Request v3:
A Modular, Extensible Task Tracking Tool

Joe Rhett, Navigist

Request v3: A Modular, Extensible Task Tracking Tool

Joe Rhett – Navigist

ABSTRACT

Tracking tasks remains one of the most difficult issues facing any working team of administrators. Even with the implementation of commercial tools available today, e-mail and hallway conversations remain the standard for task management in many organizations; however, these make it difficult and time consuming to remain current on issues, and do nothing to summarize the long-term history of tasks and completion thereof.

Many commercial tools are available to handle task management, and most work quite well for stereotype models of their intended environments – development teams, help desk, etc. Unfortunately, these systems often have limitations which prevent their use (or a simple deployment) in a pre-existing, working environment. Other systems are difficult or time-consuming to use, and remain ignored in favor of task accomplishment. Few freely available systems provide the statistics to analyze productivity, generate statistics, and otherwise please management.

Request v3 was designed to provide the necessary essentials for modern task management: a selection of user interfaces, support for multiple database backends, flexible security controls, and extensive reporting capabilities. It runs cleanly in heterogeneous environments, including those that have a large installed base of Windows users. It includes command line, e-mail, and web interfaces, in addition to an Extension Interface which provides a simple way to access the Request system from other programs, scripts, or any custom interface one may create. The authentication, notification, data storage, and logging functions are processed within separate modules, allowing a variety of backend databases to be supported.

Introduction

This paper focuses on the modifications and extension of the venerable task tracking tool Request, and how it may be effectively utilized to increase the response capability of any organization that does not have an effective, well-used task management tool.

The paper starts with an overview of the motivation behind the project and some history of Request itself. The following sections present the changes made and the functionality added to the system. The final section presents potential and operational uses for Request v3.

Motivation

Although our team of administrators were quite effective at using e-mail for task management, we believed that not enough information was available to our users about task status, and changing responsibility for the task was never handled well. Although our staff was competent, we could not step in to handle emergencies in another's field of responsibility. No information was available to analyze our performance, or set expectations for our users.

The initial goal was to find a simple system we could quickly implement to track tasks. Given the extensive automation in our environment and the highly mobile nature of our user base, it was clear that any solution for task management must support the existing environment, work with the existing systems, and make it easier for administrators to perform their jobs, not add to their existing burden. The solution had to provide additional functionality without requiring changes to our software or current administration style. The goal became one to find a system which would provide this baseline functionality for task management in an operations environment.

After analyzing the existing commercial solutions, it was clear that none of the solutions directly supported our existing environment of mixed version Intel, Sparc, and HP unices with a growing amount of Windows NT systems. None of them would support all of the methods of input – command line, web-based, e-mail, and non-interactive – necessary. Most of the solutions required extensive investments in time and money (Remedy,Clarify) to be fully operational, yet would not integrate seamlessly into our environment. There simply was not a practical solution we could quickly implement.

At this point, we began to examine the freely available products. As with the commercial products, these packages worked seamlessly for one or two environments, but did not handle everything we were required to support. Many of the latest ones did not include command line or non-interactive input (Jitterbug, PTS, troublemh), and some of the web interfaces were obvious hacks (GNATS). None of them included clean support for MIME e-mail. None of them provided even a basic, functional set of statistics that our management desired.

The available products fell into two categories: simple, useful systems for tracking progress, and complex systems that provided statistics. None of the systems had both in good measure, and all of them required further change to make a viable system capable of supporting our diverse systems and Windows-based user community. We needed something we could quickly add the missing functionality to, and start using immediately. In the end, this was perhaps the strongest criteria.

We chose Request 2.1c as our starting point due to the fact that I was able to install and use it within an hour, and add the essential missing functionality within an afternoon.

History

Request was originally written by Shawn Instenes and James Sharp of Lawrence Livermore National Laboratories. It started as a single, large script written to manage the tasks of a specific set of administrators assisting Unix workstation users. This KISS[1]-style system works well in an environment of trusted users, each having accounts on the system where the database was stored.

