Using Starkits for Easy Deployment of Server Applications

Mark Roseman
CourseForum Technologies
mark@markroseman.com

Abstract

Advances in consumer technology are leading to demand for network server applications that are being run by less technically inclined users, making ease of deployment critical. The Starkit framework makes Tcl deployment in general much easier, but can be enhanced for servers. Using the CourseForum and ProjectForum applications as examples, issues such as reducing dependencies, installation and configuration, starting and stopping, and various platform specific issues are explored. Experiences show that in this important application niche, Tcl, based on Starkits and the techniques described here, provides a strong deployment advantage.

1. A New Generation of Server Applications

Easy deployment? For *server* applications? Just the thought of it seems at odds with reality. We are used to thinking about network server operations being managed in dedicated centers, composed of extremely complex hardware, and run by extremely technically oriented information technology workers. Vast complexity, high degrees of configurability, and flexibility in the face of unique network setups are the norm for network server applications, aren't they?

Though the impenetrable data centers are still safe, we are seeing today a new type of environment where an ever-increasing range of new server applications are being run: in homes, personal offices, small workgroups. These changes have been spurred on by dramatic growth in consumer and SOHO (Small Office / Home Office) networking products like always-on broadband connections, wireless networks and devices, coupled with consumer operating systems (e.g. Windows XP, Mac OS X) that are powerful and robust enough to work as network servers, alongside regular desktop use.

We are seeing applications like Userland's Radio or Manilla products [17], which provide individual and small group weblog and content management support, or Amphetadesk, a personal news aggregator [2]. Both run as applications under Windows or Mac OS X, but work as web servers that are accessed via the user's web browser, rather than via a normal application user interface. We also see a range of calendars, personal web-mail, document management, file sharing, MP3 streaming, project management and other similar applications, which run as servers on users' desktop computers, but are made accessible via a network.

Unlike the traditional network administrators found in the bowels of corporate data centers, users of this new breed of network server applications are not generally trained in the technical intricacies of deploying and administering networks. Applications need to be installed and setup with a minimum of fuss, and they have limited resources and patience available to fix things if they go wrong. As with other types of software, free or trial versions of server applications are now commonly downloaded and experimented with, rather than being installed and run only after financial and resource commitments have already been made.

For these users, applications that are hard to install or configure are simply non-starters. For developers of such applications, hoping to reach this new audience, easy deployment of their software is critical.

This paper starts by examining the root causes of hard to deploy server applications, and details some of the resulting problems. It then turns to building such applications in Tcl, using the Starkit technology as a basis. While Starkits get us much of the way towards a deployment solution, for network server applications additional issues like reducing dependencies, installation and configuration, starting and stopping all need to be concerned. As well, there are many platform-specific issues, particularly on Windows and Mac OS X, that if addressed can greatly ease deployment.

Using the CourseForum and ProjectForum applications as examples, this paper provides a "cookbook" approach to assist others in developing easy to deploy network applications, building on the existing Starkit technology. Deployment has been a particular strength of Tcl when compared with many other development tools, and this work shows that advantage can be particularly significant in the domain of network server applications.

2. Deployment Today

We know that Tcl makes building network servers particularly easy. Here is the canonical network server application in Tcl, just a few lines of code that works on any platform.

```
socket -server Accept $portnum
proc Accept {sock args} {
    fconfigure $sock -blocking 0
    fileevent $sock readable "Readable $sock"
}
proc Readable {sock} {
    # … do something interesting
}
```

Similarly, tools like Rivet [3], which runs under Apache, or various CGI packages make writing Tcl scripted web pages as easy as:

```
<?
puts "Hello World"
?>
```

Despite the simplicity in coding, there are still a lot of steps involved in taking those simple fragments, and getting them to run smoothly on someone else's machine.

2.1. How Server Applications are Built

Most of today's network server applications, most particularly web applications, are neither short nor simple. They are also generally not written simply using the facilities of a base programming language, as in our canonical Tcl example.

Most such applications today are constructed to rely on a myriad of other technologies, including a mix of web servers, application servers, middleware, scripting languages, databases, programming libraries, external tools and applications, protocol handlers, and plugins. Each requires its own separate installation and configuration, usually via hand editing of multiple text-based configuration files. Static HTML files, images, scripts, libraries, configuration files and data files are scattered in many places throughout a complex directory structure.

This is not an unreasonable approach, particularly if you are developing applications to be installed only on your own custom dedicated computers housed at a central location, as was traditionally done. While the installation may take some amount of additional time, that time may be more than made up for by savings in development time that these various tools provide.

As the deployment of many network applications are now being decentralized and distributed though, this equation changes. Now you are asking each of your users, which depending on your application may potentially be a large number of people, to each pay the price of additional installation complexity, to cover your savings in development time. Multiplying out the time, and factoring in resources available to the users, suddenly this tradeoff may not be so appealing.

The problems are being exacerbated as we're seeing traditional Unix server tools (where most of these applications originated) being made available on consumer operating systems. Common infrastructure tools like Apache, MySQL, and PHP are readily available for modern versions of Windows, and Mac OS X (which is Unix underneath, after all). Because of this, we're seeing a great number of Unix packages being "ported" directly to Windows or Mac OS X, using the same installation and deployment techniques that have "worked so well" for years in Unix.

