix of products).
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

3.11 GRID

A ShakeMap Grid is a rectangular array of cells, each of fixed size (in terms of latitude and longitude). The GRID table describes this array.

Column	Datatype	Description
GRID_ID	Integer	Unique ID for this grid.
ORIGIN_LATITUDE	Float	Latitude of the origin of the grid
ORIGIN_LONGITUDE	Float	Longitude of the origin of the grid
LATITUDE_CELL_COUNT	Integer	Number of cells in the direction of latitude (north-south).
LONGITUDE_CELL_COUNT	Integer	Number of cells in the direction of longitude (east-west)
LAT_MIN	Float	Bounding box of the area covered by this product
LAT_MAX	Float	
LON_MIN	Float	
LON_MAX	Float	

3.12 FACILITY_SHAKING

This table contains the maximal values for each of up to six metrics associated with a particular facility and a particular grid. That is, these are the "maximal shaking values" for this facility associated with this grid. Since a grid is a particular representation of a ShakeMap, the FACILITY_SHAKING table can be thought of as the shaking values at a particular facility associated with a particular ShakeMap.

Column	Datatype	Description
FACILITY_ID	Integer	Foreign key to the FACILITY table. Defines the facility for which these values apply.
GRID_ID	Integer	Foreign key to the Grid Table. Defines the GRID, and indirectly the ShakeMap, for which these values apply.
VALUE_1	Float	The values for this grid cell for each metric generated by this ShakeMap are stored in these columns. The VALUE_COLUMN_NAME table SHAKEMAP_METRIC defines which column contains a particular metric. Additional columns may be added in the future. No assumption should be made about the order in which metrics appear in these columns.
VALUE_2	Float	
VALUE_3	Float	
VALUE_4	Float	
VALUE_5	Float	
VALUE_6	Float	

3.13 SHAKEMAP_METRIC

This table defines the metrics that are available from a particular ShakeMap. It also defines which column in the FACILITY_SHAKING table contains the values for a particular metric for this ShakeMap. Software can examine the SHAKEMAP_METRIC table to find the maximal values for any particular metric for any particular ShakeMap (and, by extension, for any particular event). Alternately, software can examine the table to find all of the maximal values for all metrics for a particular ShakeMap, or for a particular metric.