Request 2.1 provided the following features:

- Command line and interactive interfaces
- The ability to open, close, assign, and update tasks
- The ability to set assigned and default due dates for tasks
- A defined list of who the available assignees where
- The ability to list open and closed tasks within the last week or month

Request 2.1c (UMI): Stuart Levy (University of Minnesota) added support for e-mail and a proto-standard query interface, which allowed reports to be generated using keywords and values. Unfortunately, these routines contained their own implementations of each function and had little data verification.

Request 2.5 (LLNL): Mike Miller had updated the original source tree, and began to generalize the routines and remove some redundancy. He also added printer support and defaults to the prompts.

Request 2.6-alpha (LLNL): Mike Miller added NFS-friendly locking code, and time spent and priority fields to the record format. The time spent and assigned fields became arrays, providing a history mechanism.

[1]KISS: Keep It Simple, Stupid – a philosophy of programming which accents functionality rather than the addition of extra features.

When I first started working with the code, my immediate goal was to provide functional e-mail and web interfaces. Stuart Levy's e-mail interface was functional, but performed all of its own data processing, which sometimes made it incompatible with the command line interface. In particular, dates were input in different ways, making the limited reporting functionality almost useless.

Due to the lack of standardized methods in the code, I was not able to easily add a web interface. To fulfill the immediate need, I created a quick web interface by checking the data input and using the command line features of Request. The data was provided back in fixed-width, preformatted output. This was not the best approach, but fulfilled the immediate need.

Even after all of these changes, Request remained lacking in the following ways:

- No security (all files mode 777)
- A web interface was not provided
- MIME e-mail was not supported
- Code was inconsistent, sometimes non-functional
- Comprehensive statistics were not available
- The ability to notify or alert about status was not available

Design Goals

Review of the system revealed many problems within the code, mostly related to improper assignments, and logging changes before testing whether to perform said changes. The code base was inconsistent and redundant, where each function reimplemented every operation, often in incompatible or inconsistent ways. Adding new features to the system required modifications to every function. The "simple task" of integration became a complete re-write of the system.

Four major issues define changes to the code base: ease of use, remote access, security, and interoperability. The system is designed to:

1. Secure the dataset against unauthorized operations or direct attacks.
2. Support local and remote users on Unix, Windows, and Mac platforms.
3. Provide an intuitive system which does not require instruction to use.
4. Process non-interactive input and output from existing automated systems (for example, accept input from HP OpenView, and make use of our alpha-paging software).

To prevent similar problems in the future, the new design centralizes all data access, verification, and logging. In addition to resolving the inconsistency and redundancy issues, these changes make it simple to add a modular Extension Interface, allowing new methods of access to the task information with a minor amount of code.

Major Feature Changes

1. Year 2000 Compliant – code automatically fixes millenia problems when loading/saving tasks
2. MIME e-mail capability – can parse and add plaintext sections of MIME messages.
3. E-mail, web, and interactive interface use the same common routines.
4. Standardized logging mechanism.
5. Common notification methods.
6. Configuration option for European date format.
7. Pointers to data structures are used, rather than copying arrays between each routine.
8. Clean code interface allows simple extensions without breaking upgrade capability.
9. New fields store information regarding time spent and task priority, and retain a history of assignments for each user.
10. Debugging can be enabled at any time within the program (limited by authorization).

Common Routines

As mentioned before, the biggest problem in the old Request code was the lack of centralized data control. An environment which forces every function to reimplement data validation and storage will develop inconsistencies in the different implementations. Request 2.1c is perhaps the worst example of this, where dates were stored in a different format (Feb25 vs 2/25) by the e-mail interface, making the date-oriented reporting absolutely useless.

Request v3 uses a common set of routines for all data access, providing consistent data validation and verification, thus relieving the burden from the interface. A reference to the data is returned from the method, allowing the interfaces to input and output in any fashion desired. It becomes trivial to design interfaces for non-interactive scripts.