2.2. A Typical Example

A recent O'ReillyNet article [16] described in excruciating detail the author's experiences installing PhpWiki [11] on Mac OS X. PhpWiki is a pretty standard Wiki application written in the scripting language PHP. Wiki's are quite familiar to the Tcl community; they are fairly simple applications, and many Wiki and Wiki-like tools have been created, using a wide variety of tools and technologies. They are a very good example of the kind of personal or small group networked tool that would make sense to install on a personal computer.

This summary doesn't begin to do the full description justice, but here are some of the steps that were involved in doing the installation:

- download the latest source code
- open up Terminal.app (the Unix shell on Mac OS X)
- unpack the code in your Sites directory with "tar zxf"
- make sure Apache is running (done via a system preferences panel, since Apache actually is included with Mac OS X)
- via 'sudo' open /etc/httpd/httpd.conf in a text editor
- uncomment two lines to activate the PHP interpreter
- add four lines to tell Apache about PHP files
- restart Apache
- download MySQL (luckily someone packaged that one up nicely for Mac OS X, so its not much more than just a double-click to install)
- check that its running (another command line)
- set a password for the root account of MySQL
- setup a user for PhpWiki in MySQL
- create a database for PhpWiki to use
- do some other MySQL setup for PhpWiki via a SQL script provided in the PhpWiki distribution
- edit the index.php file to tell PhpWiki the database name and password, as well as where to find the database
- open your web browser and type in the address of your machine, followed by "~" and your username then "/phpwiki/" and PhpWiki's home page will load

2.3. Deployment Problems and their Costs

As you might expect, for a platform where the user's expectation of running new software is supposed to be drag-and-drop then double-click, and where most users don't even need to know how to get to the Unix shell, let alone how to use it, this kind of installation is simply just wrong. Unfortunately, its not just PhpWiki that suffers from this, but many hundreds of other tools, most originally developed on Unix.

It is probably unfair picking on PhpWiki here, as the same example could be made of any number of other tools. The software was never explicitly designed to run on Mac OS X, which admittedly lives in a world all its own. Many of the steps above would still be required on traditional Unix platforms though, and are particularly sensitive to changes in system configuration (hardly unheard of on Linux for example). For a software package touted as ready-to-run right out of the box, that is perhaps stretching the truth somewhat.

Generally, the developers of these tools are very talented individuals, who have put great effort into their creations. In many ways it's a huge shame that complicated deployment options are keeping their work from benefiting a much broader audience. It's a tribute to the quality of the software that enough people are willing to persevere through complicated installations to get the benefits of using the software, even like with PhpWiki on Mac OS X, where it was never explicitly designed to run. But those people are definitely in the minority, particularly on consumer-oriented platforms.

The cost of this is a huge amount of wasted time and effort for many users, and for the developers a lost opportunity to have their work used by others (critical for a commercial software venture, but equally valuable for most open source projects). Users are going to simply abandon their attempts to install such software, or never attempt it in the first place.

The remainder of this paper discusses strategies for building these applications in a way that they can be easily deployed. First though, it introduces the applications built at CourseForum Technologies where easy deployment was a critical requirement.

3. Case Study: CourseForum / ProjectForum

The remaining sections will often be referring to CourseForum [6] and ProjectForum [12], two commercial network server applications that were developed. These virtually identical applications, shown in Figure 1, provide secure web-based collaboration forums intended for use by online courses or corporate/organizational work groups.

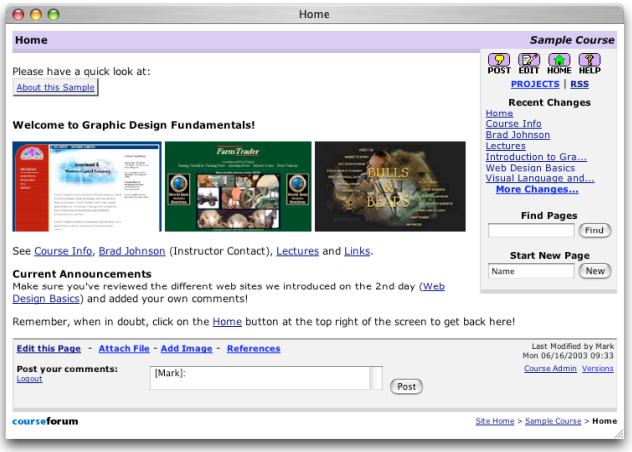

Figure 1. ProjectForum Screenshot.

These applications share many common features with Wiki applications like the aforementioned PhpWiki, or the Tcl based wikit [18]. They are organized around web pages which can be edited in a web browser using simple markup, support links, search, recent changes and so on.

CourseForum and ProjectForum go beyond the basic Wiki by supporting multiple forums (wikis) hosted on the same server, projects (sub-wikis), various authentication and security models, file and image attachments, user tracking, version management, advanced administration tools and more (some of which can be found in other systems, both commercial and open source).

Both systems are designed to run on Unix (particularly Linux and FreeBSD), as well as Windows (95/98/ME/NT/2000/XP) and Mac OS X. The target audience is the sort of users described earlier, who are not necessarily technically proficient when it comes to software installation, networks, etc.

CourseForum for example is targeted at distance education, and is most often set up and run by individual instructors for a course. ProjectForum is most often set up by an individual within a small company or organization, a user within an individual department of a larger company or organization, or by individuals wanting to collaborate on a project with friends or colleagues elsewhere. Less commonly, but no less effectively, both products are also run by network administrators in more traditional centralized server environments.

