

Proposing a Standard Web API!

#244 FEBRUARY 1996

Dr. Dobb's

JOURNAL

SOFTWARE
TOOLS FOR THE
PROFESSIONAL
PROGRAMMER

Data Communications AND Internet Development

- JAVA COMMAND-LINE ARGUMENTS
- IMPROVING KERMIT PERFORMANCE
- DEBUGGING CGI APPS
- NETWORKING WITH WINSOCK 2.0

**IMPLEMENTING MULTILEVEL
UNDO/REDO**

NETWORKING INTELLIGENT DEVICES

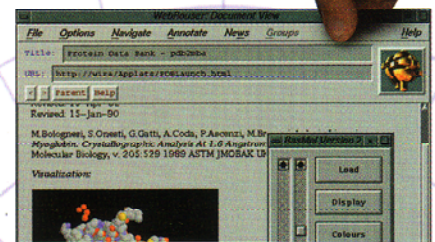
EXAMINING TOOLS.H++

\$3.95 (\$4.95 CANADA)



A Miller Freeman Publication

**BOOKS ON HTML,
ALGORITHMS,
AND MORE**



PROPOSING A STANDARD WEB API

by Michael Doyle, Cheong Ang, and David Martin

At last count, there were nearly a dozen APIs vying for hearts and home pages of Web developers. Our authors propose a standard API that leverages the concept of embedded executable content for interactive application development and delivery.

IMPROVING KERMIT PERFORMANCE

by Tim Kientzle

Tim compares the error-handling strategies of a variety of popular protocols, then presents heuristics that improve the performance of Kermit's windowing strategy.

CGI AND THE WORLD WIDE WEB

by G. Dinesh Dutt

The Common Gateway Interface (CGI) makes it possible for Web servers to interact with external programs. Dinesh presents a program that reports gateway-execution errors.

USING SERVER-SIDE INCLUDES

by Matt Kruse

Server-side includes are commands embedded inside HTML documents that enable your page to do something different each time it is loaded. Matt describes the format of these commands and shows how to write programs that work with your Web pages.

JAVA COMMAND-LINE ARGUMENTS

by Greg White

Greg introduces a package of Java classes that parse the command-line parameters for HtmlXlate, an application that converts HTML to RTF. Because HtmlXlate doesn't require display graphics, Greg made it an "application" instead of an "applet."

IMPLEMENTING MULTILEVEL UNDO/REDO

by Jim Beveridge

The Undo/Redo mechanism Jim presents here is based on a history length limited only by available memory. Because it is implemented in Visual C++ and MFC, this mechanism can easily be added to your applications.

18

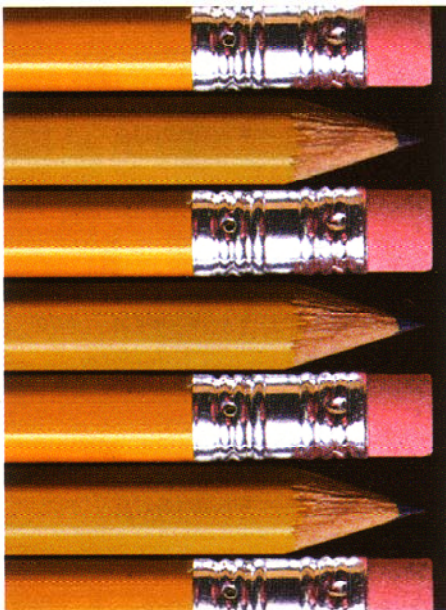
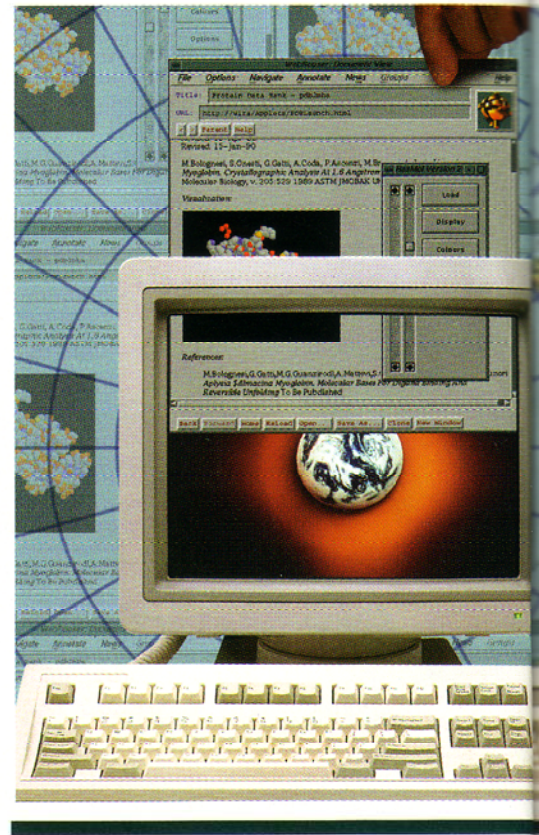
28