The centralized system makes it easy to add external functionality. For example, the Notify module can interact with other management applications, such as management software, alpha-pagers, and pop-up windows. The documented API used by the modules allows these additions without modification to the source code, thus allowing live testing and integration.

Interfaces

Command Line

The command line interface is backwards compatible with the previous versions of Request, but adds additional functionality not present in the previous versions:

1. Configurable default action (default was to create a new request, is now often interactive)
2. Chaining of commands (“request -u 1234 -c -Q 1234”)
3. Word forms for all the actions (“update 1234”)
4. Extended search criteria (“find state open,assigned jrhett,contains Print”)[2]
5. The ability to enable debug output within the program
6. The ability to create new requests while in interactive mode
7. Command history mechanism ala c-shell
8. Optional direct access to routines for testing purposes (limited by authorization)[2]

E-Mail

The e-mail interface, originally written by Stuart Levy, performed all data operations itself and was therefore sometimes incompatible with the main routines. The e-mail routines were rewritten to utilize common data access routines, and handle messages in MIME format.

Experience and user input lead to the implementation of these features:

- External files for custom response messages
- Immediate feedback to the user[3]
- Automatic processing of looped or failed messages

[2]These features were thought of and partially implemented by Stuart Levy. We simply extended the ideas a bit further, or rewrote it completely to provide the intended functionality.

[3]Actually implemented, then removed by Stuart Levy when he decided it was too annoying; obviously optional.

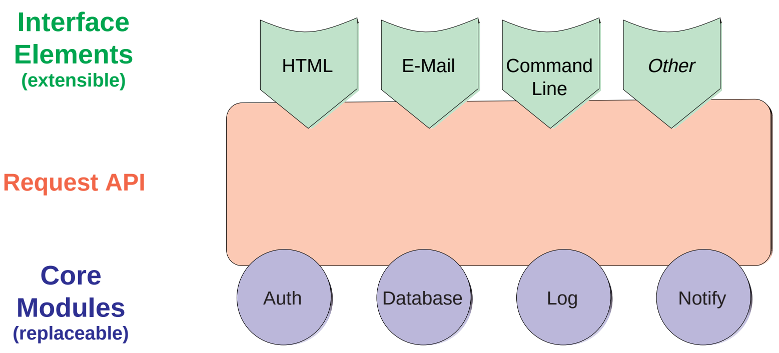

Figure 1: Request structure.

One site of which I am aware has added encryption and digital signature authentication to the e-mail interface.

Web Interface

Supporting a wide variety of users requires the ability to provide a variety of interfaces, accessible in a method understood by each user community. Users do not always take the time to learn an interface, especially when they already feel inconvenienced by some issue; therefore, we found it important to provide an interface with which most users feel comfortable. Over the last few years it has become apparent that the web browser is the most common, usable interface.

The web interface uses the CGI environment to receive data from HTML forms and return output in HTML style. The default output is an HTML table not dissimilar from the command line output, but can be easily modified to output in any format desired. It seems likely that every site will use a different style, so customization is left to the site administrators.

This is actually an example implementation of the Extension Interface.

Extension Interface

The Extension Interface provides the ability to support new interfaces and custom access methods without modifications to the original code base. Originally designed to interact with HP OpenView NNM and PPT's E-Page software, the interface easily adapts to most needs. Perl/TK and other interfaces would be simple implementations for enhanced user input mechanisms.

The interface is well documented. Skeleton and example invocations are provided to assist new development.

Authentication, Storage, Notification and Logging

All of the following components are PERL classes (Request::Authentication, Request::Storage, etc) implementing a documented interface. Any or all of these could be replaced by custom modules utilizing different resources, such as an existing Oracle database or an LDAP server. An example module of each type is supplied to assist in testing.

Authentication Module

Request v2.1 performed no authentication, allowing any user to execute any command. While this obviously is not sufficient for complex environments, anything which enforces stricter security may complicate administration unnecessarily. Therefore, version 3's authentication module provides flexible security controls, allowing everything from relaxed, open administration to strict, multilevel authentication which validates each operation against the user's rights.