The software is distributed and sold through Internet distribution. Potential users typically hear about the software through word of mouth, a news release, etc., will visit the website, download the software, and obtain a time-limited demo license to try it out for a few weeks. Based on a favorable impression during the trial period, they might then decide to purchase a license to continue using the software.

Given the background of the target audience, and the importance of a positive first impression, any problems in deployment would have significant negative effects.

4. Starkits

Starkits [9,19], an evolution of earlier work on scripted documents [20], are a set of technologies that solve a number of deployment related problems for Tcl developers.

Briefly, Starkits can bundle together all the elements of a Tcl application into a single platform-independent file, which can be copied onto a user's machine. The Starkit retains the benefits of both source and binary distributions. Starkits can be easily unpacked, changed and repacked to get access to the source. With the addition of a platform-dependent run-time executable called Tclkit, the application in the Starkit can be run. A special Starkit, called a Starpack, can be easily made that combines the platform-independent Starkit with the platform-specific Tclkit. This produces a single file executable containing the application, tailored for that particular platform.

Technically, the Tclkit runtime, which exists for dozens of platforms, consists of the base Tcl language, the Tk GUI toolkit, the [incr Tcl] object extension, the Metakit embedded database, Zlib for compression, and TclVFS, providing a virtual filesystem-in-a-file. All of these facilities are automatically available to Tcl applications running in Starkits.

Starkits provide a lot of benefits when it comes to deployment: platform-independence, easy platform-specific binaries when required, access to source, many built-in facilities which reduce the need to depend on other packages, and a convenient way to effectively mirror your entire development environment and all the pieces of your application onto a user's machine, just by copying a single file.

Because of this, Starkits and Starpacks are the natural basis for building cross-platform, easily deployable network server applications in Tcl. But just using Starkits doesn't guarantee your application will be easily deployable. The rest of this paper describes how to leverage Starkit technology to make sure your application is easy to deploy.

5. Building Deployable Server Apps

This section will describe some of the main approaches that were taken in CourseForum and ProjectForum to make them easier to try out and deploy for users. These include removing all external dependencies, paying attention to installation and configuration, and building in dual solutions for starting and stopping the server. The next section will raise several platform-specific issues, which tend to be refinements of those described here.

As will become clear, thinking about deployment at the outset of a project is absolutely critical. The earliest decisions about how the application will be developed can have a critical impact on its deployment. As with many characteristics of software systems, factoring in deployment up front can save considerable time and effort later on.

The techniques and approaches described here are best seen as suggestions, tips, or recipes. They may provide direct guidance, but may also be modified, augmented or discarded depending on the application. Together they form more of a "cookbook" or a checklist of issues and approaches to be considered, rather than unique or novel solutions to problems.

5.1. Removing Dependencies

Removing the need for dependencies on external software or a specific system configuration is clearly one of the most dramatic steps that can be taken by developers to make their software easier for end users to deploy.

As an example, note that the Wiki application from earlier depended on the following software packages, separate from the application itself:

- Apache web server
- PHP scripting language module for Apache
- MySQL database

It is not unusual to find other similar applications that require many more other packages, including specific third party utility libraries or modules, external tools like CVS to provide version control, etc. This is particularly common given our heavy reliance on frameworks, application servers and multi-layered architectures to build web applications today. Further, each of these packages requires further configuration, and in many cases, an additional installation.

The presence of each additional package, and usually a very particular version or configuration of each, provides more opportunities for mistakes to be made during the installation. Further, it means that if any of these other packages is modified (through routine maintenance or updates, or because of use by another application requiring them), there is the possibility of that change breaking our application.

It should be clear that making decisions on which if any external tools to rely on is something that must be done as early as possible in the overall software development process. It is equally clear that deciding to restrict use of external packages will usually require making huge tradeoffs. The reason these packages are so often used in the first place is because they provide immense value to developers, saving considerable time and allowing us to leverage and reuse the expertise and effort of others. Relying on these packages can be the difference between developing a successful application or not. Clearly, the many benefits provided by using external packages must be carefully weighed against the increased burden that users must bear to deploy the resulting application. These decisions are complex, and have more to do with marketing or other outward-looking considerations than strictly development.

Bundle Everything. One solution to this problem is simply to bundle everything you need in your application, so that when your application is installed, so are all the other packages it depends on. This is of course commonly done, particularly with code libraries, shared libraries, and so on.

It is also not uncommon in many larger applications (e.g. complex bulletin board systems, educational course management software), where an installation will often install private copies of web servers, database tools, and dozens of other packages within the main application's installation directory.

Taking this approach preserves the benefits to developers, who can rely on all the other packages, and whose only cost is to write a fairly complex setup application. For users, this complexity is mostly hidden.

The downside though can often be a considerably larger package to download and install, often several hundred megabytes, and a greater impact on the user's machine. For something like running a small Wiki on a desktop machine, it may seem overkill to need to download a 300 megabyte application, which when run starts a dozen extra processes and daemons running on the machine.

Find Minimalist Alternatives. The other approach is to find alternatives to external packages that are more minimalist, and designed to be embedded within other applications (i.e. as Tcl packages). Do you really need the full power and generality of Apache and its dozens of modules, or can your application be developed on top of a framework like Tclhttpd?

Once identified (or developed), these embeddable alternatives can simply be included in the Starkit that contains the rest of the application. This is a key benefit we get from using Starkits; using Tcl packages from within a Starkit is as easy as using those installed directly on your disk [9].