42

52

58

64



EMBEDDED SYSTEMS

NETWORKING INTELLIGENT DEVICES

by Gil Gameiro

Novell's Embedded Systems Technology (NEST) lets you incorporate network protocols and client services into embedded systems. Gil uses NEST to put an intelligent coffee maker online, then controls it with a Windows-hosted menu program.

68

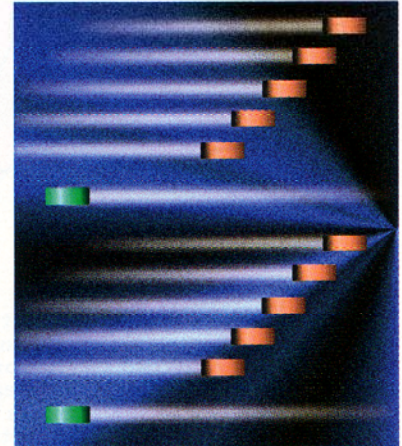
NETWORKED SYSTEMS

FAST NETWORKING WITH WINSOCK 2.0

by Derek Brown and Martin Hall

Derek and Martin show how you can get maximum performance from WinSock 2.0 applications by taking advantage of two features new to the spec—event objects and overlapped I/O.

76



EXAMINING ROOM

EXAMINING ROGUEWAVE'S TOOLS.H++ 80
by P.W. Scherer
RogueWave's Tools.h++, a C++ library consisting of more than 100 classes, has been the cornerstone of Perry's development efforts ever since he ported over 30,000 lines of C++ code to an equivalent app that was only 6000 lines long.

PROGRAMMER'S WORKBENCH

LEX AND YACC 86
by Ian E. Gorman
Ian describes how he used traditional compiler-development techniques and the MKS Lex & Yacc Toolkit to build a keyword-query compiler for a CD-ROM database.

COLUMNS

PROGRAMMING PARADIGMS 117
by Michael Swaine
Before continuing his examination of little languages for the Macintosh, Michael looks at a number of books devoted to HTML coding.

C PROGRAMMING 121
by Al Stevens
Al launches "Quincy 96," a C/C++ interpreter that runs under Windows 95 and is based on GNU C and C++. Among other features, Quincy 96 manages two kinds of documents—project documents and text source-code documents.

ALGORITHM ALLEY 135
edited by Bruce Schneier
Binary searches are algorithmic staples that can be used in just about any program. Micha Hofri sees how efficient he can make a basic binary-search algorithm.

PROGRAMMER'S BOOKSHELF 139
by Dean Gablon
Dean compares *Practical Algorithms for C Programmers*, by Andrew Binstock and John Rex, and *Practical Algorithms in C++*, by Bryan Flamig.

FORUM

EDITORIAL 6
by Jonathan Erickson

LETTERS 10
by you

SWAINE'S FLAMES 144
by Michael Swaine

PROGRAMMER'S SERVICES

OF INTEREST 142
by Monica E. Berg

SOURCE CODE AVAILABILITY

As a service to our readers, all source code is available on a single disk and online. To order the disk, send \$14.95 (California residents add sales tax) to *Dr. Dobb's Journal*, 411 Borel Ave., San Mateo, CA 94402, call 415-655-4100 x5701, or use your credit card to order by fax, 415-358-9749. Specify issue number and disk format. Code is also available through the DDJ Forum on CompuServe (type GO DDJ), via anonymous FTP from site ftp.mv.com (192.80.84.3) in the /pub/ddj directory, on the World Wide Web at <http://www.ddj.com>, and through DDJ Online, a free service accessible via direct dial at 415-358-8857 (14.4 bps, 8-N-1).

NEXT MONTH

Little languages are a big deal to most programmers—and they're the focus of our March issue.