The authentication information can be accessed from any source. The default module uses the local system's user information, but modules which support DBM databases, Radius, LDAP, or any other system could be easily created.

Database Module

By moving all data storage and retrieval operations into a module, data can now be stored in one of any number of different backend databases. This provides the ability to manage data using a site's current environment. Standardized tools can be used to manage, distribute, and generate reports from the data, instead of Request-specific code.

The standard DBM database provides the default database support, to retain backwards compatibility with older versions.

Logging Module

Unlike Request v2, extensive support for basic, extended and debug logging is available. Similar to the other modules, the logging routines can be replaced at will.

The default logging module provides direct file logging with a priority control. An additional module supplies standard syslog() logging methods.

Notification Module

Unlike Request v2, extensive support for basic, extended and debug logging is available. Similar to the other modules, the logging routines can be replaced at will.

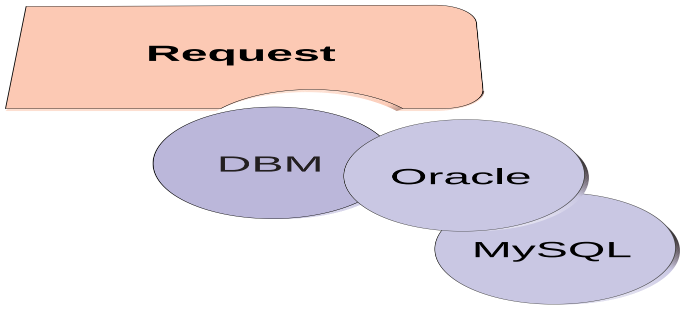

Figure 2: Request module.

The default logging module provides direct file logging with a priority control. An additional module supplies standard syslog() logging methods.

Reporting and Statistics

One of the significant differences between the freely available systems and the commercial systems is that the commercial systems have extensive capability to produce reports and statistics of any flavor. The freely available systems usually provide somewhat limited functionality, often being concerned with ease of use and functionality. Managers often need reports and statistics to justify budgets and review resource allocations. The implementation of commercial systems is often based upon that criteria. Far too often administrators are saddled with unfriendly, cumbersome task management tools, simply because their managers need more reporting capability.

Managers desire methods to review activity, the project history, and statistics on task completion and overall performance. And sometimes the pointy-haired ones just like pretty pictures. In the trenches, administrators need to justify their time and resource allocations to their management, and identify problem areas which strain the group as a whole.

Request v2.x provided only limited support for reports, mostly oriented towards those which were based upon the open and closure of tasks. Request 2.1c provided a basic query language for generating reports based on a variety of criteria. Testing found that not all of the functionality was implemented, and many of the reports were not useful due to a lack of data validation in the input routines.

The Request v3 reporting engine extends the query language defined by Stuart Levy, providing access query functionality with a standard format. Stronger, common validation provides more accurate reporting. Reports can be generated against any field, including custom fields defined in the local configuration. Thus, changes can be made to the data structure without modification to the reporting engine.

An optional extension of the reporting engine will create GIF graphs from any single or dual-column numeric report.

Security

The previous versions of Request provided little security. All of the shared data areas were world-writable, and changing or removing information from the database was trivial. Request v3 removes all need for globally accessible directories, implementing access using a Unix group membership.

This is considered somewhat basic. The best security comes from using the modular structure to store data in a backend database (Oracle, MySQL, etc), which implements a much stronger security mechanism.

Internally, Request v3 includes a per-operation authorization mechanism. By default it remains as open as the previous versions, but may be configured as tight as the local practice requires. This is discussed in the Authentication Module section.

Implementation

Installing the Package

If you have fought your way through installation of many older task management tools, you will be pleased with the installation process in Request v3. Unlike the previous versions of Request, installation is simply make install. The installation will ask a few questions, and installs a working system. After installation, the configuration file may be edited to provide additional customization, but this is often unnecessary for a basic installation.

