Design for Robustness

by Theresa Ray of Tensor Information Systems, Inc.

Sponsored by Apple Computer, Inc.
Apple Developer Connection

Design for Robustness

by Theresa Ray of Tensor Information Systems, Inc.

A quick reference guide to the major topics involved in designing a bulletproof WebObjects application including application coding techniques, application configuration options, multi-platform development and testing.

Developing a robust application for either a web or client-server interface is critical for your business, particularly with web applications, which are generally developed for use by people outside your company. A web site makes an impression on its users, in many instances defining the way they feel about your company. Therefore, developing a robust application is critical to your business.

Application Coding for Robustness

There are several coding techniques that you can use that will enhance the strength of your application. These include placing commonly used code in a framework, exception handling techniques, memory management optimization, and management of database connections. Each of these topics is explained below.

Frameworks

As you develop your WebObjects applications, you will probably create several methods or components that are of a general-purpose nature and are not specific to the application you are developing. There are three ways you can treat this code. The first way is to put the code "in-line" in the method that requires the function. While this technique works, it makes it very difficult for other developers to recognize that you have a solution that is, in fact, generic. It also makes it impossible to reuse the code without cutting and pasting it everywhere that you need it, even within the same application. This obviously does NOT follow any object-oriented design methodology and the practice should be avoided.

The second way is to create a method within one of your application's classes for the code. This technique allows you to reuse the code from within your application without cutting and pasting it, and on the surface adheres to object-oriented design methodology. However, this technique still restricts availability outside the current application and therefore does not follow the true spirit of object-oriented design. If you want to use the same component or method in another application, you will have to cut and paste it into your other application. Again, this makes it very difficult for other developers to recognize that you have a generic solution. In addition, every time you cut and paste into a new application, you have the potential to introduce errors that will have to be tracked down and fixed.

The third way – the most robust solution – is to add the method or component to a framework. A common framework can work as a code repository, providing a library of sorts for generic code that can be used by any number of applications. The code is written and tested once, saving time during the development and debugging of a new application. While this technique sounds obvious, some developers forget to identify generic components that they create when developing a new application. Periodic analysis of an application's code during development is a good way to discover these generic components.

When creating a framework for common or generic code, you may want to consider developing a SET of frameworks instead of a single framework. Just as Apple has divided its frameworks into distinct sets – Foundation, EOAccess, EOControl, WebObjects and so forth – you should divide your frameworks into logically distinct sets. If you have a neat subclass for NSString and a generic WebObjects login component, you might want to put these into two different frameworks. This is particularly important if you ever develop non-WebObjects applications with OPENSTEP because both the non-WebObjects and WebObjects applications can use the NSString subclass.

For information on how to create reusable components and frameworks, refer to the WebObjects Developer's Guide and the Survival Guide: on Frameworks.

Exception Handling

When you release your application, there may be times when your application will raise an exception. There are many reasons why exceptions occur, ranging from errors in the code to a database server that is currently unavailable or a network failure. The standard exception page presented by WebObjects is meant to aid developers in the creating and testing of their code – and is NOT meant to be a production-type user-friendly error page. You will want to replace the default error page with an error page customized for your application. You can even customize the error page based on the type of exception generated if you so desire.

But before you can return a distinct error page, you need to catch the exception first. There are several ways to catch an exception in a WebObjects application. Implementation of the first four bullet points is usually sufficient to produce a robust application.

- The WOApplication class provides a method called handleSessionRestorationErrorInContext that is usually called because the user's session has timed out. This method, by default, returns a page with debugging information. Your subclass of WOApplication can override this method and return a friendlier error that says something like "We're sorry, but after a period of inactivity your session has timed out. Please click here to begin a new session".

- The WOApplication class provides a method called handleSessionCreationErrorInContext that, by default, returns a page with debugging information. Your subclass of WOApplication can override this method and return a friendlier error.

- The WOApplication class provides a method called handlePageRestorationErrorInContext that is usually called because the user has backtracked too far. This method, by default, returns a page with debugging information. Your subclass of WOApplication can override this method and return a friendlier error.

- The WOApplication class provides a method called handleException that is called whenever an error occurs within the request-response loop (in Objective-C, this method is handleException:inContext:). This method, by default, returns a page with debugging information. Your subclass of WOApplication can override this method and return a friendlier error.