Not Without Controversy

You'd think that someone who's devoted his professional life to writing about "serious" literature would be immune to controversy. But then Sven Birkets, noted critic, author of the acclaimed book *The Gutenberg Elegies: The Fate of Reading in an Electronic Age*, and self-appointed latter-day Luddite has an excuse—it's his editor's fault. Isn't it always.

Birkets's *The Gutenberg Elegies* is a wonderfully written, thought-provoking examination of the impact of digital technology on literature and reading. Presumably, it was the book—in particular the final chapter entitled "Coda: The Faustian Pact"—that led to Birkets's participation in an August 1995 *Harper's Magazine* forum dubbed "What Are We Doing On-line?". Along with the Electronic Frontier Foundation's John Perry Barlow, *Wired* magazine's Kevin Kelly, and like-minded author Mark Slouka, Birkets debated in print the pros and cons of the "message of this new medium," the Internet. Birkets was the resident skeptic.

Still, nothing Birkets said in *Harper's* matched his "Coda" chapter, which ends with the exhortation "Refuse it." I'll let you guess (or at least read on your own) what he wants us to shy away from. Birkets may be on to something when he says, "my core fear is that we are...becoming shallower" and "our whole economic and technological obsession with getting on-line is leading us away...from the premise that individualism and circuited interconnection are, at a primary level, inimical notions." In any event, it is entirely proper to question basic premises as our online experience evolves.

Letters to the editor in subsequent *Harper's* were hot and heavy, with Birkets even having Rilke poems thrown back at him. It's doubtful that analyzing imagery in the works of Virginia Woolf ever created such a firestorm.

All of this led me to go hear Birkets when he showed up at a local bookstore. He agreed, for instance, that the "Coda" chapter was the most controversial—and newsworthy—part of the book. More interestingly, "Coda" wasn't part of the original manuscript—it was added when his editor sensed something was lacking. What the book needed, said the editor, was a more powerful ending. Birkets returned to his Smith-Corona (what else?) and wrote "Coda," launching him on to the literati equivalent of day-time talk shows.

While Birkets's editor is possibly an exception, sometimes we editors do end up eating crow. Not that I'd ever fess up to feasting on such foul fowl, but even editors can change their minds once in a while.


For instance, regular readers of this space (both of you) may recall that in the November 1995 issue, I made a smart-aleck remark about a University of California software patent covering embedded executable content ("applets") and the World Wide Web. Shortly thereafter, I heard from Michael Doyle, chairman of Eolas Technologies and co-inventor of the patented technique. Michael set me straight on the licensing terms of patent, pointing out that Eolas, which holds exclusive rights to the patent, is not asking programmers to pay royalties for developing software that runs applets. Instead, Eolas is only requiring that developers adhere to a standard API for Web development.

Michael gladly put his comments in writing, which we published in the January 1996 "Letters" column. For this issue, Michael—along with Eolas cofounders Cheong Ang and David Martin—wrote our lead article, which delves into the history and technical underpinnings of their work.

By using the patent as a carrot instead of a stick, Eolas has taken a step in helping programmers get on with the job of developing next-generation, interactive Web applications. Clearly, supporting a single standard API is better than tinkering with a dozen or so competing ones. (This problem of dealing with competing APIs is partly behind the current campaign to eradicate those annoying "enhanced for Netscape" tags.) The bottom line is that both developers and users want a standard.

This isn't to say that I've changed my mind about software patents. I've yet to see how they've helped the software industry move forward. On the surface, however, the Eolas proposal may be an exception. Of course, there's nothing to say that Eolas will be successful. In all likelihood, browser vendors will continue to plod along their proprietary paths, much like operating-system vendors of a decade ago (remember TRS-DOS?).

You don't have to agree or disagree with the concept of software patents to appreciate the spirit of Eolas's proposal. If nothing else, Doyle and crew should be commended for coming up with a creative approach to a thorny problem. Doyle's article—and the proposal it makes—is certainly one of the more controversial pieces we've recently published. I look forward to hearing from you about both the article and the proposal.



Jonathan Erickson
editor-in-chief

Proposing a Standard Web API

Short circuiting the API wars

Michael Doyle, Cheong Ang, and David Martin

The World Wide Web has matured from a relatively limited system for passive viewing of hypermedia-based documents into a robust framework for interactive application development and delivery. Much of this progress is due to the development of embedded executable content, also known as "inline Web applets," which allow Web pages to become full-blown, compound-document-based application environments. The first Web-based applets resulted from research begun in the late 1980s to find a low-cost way to provide widespread access for scientists and educators to remote, supercomputer-based visualization systems.