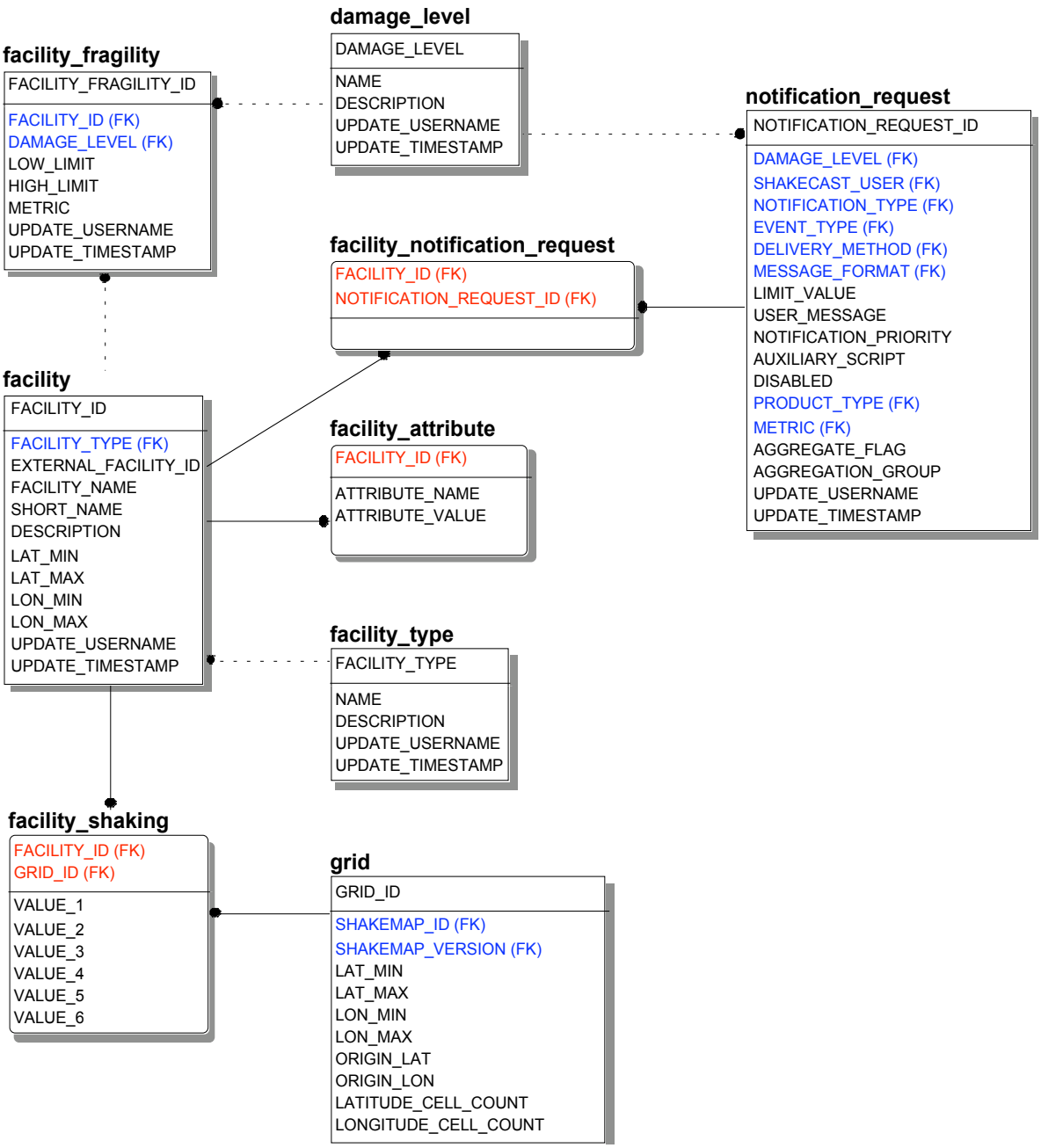
Column	Datatype	Description
SHAKEMAP_ID	Integer	Foreign key to the ShakeMap Table. Defines the ShakeMap for which this Grid Value applies.
SHAKEMAP_VERSION	Integer	
METRIC_ID	Integer	Foreign key to the Metric Table
VALUE_COLUMN_NUMBER	Integer	The VALUE_x column number (i.e., VALUE_1, VALUE_2) in the FACILITY_SHAKING table that contains this metric for this ShakeMap.
MIN_VALUE	Float	The minimum value of this metric in this ShakeMap (not including possible null values).

MAX_VALUE	Float	The maximum value of this metric in this ShakeMap (not including possible null values).
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

Chapter 4 Database Tables for Facilities

The ShakeCast System uses the following tables to define facilities and their attributes.

Facilities



4.1 FACILITY

A ShakeCast Facility is a specific structure (e.g., bridge, school, pumping station, etc.) at a specific location or a region. The location may be defined by a latitude/longitude for "point" facilities, or by a bounding box for non-point facilities. Note that the FACILITY table may be used to define a physical facility (e.g., a freeway overpass) or a region (e.g., a county). How-

ever, when used to specify a region, the FACILITY table must express the region as a rectangular bounding box.

Column	Datatype	Description
FACILITY_ID	Integer	A locally-unique primary key
NAME	String	Name of the facility
SHORT_NAME	String	Abbreviated name for facility
DESCRIPTION	String	A free text description or comment
EXTERNAL_FACILITY_ID	String	A text string that contains a facility identifier in the format that can be used by an external system. For example, this might contain a “Reservoir Number”, or “Pipeline Segment Number”.
FACILITY_TYPE	Integer	The foreign key to the FACILITY_TYPE table that defines valid facility types for your organization.
LAT_MIN	Float	Bounding box of the area covered by this facility
LAT_MAX	Float	
LON_MIN	Float	
LON_MAX	Float	
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

4.2 FACILITY_TYPE

The FACILITY_TYPE table defines the valid codes for the FACILITY_TYPE column of the FACILITY table. The facility type is a business definition unique to each installation of ShakeCast, but usually contains things such as “Bridge”, “Reservoir”, “Substation”, or “Pipeline”.