- You can wrap exception-handling mechanisms around key portions of your code to specifically handle that exception. For example, when saving changes to the database, you may implement Java code such as the following to directly handle the failure to save the record successfully:

```
try {
  this.session().defaultEditingContext().saveChanges();
}
```

```
catch (Throwable exception) {
  // Implement whatever code you want for a failure to
save
  // successfully log, return a different page, etc
}
```

What you do when you catch an exception depends on the type of exception that was raised. For a session restoration error, you probably want to display a user-friendly message with a link to a destination that allows the user to re-enter the site. For a generic error of unknown type (trapped by the handleException method) you might want to log the error and email the site administrator. Or as in the example above, you can wrap exception-handling code around key areas of your application and handle the errors on a case-by-case basis.

Memory Management and Application Optimization

Robust applications aren't just applications those that handle errors properly or that make optimal use of reusable components. Certain application design techniques may allow a single machine to serve more instances of an application or to serve each instance of an application more efficiently. First, consider the variables you have defined at the application level. Are they really generic for the entire application? Do they provide any added value to the application or are they just flashy? Application variables persist throughout the entire lifetime of the application, and therefore have a direct impact on the memory footprint of your application.

Second, consider your caching strategies. By default, a WebObjects application caches up to 30 response pages for a session. This parameter can be disabled or deepened, depending on the needs of your application. Caching enhances application performance because when a user backtracks to page, it does not need to be recreated from scratch. A cache size of 30 pages is usually sufficient for most applications. Additionally, by default, a WebObjects application allows the client browser to cache response pages so that when a user backtracks, the application need not be contacted at all. If you wish to disable this (using the setPageRefreshOnBacktrackEnabled method), the retrieved page will only be asked to generate its response. Value extraction from the request and action invocation is not performed. Use of the permanent page cache can also optimize your application. The WOApplication class has a method called savePageInPermanentCache that will place the target page into a separate permanent cache (with its own cache size), effectively caching the page for the duration of the application regardless of your page cache size. This is useful for components that should always stay cached such as the navigation frame of a multi-framed application.

Third, write your application in a thread-safe manner, locking any access to shared resources as necessary, so that you can enable the concurrent request handling capabilities of WebObjects 4.0 and higher by overriding the allowsConcurrentRequestHandling method to return true.

Fourth, consider your logging strategies for deployment. Logging a certain amount of information from your application is wise, as it aids in disaster-recovery and debug efforts. But you should log wisely. Make sure that the information you log is useful. There is usually no need to log all the SQL you generate for the duration of the entire application. Turning your debug log statements off (using WODebuggingEnabled) before deployment is usually desirable. Similarly, turning off all trace options is usually desirable. Leaving all the SQL logging, debug logging and trace logging enabled will quickly consume disk space on your production machine.

Fifth, consider periodically shutting down and restarting your applications. Some applications leak a certain amount of memory, and periodically terminating and restarting the application allows the memory to be freed. Use Monitor to gracefully shut down and restart the application by having the application refuse new sessions. When the number of active sessions reaches zero, you can safely terminate the application without interfering with any user's active session.

Sixth, optimize your sessions. If there is a minimal amount of session-level information that you need to track for your user, you may be configure your application by overriding the WOSession method setDistributionEnabled so that the user can return to any running instance of your application as he or she navigates through your site. See "WebObjects Viewed Through Its Classes" for more information about running a stateless application, or see "Managing State" for information on alternate stage storage strategies. If your application will be running hundreds of concurrent sessions, limiting the session level information stored reduces the memory footprint of your application, allowing one machine to serve more users.

Seventh, because storage of sessions in application memory can consume large amounts of memory over time, terminate any old sessions in order to free up this memory. This can be accomplished either by overriding the WOSession method setTimeOut or by calling the WOSession method terminate directly when you know that a user's session is finished. Overriding the setTimeOut method allows you to specify the number of seconds that a session is allowed to be idle before being terminated. Occasionally, you may develop an application that "finishes" with a user at a well-defined point. For example, if a user came to your application to register for a seminar, after the registration was complete you may no longer need that user's session. If this is true, you may message the WOSession method terminate directly to immediately end the user's session and free up the memory associated with that session. If the user backtracks and resubmits any forms, a session restoration exception will be thrown.