The Visible Human Project

In the late 1980s, the National Library of Medicine began a project to create a "standard" database of human anatomy. This "Visible Human Project" was to comprise over 30 GB of volume data on both male and female adult human anatomical structures. It was one of the original Grand Challenge projects in the federal High-Performance Computing and Communications initiative, the brainchild of then Senator Al Gore. As a member of the scientific advisory board for this project, one of us (Michael Doyle) became interested in the software issues involved in working with such a large database of the most detailed image information on human anatomical structure yet available. His group in the Biomedical Visualization Lab (BVL) at the University of Illinois at Chicago realized at the time that much research would have to be done to make such a vast resource both functional and accessible to scientists all around the world.

Until that time, medical visualization systems were designed to work on 3-D datasets in the 15–30 MB range, as produced by the typical CT or MRI scanner. High-end graphics workstations had adequate memory capacity and processor power to allow good interactive visualization and analysis of these routine

The authors are cofounders of Eolas Technologies Inc. and can be contacted at <http://www.eolas.com>.

datasets. The Visible Human data, however, presented an entirely different set of problems. To allow widespread access to an interactive visualization system based upon such a large body of data would require the combined computational power of several supercomputers, something not normally found in the typical biomedical scientist's lab budget.

Doyle's BVL group immediately began to work on solving the information-science problems related to both allowing interactive control of such data and distributing access to the system to scientists anywhere on the Internet. Our goal was to provide ubiquitous access to the system, allowing any user connected to the Internet to effectively use the system from inexpensive machines, regardless of platform or operating system.

The Promise of the Web

We saw Mosaic for the first time when Larry Smarr, director of the National Center for Supercomputing Applications, demonstrated it at an NSF site visit at BVL in early 1993. We became immediately intrigued with the potential for Mosaic to act as the front end to the online visualization resource we had been designing. Immediately after Michael Doyle left the University of Illinois to take the position of director of the academic computing center at the University of California, San Francisco, we began enhancing Mosaic to integrate it with our system. We designed and implemented an API for embedded inline applets that allowed a Web page to act as a "container" document for a fully interactive remote-visualization application, allowing real-time volume rendering and analysis of huge collections of 3-D biomedical volume data, where most of the computation was performed by powerful remote visualization engines. Using our enhanced version of Mosaic, later dubbed "WebRouser," a scientist using a low-end workstation could exploit computational power far beyond anything that could be found in one location.

This work was shown to several groups in 1993, including many that were later involved in projects to add APIs and applets to Web browsers at places such as NCSA, Netscape, and Sun. Realizing our group's work enabled the transformation of the Web into a robust platform for the development and deployment of any type of interactive application, in 1994 the University of California filed a U.S. patent application covering embedded program objects in distributed hypermedia documents. Eolas Technologies was then founded by the inventors to promote widespread commercialization and further development of the technology.

Enhancing the Web

Once the concept of the Web as an environment for interactive applications was initiated, the question was how to further develop it. Toward the end of 1993, we discussed the relative merits of building an interpreted language, such as Basic or Tcl, directly into the browser versus enhancing browsers through a "plug-in" API. We chose the API approach, believing that the best way to add language support would be by adding interpreters as external inline plug-in applets, which we called "Weblet applications." This would enable us to add a new language or other feature merely by developing a new Weblet application, without having to reengineer the browser itself.

Figure 1 is a Weblet-based version of the public-domain RAS-MOL visualization program that lets users view, analyze, and visualize a 3-D protein structure from within the Web page. A single programmer converted the original RASMOL source code into Weblet form in only ten hours.