Column	Datatype	Description
FACILITY_TYPE	Integer	Code for the facility type
NAME	String	Facility type name
DESCRIPTION	String	Human readable meaning for this value
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

4.3 FACILITY_ATTRIBUTE

The FACILITY_ATTRIBUTE table stores arbitrary, user-defined tuples that describe a facility. These values are used when grouping facilities for purposes of defining notification requests, reports, or other actions on facilities.

Column	Datatype	Description
FACILITY_ID	Integer	Unique ID for the facility
ATTRIBUTE_NAME	String	Arbitrary attribute name string, user defined
ATTRIBUTE_VALUE	String	Arbitrary attribute value, user defined
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

4.4 FACILITY_FRAGILITY

The FACILITY_FRAGILITY table defines the facility thresholds for a facility for each fragility level and each product metric.

Column	Datatype	Description
FACILITY_FRAGILITY_ID	Integer	Auto-generated unique primary key of the FACILITY_FRAGILITY record.
FACILITY_ID	Integer	Foreign key to the FACILITY table. Contains the facility for which this FACILITY_FRAGILITY applies.
DAMAGE_LEVEL	Integer	Foreign key to the DAMAGE_LEVEL table.
METRIC	Integer	Foreign key to the METRIC table. Contains the metric for which this FACILITY_FRAGILITY applies
LOW_LIMIT	Float	Low limit of this fragility threshold at this facility for this metric and damage level.
HIGH_LIMIT	Float	High limit of this fragility threshold at this facility for this metric and damage level.
UPDATE_TIMESTAMP	Float	Last time this record was updated
UPDATE_USERNAME	String	Local username of the user who last updated this record