Eighth, optimize your components by using standard code optimization techniques. Don't fetch more information than you need from the database. Consider prefetching small look-up tables from the database at the application level so that each session or component doesn't have to make the round-trip to the database for relatively static, frequently-used information. When fetching information from the database, be sure to specify a fetch limit in your fetch specification or WODisplayGroup. This limits the number of rows that can be fetched from the database at one time, preventing a user from inadvertently fetching an entire table from the database.

Robust Database Connectivity

When a single machine performs both as the application server and the database server, the connection between application and database rarely drops. When two or more machines are involved, a network problem is usually the culprit for a dropped database connection. Nevertheless, an application should be able to gracefully handle a missing database connection. Overriding the WOApplication method handleException will gracefully trap all database-related errors. But what if you want to try to restore the connection automatically?

WebObjects 4.5 provides just such a capability! See the WebObjects 4.5 documentation for details. For versions of WebObjects prior to 4.5, it is difficult to robustly implement a solution that automatically reconnects to the database, but the basic steps are as follows:

1) Iterate over a list of registered database channels (note: may be more than one!). Call both unregisterChannel: and disconnect.
2) Disconnect and remove obsolete contexts.
3) Have a timer call some fetch mechanism periodically until you time out (fail) or successfully fetch from the database.

If the attempt fails, you could display a page that informs the user that the database is currently down, and that he or she should try again later. If the user has submitted a significant amount of data that should be saved to the database, and you don't want the user to have to enter all of the information again in a later session, you could display a page telling them that the database is currently unavailable, and that the application will continue to try for

several minutes to reconnect. Your application could then try to reconnect to the database a set number of times – raising the exception if the number of times is exceeded without a successful connection, and continuing with the save operation if a connection is reestablished.

Application Configuration

There are many application configuration options that can affect the robustness of your application, including the number of threads, queue depth, the number of instances to start, and web server configuration. A few details are provided here, but you should read the on-line document "Serving WebObjects" for a thorough explanation of each of these topics.

Threads – The default adaptor does multithreaded adaptor I/O and resource handling, but some applications (particularly those converted from prior WebObjects versions) should be run in a single-threaded manner in order to be robust. To run adaptor in single-threaded manner, set the WorkerThreadCount to 0.

Listen Queue Depth – The listen queue depth indicates the number of transactions that can be in the socket buffer (the listen queue) awaiting processing. If the number of transactions in the buffer reach the limit set by the listen queue depth, the socket refuses new requests. The default depth is five. When an application's request load varies by period (that is, it experiences "spikes"), you can increase the listen queue depth to improve performance either by changing the listen queue depth using the Monitor application, or by messaging the WOApplication method setListenQueueSize.

Auto Recover – Selecting the Auto Recover option in the Monitor application allows your applications to be automatically restarted if they should crash. The Monitor application keeps track of how many application "deaths" have occurred and when each "death" occurred, so that you can search your application log files more efficiently for evidence of what might of caused the "death". By allowing Monitor to automatically restart your applications, you keep more instances running simultaneously with less human intervention, increasing your site's performance and responsiveness.

Determining the Number of Application Instances to Start – There is not easy answer to the question, "How many instances of my application should I start?" The answer depends on the average load and peak load that your application will experience in conjunction with the average and maximum response times of your application. Use the performance information gathered by the Monitor application, as well as the statistics gathered by the application to determine the correct number for your application. See the "Monitoring Application Activity" section in the online document "Serving WebObjects" for more information about performance and statistics information.

A note regarding web server configuration: while it is beyond the scope of this survival guide to discuss the configuration options for your web server, you should note that optimization of the application itself does NOT guarantee a robust web site. You cannot ignore the configuration of the web server used to serve the application. Read your web server documentation carefully for options you may want to take advantage of to ensure a robust deployment configuration. For example, most web servers allow you to specify how many web server instances to start (both minimum and maximum numbers are usually specified). This option allows the web server to start more instances of itself during peak loading, and to shut down unnecessary instances during quieter periods.

Multi-Platform Development and Deployment

In many cases, it is useful to develop on one platform, and deploy on a different platform. For example, many times it is convenient to develop on NT and deploy on Solaris or HP-UX. In this kind of configuration, there are useful custom environment variables that optimize the transition from one platform to another. And the fewer changes you need to make in transitioning your project between platforms, the fewer opportunities you create for errors to

creep into your application. Also, if your deployment server ever crashes, you have a design that can easily be placed on a different machine until your deployment server is repaired.