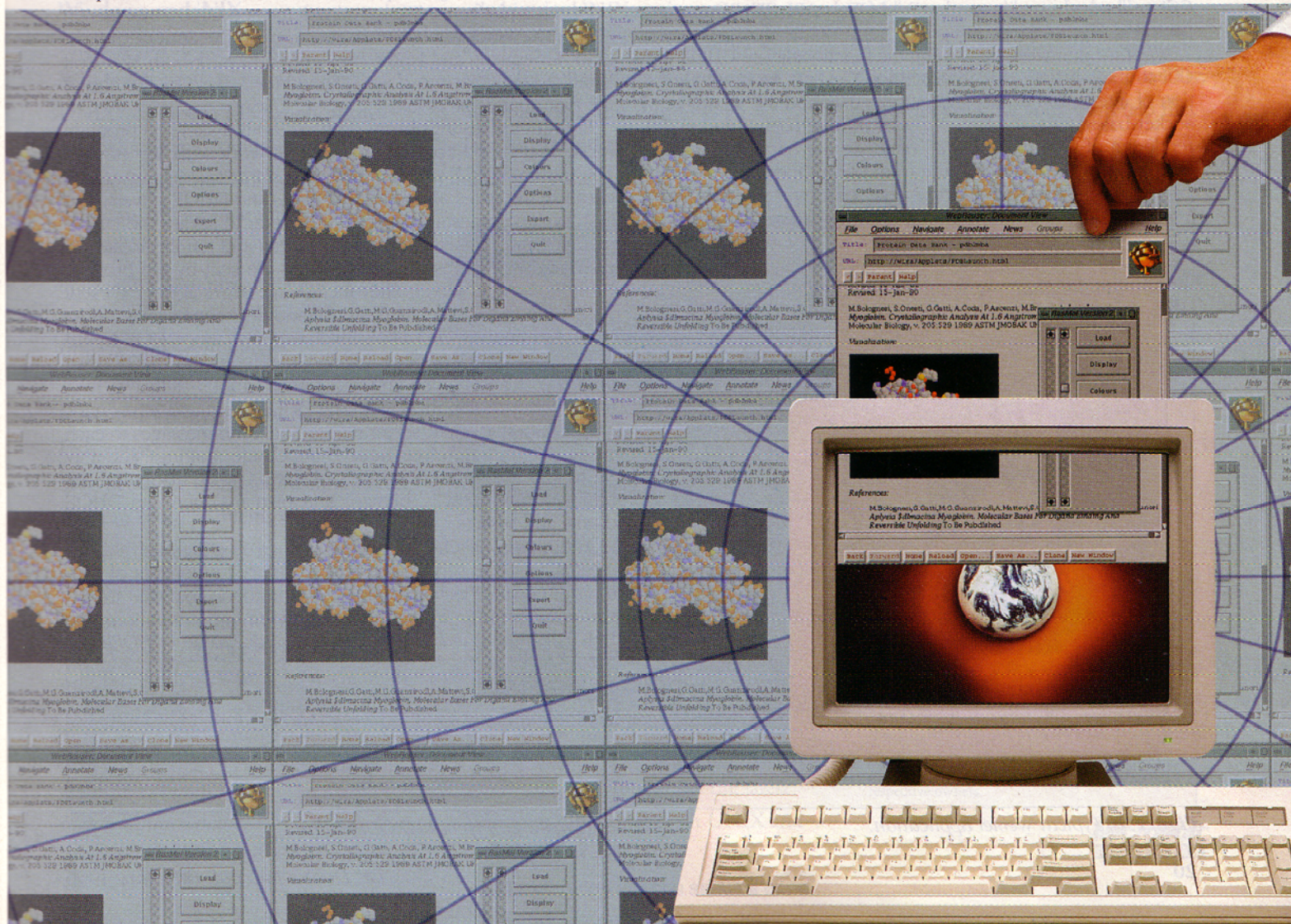
Some time later, both limited API support, such as NCSA's CCI, and embedded-language support, such as Java, began to appear in various Web browsers; the <EMBED...> tag (which we first implemented in mid-1993) appeared in beta versions of Netscape's product by summer of 1995. Still, it wasn't until October of 1995 that the Netscape implementation began to approach the functionality of <EMBED...> used in WebRouser. The enormous effect of these developments in accelerating the commercialization of the Internet industry prompted us to release the first (free-for-noncommercial-use) distribution version of WebRouser for UNIX platforms in September 1995 (<http://www.eolas.com/eolas/webrouse/>).

The WebRouser Approach

Our general philosophy with WebRouser was to allow enhancement of the browser's functionality through object-oriented, modular application components that conform to a standard API, rather than turning the browser into a monolithic application with an ever-increasing code base. This encourages Web developers to take a document-centric approach to application development. The Web page itself becomes the mechanism for doing work, through collections of small, efficient Weblet building blocks, rather than the menagerie of top-heavy applications found on the common desktop PC.

The first release of WebRouser also included other enhancements aimed primarily at improving the interactivity of Web pages. These included client-side image maps, a document-driven button bar, and document-driven modification of the browser's menu tree.