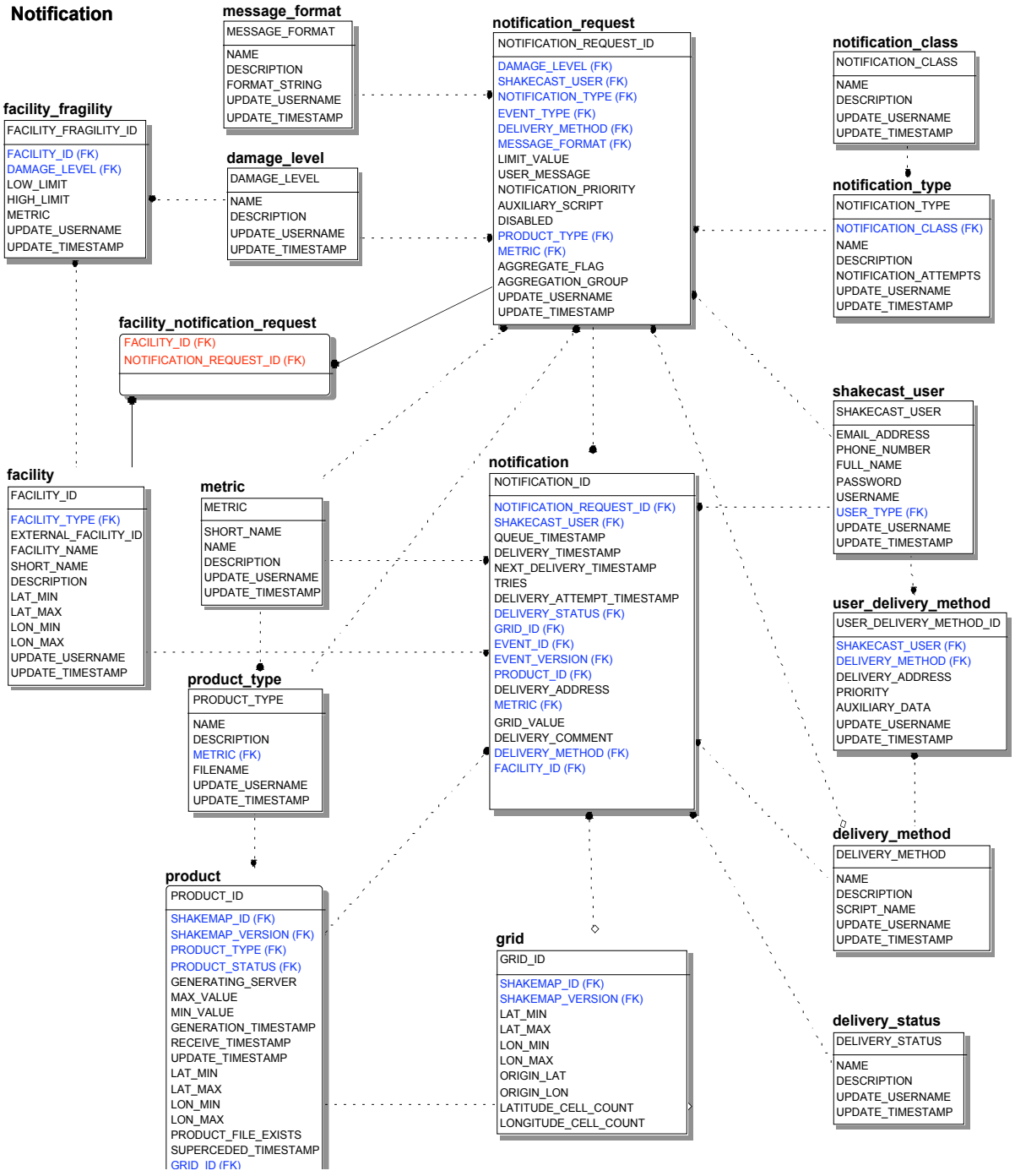
4.5 FACILITY_SHAKING

The FACILITY_SHAKING table stores the actual shaking levels associated with a specific facility and a specific ShakeMap Grid. The shaking levels are stored in the values VALUE_1 through VALUE_6, each corresponding to a different ShakeMap Metric. The column that is used for storing each Metric are defined in the SHAKEMAP_METRIC table.

Column	Datatype	Description
FACILITY_ID	Integer	Unique ID for the facility
GRID_ID	Integert	Unique ID for the ShakeMap Grid
VALUE_1	Float	Value for a particular metric from a particular ShakeMap Grid at the location of this Facility
VALUE_2	Float	Value for a particular metric from a particular ShakeMap Grid at the location of this Facility
VALUE_3	Float	Value for a particular metric from a particular ShakeMap Grid at the location of this Facility
VALUE_4	Float	Value for a particular metric from a particular ShakeMap Grid at the location of this Facility
VALUE_5	Float	Value for a particular metric from a particular ShakeMap Grid at the location of this Facility
VALUE_6	Float	Value for a particular metric from a particular ShakeMap Grid at the location of this Facility

Chapter 5 Database Tables for Notification

In addition to tables defined elsewhere, the ShakeCast system uses the following tables to compute and execute notification operations.



5.1 NOTIFICATION_REQUEST

A ShakeCast notification event is generated for each NOTIFICATION_REQUEST where the value in a grid cell exceeds the corresponding value in the request.

Column	Datatype	Description

NOTIFICATION_REQUEST_ID	Integer	Locally generated primary key
AGGREGATE_FLAG	Integer	This value is non-null if the notification is to be aggregated.
AGGREGATION_GROUP	String	When a notification is to be aggregated together with other notifications (i.e., delivered together), this value is common to all the notifications that are to be delivered together.
AUXILIARY_SCRIPT	String	Name of script to run to execute this notification request. (See the ShakeCast Template Manual for more information on the calling of auxiliary scripts.)
DAMAGE_LEVEL	Integer	Foreign key to the DAMAGE_LEVEL table. Defines the damage level used in the notification computation.
DELIVERY_METHOD	Integer	Foreign key to the DELIVERY_METHOD table. Defines how the notification is to be delivered to the user.
DISABLED	Integer	Non-null if this notification request is no longer to be honored by the notification software.
EVENT_TYPE	Integer	Foreign key to the EVENT_TYPE table. Define the type of events to which this notification request applies. For example, some notifications may apply only to “live” events, or just to scenarios, or just to test events.
LIMIT_VALUE	Float	The value of this product in this cell.
MESSAGE_FORMAT	String	Foreign key to the MESSAGE_FORMAT table. The message format defines the layout of the message, such as which data items are to be included.
METRIC	Integer	Foreign key to METRIC table. Defines the metric used in the notification computation.
NOTIFICATION_PRIORITY	Integer	Defines how this message is to be prioritized relative to other messages also to be sent to this user. (This value is not currently used.)
NOTIFICATION_TYPE	Integer	Foreign key to NOTIFICATION_TYPE table
SHAKECAST_USER	String	Foreign key to the USER table, defining which user is to be notified
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record
USER_MESSAGE	String	Arbitrary additional message the user wants sent when this notification request is executed. (This value is not currently used.)