For CourseForum and ProjectForum, the following minimalist alternatives were used to provide key functionality normally provided by external tools:

- *Web server*: one of the minimalist web servers included in the Tclhttpd [14] distribution was used, modified to call an application-provided routine to process all requests; this routine would then call the various objects in the application
- *CGI libraries*: our web-page processing was based on a simple "subst"-based parser, similar to that used for TclHttpd's TML files, coupled with several small utility libraries developed in house for standard components like forms, etc.
- *Database*: the applications naturally took advantage of the Metakit database library automatically provided by Tclkit, rather than using an external database
- *Version control*: while many systems rely on a separate CVS installation, the fairly minimal version control needs were easily met by a small custom Tcl library built using Metakit

Clearly these alternatives are nowhere near as powerful as the full-featured systems we might conventionally use. But they are more than sufficient *for our application*. For the specific requirements of any single application, the savings in deployment complexity usually offset the minimal gains a more general solution would provide.

The resulting applications, including the Tclkit runtime, dependent libraries, application code, web pages and associated data files, all fit within a single executable approximately two megabytes in size.

5.2. Installation and Configuration

A painless initial setup and install is an important part of deployment. This is especially true for applications where an initial trial is a key part of the decision for whether or not to use the application.

Starkits hold the promise of installation being as simple as a copy and uninstall a delete [20]. However, modern user expectations that grew from too much complex software dictate that things are usually not this easy.

To conform to platform expectations, on Unix platforms, our single binary (a Starpack) is wrapped up in a gzipped-tar file. On Windows, we install via a simple installer/setup program. On Mac OS X, we bundle the application binary into a compressed disk image, where installation actually is nothing more than copying the file from the disk image to the hard drive. These will all be discussed shortly.

Extensive configuration is common in web and network based applications. Limiting most of the dependencies on external components should minimize that as much as possible. Relying on sensible defaults, and not necessarily requiring information about local hostnames or IP addresses — which may have implications on how the application is coded — can prevent tripping up many users.

Rather than relying on hand editing of text-based configuration files, the use of sensible defaults, coupled with web-based configuration systems built into the main application (see Figure 2) is preferred.

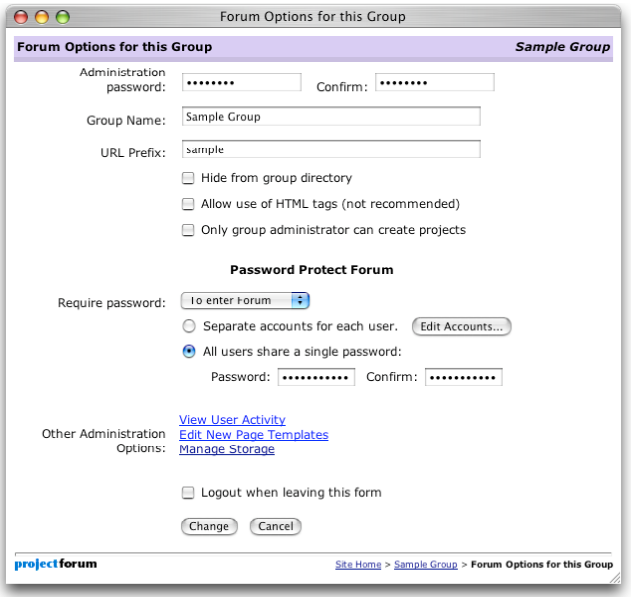

Figure 2. Web-based configuration in ProjectForum.

5.3. Starting and Stopping

Server applications typically run as "daemons" — "background" or "faceless" applications that present no direct user interface. Instead, when started, they simply listen on the appropriate network ports, and respond to connections and requests from clients (e.g. web browsers). This makes sense for any number of reasons, particularly in the conventional data center environment.

For users not as familiar with network server applications though, this model of use can be problematic. It raises questions like:

> "I double-clicked on the application, and it looked like it opened, but I didn't see anything. Did it crash?"

"Is it running?"

"How do I use it?"

"How do I stop it?"

On Unix environments, where applications are usually run from the command-line, this wasn't much of an issue; users know about processes, can see if the application is still running in the shell, and a useful startup message can be enough to tell them where the server is listening (see Figure 3).

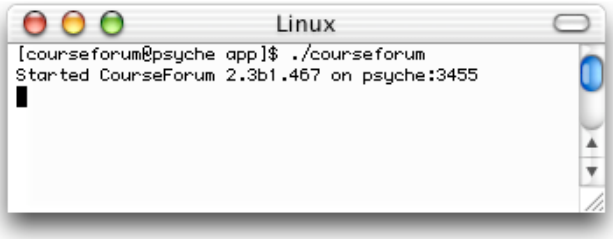

Figure 3. Starting CourseForum on Linux.

On Windows and Mac OS X, where applications are started by double-clicking (and where no windows coming up after double-clicking does usually signify a crash!), a small "launcher" graphical user interface is provided instead (see Figure 4).

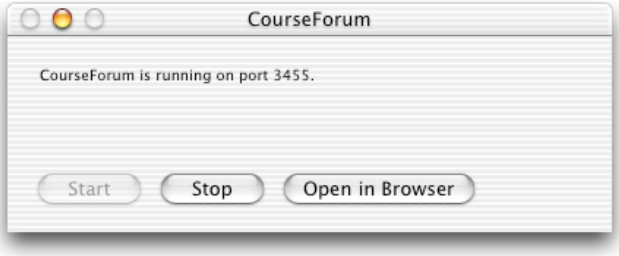

Figure 4. Starting CourseForum on Mac OS X.