Client-side image maps are supported through the Polymap format. Polymap files are essentially GIF files with polygon and URL information stored in the image's comment fields. To prevent complex polygon information from bloating the file, all of the comment fields are compressed and decompressed using the GIF LZW algorithm. Polymap files require no special treatment of the HTML code. WebRouser autodetects the presence of Polymap data when it reads inline GIFs. If a server-side (ISMAP) image map points to a Polymap GIF, then WebRouser will ignore the ISMAP data and give the Polymap data priority. Hotspots are decoded in real time and highlighted as the mouse moves over the image, and the associated URL is displayed at the bottom of the screen, providing users the same style of interactivity that hotwords have in HTML text. The Polymap for-



mat specification is open and freely available for use. You can find the spec at <http://www.eolas.com/papers/Papers/Polymap/>.

The <LINK...> and <GROUP...> tags allow Web pages to dynamically customize elements of the browser's GUI. The LINK tag allows the creation of a document-driven button bar implemented by placing tags in the document header, with the syntax <LINK ROLE="button label" HREF="http://...">. Several of these tags in sequence result in buttons below the URL window, similar to Navigator's What's New or What's Cool buttons, but they are dynamically defined by the page currently being viewed. Similarly, the GROUP tag allows the Web page to modify the browser's GUI; however, this tag differs by defining a hierarchical menu that reflects an entire tree of Web pages. In Example 1, a typical GROUP menu trigger, the text string "Click here to view the WebRouser slide show" appears as a conventional anchor on the Web page, but selecting it brings up the "slide_1.html" and activates the GROUPS menu option on WebRouser's menu bar. Slide Show is the first menu option, with a submenu whose options are Slide 1, Slide 2, and Slide 3. This allows the user to easily navigate through, for example, the "year, issue, article" hierarchy of online magazines.

The Web API

Of course, the key feature of WebRouser is the implementation of the <EMBED...> tag, through which inline plug-in Weblet applications are supported in Web pages. X Window applications that conform to the Eolas distributed hypermedia object embedding (DHOE) protocol can run—inline and fully interactive—within Web pages in the WebRouser window. WebRouser also supports the NCSA common client interface (CCI), which allows the Weblet to "drive" the browser application. DHOE and CCI collectively make up the Eolas Web API (WAPI) as supported in WebRouser.

WAPI is minimalist, combining the functionality of DHOE and CCI to exploit both the efficiency of X-events for communication of interaction events and graphic data and the flexibility of socket-based messaging for browser remote control and HTML rendering of Weblet-generated data. We are currently working on a cross-platform API, in the form of an OpenGL-style com-

mon-function library. The current minimalist WAPI specification will allow us greater flexibility in creating a cross-platform API, while maintaining compatibility with Weblets developed under the UNIX WAPI specification.

Eolas' primary objective with respect to the pending Web-applet patent is to facilitate the adoption of a standard API for interactive, Web-based application development, and then to develop innovative Weblet-based applications for the growing Internet software market. For an example of such a Weblet application, see the accompanying text box entitled "WebWish: Our Wish is Your Command." We intend to short circuit the API wars brewing between the major Web-browser competitors. In addition to creating a universal standard API, we are also instituting a mechanism for ensuring continued evolution of the WAPI spec on a regular timetable. Royalty-free licenses for browser-side implementation of Web applets under the pending patent have been offered to the major browser companies, and are in various degrees of negotiation. The primary condition of these licenses is that each licensee must conform to the WAPI protocol, and no other applet-integration protocol. A consortium of Eolas licensees is being formed to set the continuing WAPI specification and update it at regular intervals. The widespread acceptance of the developing WAPI standard will allow application developers to concentrate on the functionality of their applets without worrying which Web browser their customers will use.

Creating a WebRouser Weblet

WebRouser communicates with Weblet applications through a set of messages called the DHOE protocol. DHOE messages are relatively short character strings, which allow convenient, efficient use of existing interprocess-communications mechanisms on various platforms. We have implemented DHOE systems on several X Window platforms, including IRIX, SunOS, Solaris, OS/F 1, Sequent, and Linux. Implementations for both Microsoft Windows and Macintosh are planned for release by the end of the first quarter of 1996.

Listing One (listings begin on page 91) is a skeleton program for Weblet-based applications that can work with WebRouser. The current DHOE protocol defines a set of messages that synchronize the states on the DHOE clients and DHOE servers. The first four messages are used by the server to set up the DHOE system at startup, refresh/resize the client, and terminate the system on exit. The rest of the messages are sent by the browser client to the data server. They include messages about the client drawing-area visibility, and mouse and keyboard events.