The installation program attempts to find an existing Request installation, and prompts for action if found. If you choose to upgrade, the existing database will be available from the new version. If you choose to duplicate the existing configuration, both systems may be used during a test period. In either case, the installation program will automatically configure Request v3 to match your old configuration, allowing immediate use of the new system.

```
# A minimal example which changes the assigned admin
use Request;

# Get the command line input
$person = $ARGV[0];
($day,$month,$year) = (localtime(time))[3,4,5];
$tomorrow = ($month + 1) . "/" . ($day + 1) . "/" . ($year + 1900);

# Open the request and reassign it to the person
my $request = $Request->retrieve($id);
$request->assign($person);
$request->changedue($tomorrow);
```

Listing 1: Adding a new interface.

Customizing the Features

All of the standard configuration parameters are contained within a single text file. Unlike previous versions of Request, this file is not a PERL script, so values do not need to be quoted or escaped!

```
DefaultAction  -i
DefaultDue     2days
MailCommand    /usr/lib/sendmail -t
```

It would be very simple to implement an integrated configuration editor, but it has not been necessary.

Adding New Interfaces

A new or improved interface may be installed and tested without affecting normal operation. The interface needs to do nothing more than use the PERL class, create an object (if necessary) and call the methods; see Listing 1.

To assist in development, we have supplied a dummy interface which utilizes every available method of the Extension interface, but contains only comments for input/output code.

Adding New Modules

Any of the Authentication, Storage, Notification, or Logging modules may be replaced at any time. These modules are stored in the /lib path of the installation directory. A replacement module may be tested by changing an environment variable, allowing testing without interrupting normal operation; see Listing 2.

To assist in development, we have supplied a complete example for each module which implements each function of the API, but does not actually do anything. These examples provide skeletons from which to begin development.

Availability

Request v3.0 is publicly available using anonymous HTTP (web browser) at http://www.navigist.com/Reference/Projects/Request.

A mailing list has been created for questions, comments, patches, and recommendations for Request. You can subscribe by sending electronic mail to majordomo@lists.isite.net with the text "subscribe request-users" in the body.

Author Information

Joe Rhett started his career as an independent contractor, implementing Unix systems and LANs in the Washington, D.C. area. After working with the NAVSEA MAN in Crystal City, VA; he moved to California, and began work with Navigist, a small team of consultants in Silicon Valley. He specializes in network engineering, security, and performance; often focusing on application implementation. He tries to limit his programming to small applications and utilities which improve administration capability, problem isolation, and overall response time. He can be reached via e-mail at <jrhett@navigist.com>, or through any of the contact methods listed at http://www.navigist.com/Staff/JRhett.

References

James M. Sharp, "Request: A Tool for Training New Sys Admins and Managing Old Ones," *Proceedings of the Sixth USENIX Systems Administration Conference (LISA VI),* 1992.

Stuart Levy. "README for Request v2.1c," *Request 2.1c distribution*, July 1996.

Mike Miller. "README for Request v2.5," *Request 2.5 distribution*, April 1995.

N. Freed & N. Borenstein. "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies," *RFC 2045, Network Information Center*, 1996.

N. Freed & N. Borenstein. "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies," *RFC 1521, Network Information Center*, 1993.

```
# A minimal example
package Request::Log;

# logfile opened in init() and closed in final()
sub log {
   my ($pkg,$routine,$user,$level,$message) = @_;
   print LOGFILE "${level}/${pkg}:${routine}/${user}: ${message}\n"
}

# This assumes you only use debug in command line mode
sub debug {
   my ($pkg,$routine,$user,$level,$message) = @_;
   print STDERR "${level}/${pkg}:${routine}/${user}: ${message}\n"
}
```

Listing 2: Adding a new module.

N. Freed & N. Borenstein. “MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies,” *RFC 1341, Network Information Center*, 1992.

D. Crocker. “Standard for the format of ARPA Internet text messages,” *RFC 822, Network Information Center*, 1982.

Sriram Srinivasan. *Advanced PERL Programming*, O’Reilly and Associates, 1997.

Larry Wall, Tom Christiansen, Randal Schwartz. *PERL Programming*, O’Reilly and Associates, 1996.