This window can at a glance provide feedback that the server is running, where it is running, how to stop it, and provides a way to immediately access the application, by opening the user's web browser and loading the home page provided by the application (which is done automatically the first time the application is launched).

Common startup errors (such as a port already in use), port changes, etc. are also handled directly in the GUI (see Figure 5).

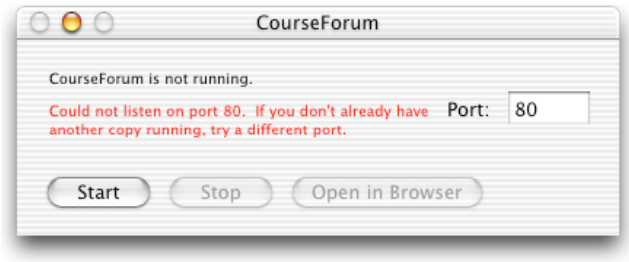

Figure 5. Handling startup errors in the GUI.

In addition to the simple GUI that first time users encounter, it is also still possible to run these servers as regular background applications; how to do this is described in the applications' administration guides and online support resources. The details of doing this vary by platform, and are discussed shortly.

6. Platform Specific Issues

Tcl and Starkits do a spectacularly good job of allowing developers to forget about the differences between platforms, hiding the details of things like networking code that can vary greatly. However, for network server applications, there are a few additional details that need to be resolved.

6.1. Unix

Not surprisingly, Unix platforms such as Linux, FreeBSD, Solaris, etc. do not provide a lot of surprises or challenges for this type of application. Long-running daemon processes, for example running at system startup time or out of /etc/inittab, are straightforward.

Privileged ports. The one additional bit of code required has to do with running on privileged network ports (i.e. those below 1024), which requires root access. However, running everything as the root user presents security risks that may be unacceptable to many users.

Rather than having the user run as root, the binaries are marked as "setuid", which allows them to temporarily change to root as needed (i.e. just to listen on the privileged ports), and then to change back to a lesser-privileged user immediately after.

These features are not built in directly to Tcl; we wrote a tiny C extension which just serves as a wrapper around the getuid(), geteuid(), setuid() and seteuid() system calls needed to implement the appropriate behavior. This extension was then of course included into the Starkit used to hold our application.

6.2. Windows

Windows (CourseForum and ProjectForum run on versions from Windows 95 and higher, just like Tcl/Tk) was also quite straightforward, but again there were a few areas to address.

Launcher User Interface. For the simple graphical start/stop interface that comes up when the application is double-clicked, we simply used a short Tk script (see Figure 6), integrated directly into the main server application.

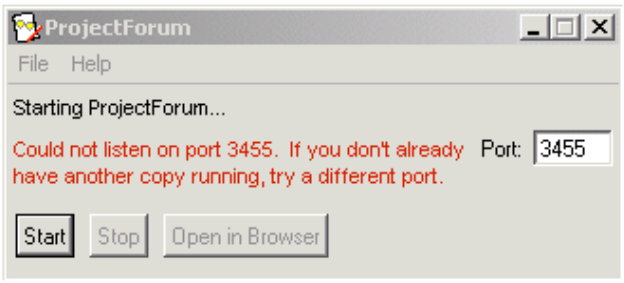

Figure 6. Launcher User Interface on Windows.

Installers. From the platform that originally brought you "DLL Hell", it's probably no surprise that applications must have installers and uninstallers to be accepted by users. There are many tools to help write such programs; CourseForum and ProjectForum use the scriptable NSIS installer [10].

Windows Services. The accepted way to run long-running processes (daemons) on Windows platforms, especially on NT/2000 and above, is to use the "Services" facilities provided by the operating system. This provides a monitoring facility, the ability to automatically run the program at startup, start and stop it via a control panel, and so on.

The ability to have an application run as a service is not directly built into Tcl, and does require some small code changes. There is an extension, TclSvc [13], which will provide all the needed hooks into Windows. You will need to provide for mechanisms to install and remove the service from the system. Building in support for services directly into the application is the preferred approach.

An alternative approach (and the lazy one that we've taken to date) is to require an extra piece of software — introducing a dependency on another software package — for those who need to run the software as a service. The excellent FireDaemon package [8], provides an easy way to turn any Windows application into a service.

6.3. Mac OS X

The last platform to examine is Apple's Mac OS X, the newest major Tcl/Tk port. Most Tcl developers may have only passing familiarity with OS X. This platform may prove particularly valuable for developers, but without a doubt requires the most effort to do well.

A Large Opportunity? Mac OS X [4] is the latest incarnation of Apple's user-friendly consumer operating system. A radical departure from earlier versions, OS X is at its heart BSD Unix, along with many advanced technologies deriving from the NeXT acquisition several years ago. A common refrain is that OS X brings Unix power to the masses.

Though the user base is considerably smaller than Windows (perhaps about 5% of the market, probably just a bit larger than Linux and other Unix variants), it can be an attractive market for small developers. Mac users have a culture of rewarding innovation, supporting small developers, and making it very easy, through a variety of online resources, for developers to get the word out about their applications.

Mac users are also probably the most demanding when it comes to software working smoothly, and for them "port" is definitely a four-letter word. Software has to be easy to use, follow platform conventions, and work well to be successful.

For Tcl, one of the easiest to deploy — thanks to Starkit — of all the scripting languages that originated on Unix, the opportunity to deploy applications on a popular Unix-based consumer operating system should be seriously considered.