Programming with DHOE involves initializing DHOE by installing a message-handling function, registering the DHOE client with the DHOE server, and registering various callbacks with their corresponding messages. The DHOE client and server may, at any time after client/server registration, send messages to each other. The messages (see Table 1) are character strings, and may be followed by different types of data. DHOE also supports buffer sharing (that is, bitmaps and pixmaps) between DHOE clients and servers.

Adding the DHOE mechanism into an existing data handler creates a DHOE server. The DHOE library kit consists of protocol_lib.h (the declaration file) and protocol_lib.c (the implementation file). To follow the Xt programming conventions, the DHOE strings are #defined with their Xt equivalents (DHOEkeyUp

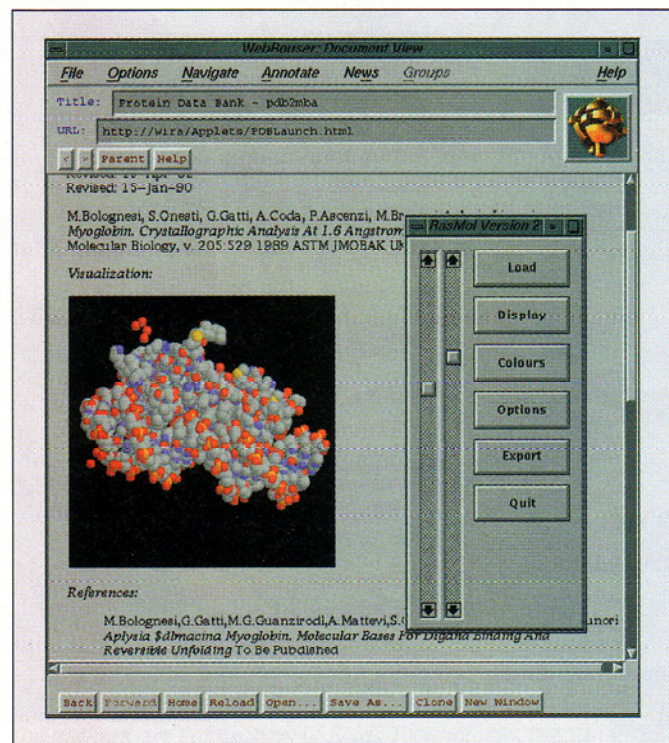


Figure 1: Typical Weblet application.

```
<GROUP ROLE="Slide Show">
  <LINK ROLE="Slide 1" HREF="slide_1.html">
  <LINK ROLE="Slide 2" HREF="slide_2.html">
  <LINK ROLE="Slide 3" HREF="slide_3.html">
  Click here to view the WebRouser slide show </GROUP>
```

Example 1: Typical GROUP menu.

(continued from page 20)

is mapped to *XtNkeyUp*, and so on). Messages from the DHOE server to the DHOE client (for example, external app→hypermedia browser) are:

- *XtNrefreshNotify*, server updating.
- *XtNpanelStartNotify*, server ready.
- *XtNpanelExitNotify*, server exiting.

Message	Description
<i>DHOEserverUpdate</i>	Tells a client to update data.
<i>DHOEserverReady</i>	Tells a client the server is ready.
<i>DHOEserverExit</i>	Tells a client the server is exiting.
<i>DHOEserverConfigureWin</i>	Tells a client to resize/reposition the DHOE window.
<i>DHOEclientAreaShown</i>	Tells the server the DHOE area is exposed.
<i>DHOEclientAreaHidden</i>	Tells the server the DHOE area is being hidden.
<i>DHOEclientAreaDestroy</i>	Tells the server the DHOE area is being destroyed.
<i>DHOEbuttonDown</i>	Sends mouse-pointer coordinates to the server on button down.
<i>DHOEbuttonUp</i>	Sends mouse-pointer coordinates to the server on button up.
<i>DHOEbuttonMove</i>	Sends mouse-pointer coordinates to the server on button move.
<i>DHOEkeyDown</i>	Sends the corresponding <i>keySYM</i> to the server on key down.
<i>DHOEkeyUp</i>	Sends the corresponding <i>keySYM</i> to the server on key up.

Table 1: DHOE messages.

Messages from the DHOE client to the DHOE server (for example, hypermedia browser→external app) are:

- *XtNmapNotify*, DHOE area shown.
- *XtNunmapNotify*, DHOE area hidden.
- *XtNexitNotify*, DHOE area destroyed.
- *XtNbuttonDown*, DHOE area button down.
- *XtNbuttonUp*, DHOE area button up.
- *XtNbuttonMove*, DHOE area button move.
- *XtNkeyDown*, DHOE area key down.
- *XtNkeyUp*, DHOE area key up.