LOCAL_LIBRARY – This environment variable can be used to specify the directory containing the installation of any custom frameworks or palettes you may have developed. This location will vary from project to project. Note that the LOCAL_LIBRARY notion of "install" has nothing to do with the "install" target defined by the OpenStep Makefiles. The LOCAL_LIBRARY location is intended to provide an interim, common location for your built development projects so they can live somewhere other than ".". For example on Solaris, the LOCAL_LIBRARY variable might be set to /LocalLibrary/${LOGNAME}/solaris while on NT the same variable might be set to D:/LocalLibrary/winnt.

DEVROOT – This environment variable can be used to identify the root development directory. This directory is usually the common location for code, and will probably contain any global Makefile components. For example on Solaris, the DEVROOT variable might be set to /developer/Projects/OpenStep/DevRoot while on NT the same variable might be set to D:/Projects/OpenStep/DevRoot

BLDROOT – This environment variable can be used to set directory where you want your build output to go. If you don't set it explicitly, the OpenStep Makefiles will use the default value of ".". This can be non-optimal for two reasons. The first reason is that the build processes for different OpenStep platforms do not generally work and play well together (i.e. it pays to keep them separated). The second reason is that its noticeably slower to build/link across NFS mounts than it is to build/link on the local machine. The value of the BLDROOT environment variable should be set to some directory on your local machine for NT builds, and to some local directory on the PDO platform of choice for PDO builds. This allows you to keep the builds for the different platforms separated, and keep the object and executable files on the local machine so the compile/link process is faster. Note that BLDROOT has nothing to do with the "install" location.

In order to take advantage of these environment variables, the following changes should be made to the project Makefiles.

Add the following lines to the Makefile.preamble (only add the second line if you have a global preamble created and specify the appropriate global preamble filename).

```
INSTALLDIR=$(LOCAL_LIBRARY)/Frameworks
-include $(DEVROOT)/Global.make.preamble
```

A sample global preamble would include:

```
GLOBAL_CFLAGS = -g
GLOBAL_INITIAL_TARGETS = update_build_info
BUILD_OUTPUT_DIR = $(BLDROOT)/$(NAME)

########################################################################
########
# Install directory locations.
########################################################################
########

ifeq "Framework" "$(PROJECT_TYPE)"
LOCAL_WINDOWS_INSTALLDIR=$(LOCAL_LIBRARY)/Installed/$(PLATFORM_OS)/Fram
eworks
endif
```

```
ifeq "Tool" "$(PROJECT_TYPE)"
LOCAL_WINDOWS_INSTALLDIR=
  $(LOCAL_LIBRARY)/Installed/$(PLATFORM_OS)/Tools
endif

LOCAL_INSTALL_PERMISSIONS = 0755
```

You may want to create platform-dependent Makefiles for your project and include the appropriate Makefile automatically. For example, you could add the following line to your Makefile.preamble. If the platform operating system is HP-UX or Solaris, the MakefileEOF.preamble-$(PLATFORM_OS) would statically link in the necessary libraries and frameworks for the appropriate database. On NT, you just would not create a MakefileEOF.preamble-NT since no static linking is required.

```
-include MakefileEOF.preamble-$(PLATFORM_OS)
```

Using ProjectBuilder, make use of the LOCAL_LIBRARY environment variable to specify the search path for framework paths (you will need to make sure that your custom frameworks have been appropriately installed to this destination).

If you have a global postamble created (you might create one to update a BuildInfo.plist file that contains information regarding the most recent build), add the following to your Makefile.postamble, specifying the appropriate global postamble filename.

```
-include $(DEVROOT)/Global.make.postamble
```

When transferring a project from NT to UNIX, don't let any ^M characters sneak into the Makefiles. This can easily happen when editing on an NT machine across an NFS mounted file system. The ^M characters must be removed from the Makefiles before proper compilation on UNIX can be performed.