Developer Tools and Documentation. Developing on the Mac, even with Tcl/Tk, does require some use of Mac OS X documentation, and Mac native developer tools. Luckily, both are free and abundant (if somewhat daunting at first), and can be found at Apple's Developer site [5].

Most OS X applications are usually developed using Objective C, using an application framework (class library) called Cocoa. The compiler is a version of gcc, though most people encounter it only from within an IDE called ProjectBuilder. The aptly-named Interface Builder application provides an easy way to quickly build user interfaces that follow platform guidelines, and that can be easily attached to underlying code.

User Interface. The first major issue we encountered was user interface quality. While Tcl itself is completely solid on OS X, unfortunately the Tk port for Mac OS X is still very new. While functional, it lacks much of the refinement expected in typical Mac applications. As an example, Figure 7 shows the same Tk launcher script we used on the Windows version, but instead running on OS X.

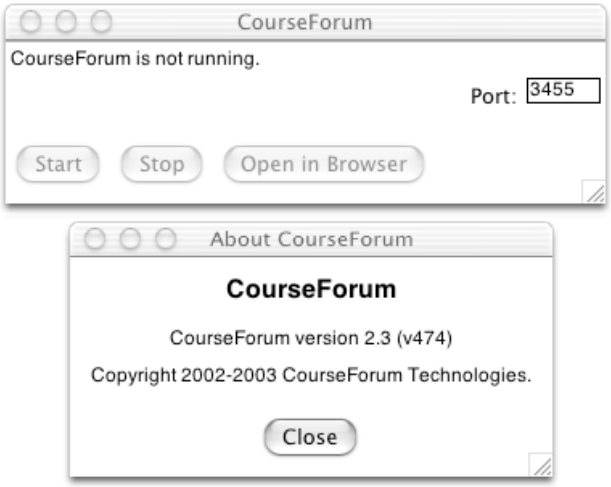

Figure 7. Tk Launcher Script on Mac OS X.

Because we had limited GUI needs, we decided to just punt on Tk, and developed the front end in Cocoa using Objective C and Interface Builder. Despite being unfamiliar with the tools, this took less than one day, and the resulting Objective C code is only about 200 lines long, including all the interconnections with the actual server program (more on that in a moment). The interface, which follows platform conventions much more closely, is shown in Figure 8.

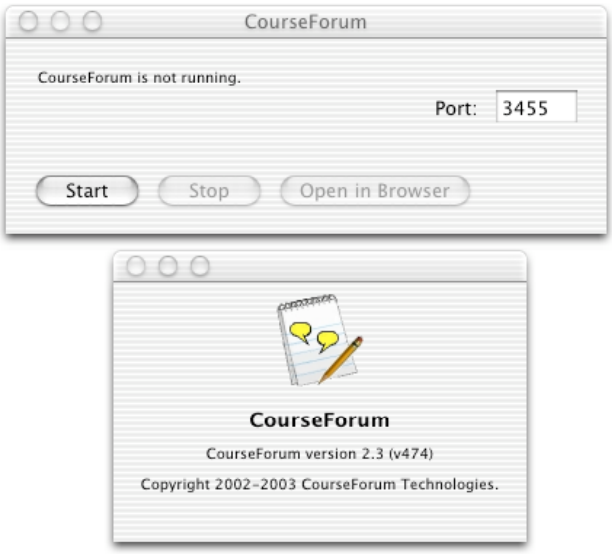

Figure 8. Native Cocoa version of Launcher.

Application Structure. To understand how the Cocoa launcher GUI fits in to the actual server, it is necessary to understand what an application on Mac OS X looks like. While it is just Unix underneath (so regular Unix binaries are present), most applications that end users deal with are actually *application bundles*. An application bundle is not a single file, but an entire directory, which appears in the Finder (the File Manager) as a single file. Inside the directory are various resources (icons, version info, etc.) as well as traditional binary executables. This is shown in Figure 9.

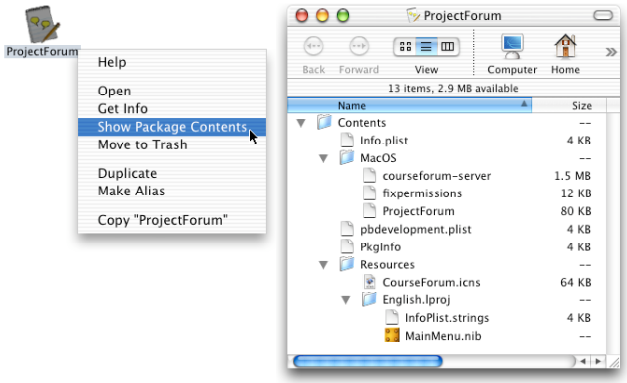

Figure 9. Application Bundles on Mac OS X.

Looking at the window on the right, the "ProjectForum" file is the main application executable, which we created in Interface Builder. This is what is run when the user double-clicks on the application's icon. But the "courseforum-server" file is another executable. In this case, it is actually a Starpack, combining a Mac OS X version of Tclkit along with our application-specific code and data files.

The two pieces communicate by opening a pipe between them, using the Cocoa framework's facilities to do the equivalent of Tcl's "open |" plus "fileevent". So port changes, status information, etc. are communicated as simple text messages written across the pipe. Yes, it's definitely Unix underneath!