5.2 FACILITY_NOTIFICATION_REQUEST

This table is used to join between the FACILITY table and the NOTIFICATION_REQUEST table. Rows in this table indicate which facilities are to be used in which notification requests. This allows a user to specify a single notification request that is in effect for all facilities. For example, a user might specify that they want an Email message delivered whenever all electrical substations are potentially damaged at the RED level. This can be accomplished by creating a NOTIFICATION_REQUEST and joining that request to all electrical substation facilities by placing a row in the FACILITY_NOTIFICATION_REQUEST table with that NOTIFICATION_REQUEST_ID and the FACILITY_ID of each substation.

Column	Datatype	Description
FACILITY_ID	Integer	Foreign key to the FACILITY table.
NOTIFICATION_REQUEST_ID	Integer	Foreign key to NOTIFICATION_REQUEST table

5.3 NOTIFICATION

The NOTIFICATION table contains all current and historical notification requests that have been actually triggered by the ShakeCast system. The Notification table may be thought of as the queue of notification activity (for pending notifications) or the log of activity (for historical notifications).

Column	Datatype	Description
NOTIFICATION_ID	Integer	Locally generated primary key
DELIVERY_ADDRESS	String	Actual user address (e.g., email address, pager ID, phone number, etc.) used for the message.
DELIVERY_TIMESTAMP	Datetime	The last time delivery was accomplished for this notification entry.
TRIES	Integer	The number of times delivery has been attempted for this entry
DELIVERY_METHOD	Integer	Foreign key to DELIVERY_METHOD table. Contains the method by which this notification is to be delivered. (Note that this value has been denormalized for performance, and could be derived from the NOTIFICATION_REQUEST table.)
DELIVERY_STATUS	Integer	Foreign key to DELIVERY_STATUS table. Contains last delivery status if delivery was attempted (may be success or errors), or completion or cancellation value.
DELIVERY_TIMESTAMP	Datetime	Time the queue entry was successfully delivered
EVENT_ID	Integer	Foreign key to the EVENT table. Contains the event for which this notification was triggered.
EVENT_VERSION	Integer	

GRID _ID	Integer	Foreign key to the GRID table. Contains the GRID for which this notification was triggered.
GRID_VALUE	Real	For Grid-type notifications, contains the specific value from the grid for this metric at the location of the facility, to be used in the notification message that is sent to the user.
METRIC	Integer	Foreign key to the METRIC table. Contains the metric for which this notification has been created. (Note that this value has been denormalized for performance, and could be derived from the NOTIFICATION_REQUEST table.)
NEXT_DELIVERY_TIMESTAMP	Datetime	Time the queue entry is next due to be processed
NOTIFICATION_REQUEST_ID	Integer	Foreign key to NOTIFICATION_REQUEST table
PRODUCT_ID	Integer	Foreign key to the PRODUCT table
PRODUCT_VERSION	Integer	
QUEUE_TIMESTAMP	Datetime	Time the queue entry was created.
SHAKECAST_USER	String	Foreign key to the USER table, defining which user is to be notified. (Note that this value has been denormalized for performance, and could be derived from the NOTIFICATION_REQUEST table.)

5.4 DELIVERY_STATUS

The DELIVERY_STATUS table defines the valid codes for the DELIVERY_STATUS column of the NOTIFICATION table. Valid types are locally defined.

Column	Datatype	Description
DELIVERY_STATUS	Integer	Status code
NAME	String	Status code name
DESCRIPTION	String	Description of the meaning of this status value.
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

5.5 DAMAGE_LEVEL

The DAMAGE_LEVEL table defines the valid levels of facility damage. Valid types are typically “Green”, “Yellow”, and “Red”. Valid types are locally defined

Column	Datatype	Description
DAMAGE_LEVEL_ID	Integer	Type code for damage level, locally defined
NAME	String	Damage level name
SHORT_NAME	String	Abbreviation for damage level name
DESCRIPTION	String	Further descriptive information about the meaning of this damage level code
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

5.6 MESSAGE_FORMAT

The MESSGAE_FORMAT table defines the actual text and substitution directives for the message to be delivered. Messages may have varying lengths, be in various languages, or provide for substitution of various kinds of event and product data.