It is a good idea to organize the directory structure around the UNIX "make" utility. Directories and Makefiles can then be set up in a way that allows "recursive" builds to take place. Basically, if you tell make to make a particular target it sees that this target gets built both in the directory you're currently in and in all descendant directories too. This allows you to visit particular branches of the tree, and just build those. For example, if you had a directory called "Frameworks" that contained three frameworks named "AOLFramework", "PGFramework" and "CBFFramework", typing "make" at the command prompt from within the "Frameworks" directory would make all three frameworks. Similarly, typing "make clean" at the command prompt from within the "Frameworks" directory would clean all three frameworks. An example of the Makefile contained within the "Frameworks" directory is shown below.

```
all:
   @(cd AOLFramework ; $(MAKE); cd ../PGFramework ; $(MAKE); cd
../CBFFramework ; $(MAKE))
clean:
   @(cd AOLFramework ; $(MAKE) clean ; cd ../PGFramework ; $(MAKE)
clean; cd ../CBFFramework ; $(MAKE) clean)
```

Testing

Of course one of the most obvious and important means to make your application robust is to TEST IT THOROUGHLY. Make sure that you budget adequate time and money to test your application before you deploy it. Beginning at the time that the application specification is developed, create and maintain a proper test plan so that you can ensure you have tested all the features of your application. The person programming the web site should

NOT be the person to create the test plan, as he or she is too familiar with how the application works to provide an unbiased point of view. The developer should definitely review and add on to the test plan, but should not be the primary person to create it.

Similarly, the developer should be the first person to test the application, but others should test it as well. The people who can provide the best testing of your application are non-technical people who are completely unfamiliar with the project. They seem to more accurately reflect the type of user who will visit your site. In addition to the testing they do based on your test plan, they can help identify whether the navigation of the site makes sense, and will take paths through the site that no developer ever dreamed of using. Personally, I have found that these types of users will find significant errors in my applications that I had no clue were there.

Any information being saved to the database should be thoroughly checked for accuracy by more than one person (ideally not the developer or the tester). It is worth a comprehensive scrubbing before you deploy your application. There is nothing more annoying or costly than discovering after deployment that you have been saving erroneous information about your users. In the best circumstance, the erroneous data can be modified by the DBA. In the worst circumstance, you will not be able to trust the accuracy of any of the records submitted before the error was caught.

Make sure that all changes to a deployed site are thoroughly tested before release. There are roughly three types of changes you can make to a deployed site. The first change is a change to an image or to the HTML template. This change requires minimal testing before release. The second type of change is a change to the application code in a single class or method that does not affect any data being saved. This change requires moderate testing before release. The third type of change is a change to a common framework or component, or a change that affects the data being saved by the application. Due to the extent of this change, the entire test plan should be run again before releasing this change.

Make use of all testing tools available to you. The online documents "Serving WebObjects" and "What's New in WebObjects 4.0" discuss the use of the WOPlayback adaptor to performance test your application.

Conclusion

Development of a robust application is essential to the success of your web site. Proper use of the techniques presented in this guide can assist your efforts to develop an optimized, robust application and system configuration.

Resources…

http://developer.apple.com/techpubs/webobjects
WebObjects Developer's Guide
Enterprise Objects Framework Developer's Guide
Enterprise Objects Framework Tools and Techniques (online with developer release)
http://www.omnigroup.com/MailArchive/WebObjects
http://www.omnigroup.com/MailArchive/eof
http://www2.stepwise.com/cgi-bin/WebObjects/Stepwise/Sites
ftp://dev.apple.com/devworld/Interactive_Media_Resources
http://www.apple.com/developer
http://developer.apple.com/media
http://til.info.apple.com/

About the Author…

Theresa Ray is a Senior Software Consultant for Tensor Information Systems in Fort Worth, TX (http://www.tensor.com). She has programmed in OPENSTEP (both WebObjects and AppKit interfaces) on projects for a wide variety of clients including the U.S. Navy, the United States Postal Service, America Online, and Lockheed-Martin. Her experience spans all versions of WebObjects from 1.0 to 4.0, EOF 1.1 to 3.0, NEXTSTEP 3.1 to OPENSTEP 4.2, Rhapsody for Power Macintosh, and yellow-box for NT. In addition, she is an Apple-certified instructor for WebObjects courses.

Tensor Information Systems is an Apple partner providing systems integration and enterprise solutions to its customers. Tensor's employees are experienced in all Apple technologies including OPENSTEP, NEXTSTEP, Rhapsody, EOF and WebObjects. Tensor also provides Apple-certified training in WebObjects, Oracle consulting and training, as well as systems integration consulting on HP-UX. And Oracle.

You may reach Theresa by e-mail: *theresa@tensor.com* or by phone at (817) 335-7770.