Granting Privileges. Splitting the application into two separate executables turns out to be the recommended approach to solving another problem, the issue of running on privileged ports on Unix systems that was discussed earlier. Under OS X, users do not open the terminal and manually change permissions with chmod (though its possible).

Instead, when permissions are first needed, our front-end Cocoa app makes a system call to prompt the user for the needed authorization (system password), and then calls a tiny helper binary (fixpermissions) which actually does the setuid changes on the courseforum-server binary. This all involves a series of obscure but well-documented utility routines provided by Cocoa.

Creating a StartupItem. Under OS X, a "StartupItem" is the equivalent of a Unix daemon that starts at system startup time (e.g. out of rc.d or /etc/inittab) or a Windows service. This is simply a directory, placed in a well-known location, containing a couple of text files (scripts) that point to your Unix-style executable file (i.e. courseforum-server).

Installers. Mac OS X applications are typically delivered as disk images. Like TclVFS, a disk image presents a "file system in a file" facility. The downloaded file is "mounted" and appears much like a CD or other removable media as a volume on the desktop. Applications can be run right from the disk image, or copied to the local machine's Applications folder. This simple model is greatly preferred by users. When absolutely needed, there are also other facilities available for doing more complex installations, akin to what would be seen in Windows installers.

7. Other Considerations

This section briefly touches on two other areas that are relevant for building easy to deploy network server applications: integration with existing web servers, and several build issues.

7.1. Integration

For web-based applications such as CourseForum and ProjectForum, integration with existing web sites and web servers is a question that can frequently arise. For example, many people may already be running a web server on the machine that they are installing the new application on.

Avoiding Conflicts. By default, our applications listen on ports not likely to be already used (3455, also above 1024 so that issues around privileged ports do not arise immediately). This is very easy to change via the launcher user interface, the web configuration, or a command-line flag on Unix platforms. If no existing web server is already running, this can be changed to port 80, the standard HTTP port.

Closer Integration. Often though, there is a requirement that both the existing web server and the web server in the new application be accessible over port 80. This may be simply to present simpler URL's (i.e. without the port number included), but is most often needed when firewalls are an issue. Firewalls may be located on the network where the machine running the application is kept (keeping outside traffic away), or on far-away networks where some users would like to connect from. The strictest of these will only allow HTTP traffic on port 80, and block everything else.

This can usually be solved through some sort of virtual hosting or proxy mechanism. We assume the existing web server can delegate specific requests (e.g. to a particular named host, or to a subdirectory on the main site), passing them on to another web server (i.e. our application running on some other port) to handle.

This can be done in Apache using the "ProxyPass" directive, with Microsoft's ISA server, with a dedicated proxy server, etc. Our FAQ [7] contains some notes on how to set this up.

When allowing proxying from a subdirectory (e.g. "http://www.foo.com/forums/" would proxy to the toplevel directory of the application), particular care must be taken to rely on relative URL's only, and not absolute URL's (those that start with a "/") to refer to resources in your application. Similarly, applications should not have dependencies on the specific host name they're running on, nor the host header passed in by the browser (which may be the proxy server, not the actual site). There are many special cases, and extensive testing is required.

Branding. A further level of integration with existing web sites may be desirable: the ability to match the appearance of the web application to an existing web site. This may be so it appears to be part of the same site, retains a standardized corporate look, etc.

To accomplish this, a mechanism to customize the appearance of your application must be provided. Items to pay attention to include images, colors, fonts, header and footer graphics, etc. Ideally, there should be customization screens in your application to allow for this. A reasonable fallback (since it is rarely needed, and usually only by more sophisticated users) is to have the application look in a special directory for customization files. These may include replacement stylesheets, images, or chunks of HTML that will be included at specific points in pages (easy if you programmatically generate your pages using standardized headers, footers, etc.).

7.2. Build Environment

A well-designed, automated build and test environment can be of tremendous help to all development teams, no matter the size. Given the complexities of building cross-platform server applications, the more that can be automated, the more mistakes that will be avoided. This section briefly describes the build environment used for CourseForum and ProjectForum.

Automated Builds. Tcl and Starkits provide a great solution already; scripts don't need to be compiled, and using "sdx wrap" you can package your development environment's entire directory structure into a finished Starkit.

However, there can be benefit to automating things further. Our build script (written in Tcl of course) performs the following steps, automatically every night on a server machine, and also on demand:

- Check out a fresh copy of the entire code tree from the version control repository (CVS) into a new directory
- Delete any unneeded files (e.g. developer notes, CVS directories)
- Automatically update a build number

- Do any needed code obfuscating (see below)
- Run the full set of automated tests
- Build a (still platform-independent) Starkit
- Built platform-specific binaries and distributions (see below)
- Email a report detailing the build and test results

Platform Builds and Packaging. After the main build script runs, platform-specific distributions must still be created. Using the platform-independent Starkit produced from the main build, Starpacks for the Unix platforms are quickly created, and then the final distributions packaged into a gzipped tar file. These steps are actually integrated into the main build (i.e. so it's one command to invoke). Also note that all Unix builds are done from a single (Linux) machine; Starpacks for other platforms can be easily created if the base Tclkit for the target platform is available.

We perform the finishing touches on our Windows and Mac OS X builds on separate machines. For Windows, we copy the Starkit created by the main build, turn it into a Starpack, and then create the installer via a NSIS script [10]. The resulting setup program is then copied back to the original machine so that all the builds are collected in one spot. Mac OS X is similar, but we also need to compile the launcher program using Project Builder, and build final disk images. Again, all of these are completely automated, so are invoked with only a single command.