Column	Datatype	Description
MESSAGE_FORMAT	Integer	Type code for message format, locally defined
NAME	String	Notification type name
SHORT_NAME	String	Abbreviation for notification type name
DESCRIPTION	String	Description
FORMAT_STRING	String	A formatted string including substitution directives for event and product data.
TEMPLATE_FILE	String	Filename of a template to be used for constructing this message.
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

5.7 NOTIFICATION_TYPE

The NOTIFICATION_TYPE table defines the valid codes for the NOTIFICATION_TYPE column of the NOTIFICATION_REQUEST table. Valid types include “SHAKING”, “DAMAGE”, “NEW_PROD”, “NEW_EVENT”, “UPD_EVENT and “CAN_EVENT”.

Column	Datatype	Description
NOTIFICATION_TYPE	String	Type code for notification requests
NAME	String	Notification type name

DESCRIPTION	String	Additional descriptive information about this notification type
NOTIFICATION_ATTEMPTS	Integer	Default value for maximum number of tries for this type of notification
NOTIFICATION_CLASS	String	Grouping value for notification types.
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

5.8 NOTIFICATION_CLASS

The NOTIFICATION_CLASS table defines groups or classes of NOTIFICATION_TYPES. Typical classes include "EVENT", "SYSTEM", "FACILITY", and "PRODUCT".

Column	Datatype	Description
NOTIFICATION_CLASS	String	Type code for notification class
NAME	String	Notification class name
DESCRIPTION	String	Further descriptive information about the meaning of this notification class
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

5.9 SHAKECAST_USER

The USER table has a single record for each user who is to receive a notification.

Column	Datatype	Description
SHAKECAST_USER	String	Unique identifier for each user
FULL_NAME	String	Full name of the user
PASSWORD	String	Hashed password of this user in the ShakeCast server
USER_TYPE	Integer	Foreign key to the USER_TYPE table
EMAIL_ADDRESS	String	Primary email address for this user
PHONE_NUMBER	String	Primary phone number for this user
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

5.10 USER_TYPE

The USER_TYPE table defines the valid codes for the USER_TYPE column of the USER table. Valid types are locally defined.

Column	Datatype	Description
USER_TYPE	String	Type code for user types
NAME	String	User type name
DESCRIPTION	String	Further description of the user type
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

5.11 USER_DELIVERY_METHOD

This table records the delivery address and auxiliary information for the delivery methods for each user.

Column	Datatype	Description
USER_DELIVERY_METHOD	Integer	Locally-generated primary key for this USER_DELIVERY_METHOD.
SHAKECAST_USER	Integer	Foreign key to the SHAKECAST_USER table.
DELIVERY_METHOD	Integer	Foreign key to the DELIVERY_METHOD table.
DELIVERY_ADDRESS	String	The address used for this delivery method for this user.
PRIORITY	Integer	The Priority of this user delivery method over other records for the same user and delivery method. (This value is not currently used.)
AUXILIARY_DATA	String	Auxiliary data used by some delivery methods. (This value is not currently used.)
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

Chapter 6 ShakeCast Internal Operational Tables

A number of tables that define the operational configuration of the ShakeCast server. These tables are documented in the following paragraphs.

6.1 Processor Parameter

The Daemon Parameter Table stores parameters that control the behavior of the ShakeCast Processors such as the Exchange Processor, the Notification Processor, and the Message Processor.

Column	Datatype	Description
PROCESSOR_NAME	String	Name of the processor. Processors "know" their own name because it is passed to them as a parameter when they are invoked by the operating system.
PARAMETER_NAME	String	The name of the parameter for which the value is being defined.
PARAMETER_VALUE	String	The value to which the parameter is to be set. Both numeric and string values are stored as strings.
UPDATE_TIMESTAMP	Date	The date and time the record was last updated
UPDATE_USERNAME	String	The local database username of the user who last updated this record

6.2 Dispatch Task

The Dispatch Task Table is a persistent record of ShakeCast Daemon dispatch activities. This table is used to communicate task data between various processes in the ShakeCast System. The table is used in such a way that dispatch requests will survive server crashes or failures in the ShakeCast System software.

INDEX