Testing. CourseForum and ProjectForum rely very heavily on automated testing, with test cases typically being developed before starting development of code for new features (test-driven development), and before fixing bugs. Run automatically every day in the nightly build, and on demand, the result is an ever-growing suite of regression tests that can be used to quickly pinpoint changes that break existing code.

Network server applications can benefit both from unit testing (white box) on the underlying code modules, as well as higher-level acceptance testing (black box), where the application's features are exercised via the network/web interface. We used Tcltest, augmented with a set of wrappers around Tcl's http package, utilities for cookies, HTML form parsing and submission, etc. Other alternatives, e.g. tclwebtest [15], are also available.

The final builds on each platform are subjected to a quick "smoke test" before being released. They are manually installed on clean machines (or VMWare virtual machines) and briefly exercised according to a defined test plan.

Obfuscating Code. For commercial applications, code obfuscation may also be an issue. Our build script automatically byte-compiles all Tcl scripts using procomp from ActiveState's Tcl Dev Kit [1] before packaging the entire directory into the Starkit. The Starkit contains versions of tbcload for all of our supported platforms, and our main program tries to load the appropriate version at application startup time. The byte-compiled files are drop-in replacements for the original Tcl files, so the application's structure on disk is preserved, but the original code is not shipped.

8. Our Experiences

Deploying CourseForum and ProjectForum for our users has been much easier as a result of applying the techniques described here. The applications are very small and therefore quick to download, in line with users' expectations for this sort of application. Once downloaded, they are very easy to get started with, and work like other applications developed natively for the platform in question. Users just run the application and it works for them; they don't necessarily know that underneath it happens to use Tcl, Metakit, etc.

Perhaps an appropriate way to summarize the results in terms of deployment might be by noting a comment from the author of the O'ReillyNet article about installing PhpWiki on Mac OS X. On being pointed to ProjectForum, he gave it a quick try and then responded:

> *I've got one of those if-only-I'd-known-that-before sort of feelings...*
>
> *Nice work. You're right, it couldn't be simpler.*

These sorts of network server applications, run and managed by end users rather than network admins, represent an important and growing new niche. Tcl, probably more than any of the other available cross-platform scripting languages, has the potential to play a very strong role, because of its ability to be easily deployed.

With Starkit providing a solid foundation for Tcl deployment, this paper has suggested some additional techniques, many particular to this application domain, which build on the existing Starkit technology. Some are conventional and unsurprising, such as worrying about platform-specific packaging and installation. Others, such as adding a GUI interface to a server, may be less common. Most importantly, dependencies on external packages must be reduced in favor of Tcl-based embedded tools. While this goes against the trend today in web development to layer on more and more packages, it is critical when building network server applications that are oriented towards average consumers rather than network administrators.

Thinking about deployment early in design and development is very important. Tcl and Starkit, enhanced with some of the strategies discussed here, provide a great foundation for building powerful network server applications that anyone can install and use.

Acknowledgements

Thanks to Jean-Claude Wippler and Steve Landers for many useful discussions on this topic, and for feedback on earlier drafts of this paper. They have, along with many other users of the software described here, helped to both motivate and refine this work.

For any additional resources that may be available on this topic, please see http://www.markroseman.com/tcl.

References

[1] ActiveState. Tcl Dev Kit Home Page. http://www.activestate.com/Products/Tcl_Dev_Kit/.

[2] AmphetaDesk Home Page. http://www.amphetadesk.com.

[3] Apache Rivet Home Page. http://tcl.apache.org/rivet/.

[4] Apple Computer. Mac OS X Home Page. http://www.apple.com/macosx/.

[5] Apple Developer Connection Home Page. http://developer.apple.com.

[6] CourseForum Home Page. http://www.courseforum.com.

[7] CourseForum and ProjectForum FAQ. http://www.courseforum.com/faq/.

[8] FireDaemon Home Page. http://www.firedaemon.com.

[9] Landers, S. Beyond TclKit – Starkits, Starpacks, and other *stuff. *Proceedings of the 9th Annual Tcl/Tk Conference*. Vancouver, Canada. September, 2002. http://www.digitalsmarties.com/Tcl2002/tclkit.pdf.

[10] Nullsoft. NSIS Home Page. http://www.nullsoft.com/free/nsis/.

[11] PhpWiki Home Page. http://phpwiki.sourceforge.net.

[12] ProjectForum Home Page. http://www.projectforum.com.

[13] Sensus Consulting. TclSvc (download and home page). http://www.sensus.org/tcl/.

[14] TclHttpd Home Page. http://tclhttpd.sourceforge.net.

[15] Tclwebtest Home Page. http://tclwebtest.sourceforge.net.

[16] Turnbull, G. Installing a Wiki on your iBook. O'ReillyNet 6/05/2003. http://www.macdevcenter.com/pub/a/mac/2003/06/05/wiki.html.

[17] Userland Software Home Page. http://www.userland.com.

[18] Wikit Home Page. http://www.equi4.com/wikit/.

[19] Wippler, J.C. Repositories, deployment factories, binary code, and CDROM's. *Proceedings of the European Tcl Conference.* Munich, Germany. June, 2002. http://www.equi4.com/docs/munich/eurotcl2002.html.

[20] Wippler, J.C. Scripted Documents. *Proceedings of the 2000 Tcl/Tk Conference.* Austin, TX. February, 2000. http://www.equi4.com/docs/austin/scripdoc.html.