You can name these messages differently as long as the names are merely aliases of the original DHOE strings. These messages are defined in *protocol_lib.h*, which must be included in your program.

The following DHOE fundamental functions are provided in *protocol_lib.c*:

- *void handle_client_msg(Widget w, caddr_t client_data, XEvent *event)*, a function called back by *XtAddEventHandler* when it sees a message from the DHOE client (the hypermedia browser). To register this function with Xt, your program (DHOE server) should call *XtAddEventHandler(Widget app_shell, NoEventMask, True, handle_client_msg, 102)*. Here, *handle_client_msg* will be called with parameters *w=app_shell*, *client_data=102*, an *event* pointing to an X-event structure generated by Xt when it sees the message. The *app_shell* variable is usually the application shell returned by *XtInitialize*, *XtAppInitialize*, or *XtVaAppInitialize*.

neoAccess™

Cross-Platform Object Database Engine



Now Available!
Version 4.0

neo•logic
Now!
<http://www.neologic.com/~neologic/>
 Download the Architectural Overview!


Power Too Abundant to Meter

Powerful
NeoAccess is the most powerful object-oriented database engine available. It displays electrifying performance—up to ten times that of its competitors. Behind its elegant programming interface is a high performance query engine utilizing: extended binary trees and binary search algorithms tuned for short access times, dynamically combined, collapsed, and compressed indices, and object caching for nearly instantaneous access to previously used objects.

No Runtime Fees
Get the power of NeoAccess, and avoid the expense and administrative hassle of feeding the runtime fees meter. You pay one affordable price no matter how many copies of your application you sell or use.

Cross Platform
Others may promise cross-platform development tools—NeoLogic delivers. NeoAccess is a set of C++ classes designed for use with popular compilers and application frameworks on Windows®, Macintosh®, and Unix™ platforms. Full source code is included so it can even be used with custom frameworks.

Proven
Thousands of commercial and in-house developers have already found that NeoAccess enabled them to build fast, powerful applications in record time. That's why NeoAccess based applications are already operating on millions of computers. Tap into the power!



Powering Development of Object-Oriented Applications

NeoLogic Systems 1450 Fourth St., Suite 12 v. 510.524.5897
 neologic@neologic.com Berkeley, CA 94710 f. 510.524.4501

(continued from page 22)

- `void register_client(Widget w, Display *remote_display);`, which registers your program with the DHOE client.
- `void register_client_msg_callback(char *msg, void (*function_ptr)());`, which registers a function to be called back when Xt sees a string that matches `msg`. This function may appear anywhere in your program. You do not need to handle the `XtNmapNotify/XtunmapNotify` pair because DHOE servers deiconify/iconify when they receive these messages. You must specify a "quit" function to shut down your application gracefully on `XtNexitNotify`. Button- and key-message handling are optional. To obtain mouse coordinates, call `get_mouse(int *x, int *y)` for button-handling functions and `get_keysym(KeySym *keysym)` for key-handling functions. `Keysym` is defined by X11 (in `keysymdef.h`) for cross-platform compatibility.
- `void send_client_msg(char *msg, Display *remote_display, Window remote_window);`, which sends a message with the value `msg` to the DHOE client at a `display=remote_display` and has an X window ID of `remote_window`. The `remote_display` and `remote_window` must be provided. This function may appear anywhere in the program after `register_client`.

A Weblet CAD-File Viewer

WT is an applet that allows interactive rotation and zooming of a 3-D CAD file stored in NASA's neutral file format (NFF). The source code for the sample Weblet application is available electronically (see "Availability," page 3) and at <http://www.eolas.com/eolas/webrouse/wtsrc.tar.Z>. What follows is a brief walk-through of the weblet-enhancing sections of the code (illustrated in the code listing just mentioned as a "simplified sample program outline").

1. The outline starts with a `typedef` and some global declarations. The new type, `ApplicationData`, defines a structure common to all Xt Weblets. Together with the `myResources` and `myOptions` static variables, `myAppData` (which is of type `ApplicationData`) is used with `XtGetApplicationResources` in `main()` to extract the command-line arguments flagged with `win`, `pixmap`, `pixmap_width`, `pixmap_height`, and `datafile`. This is how Xt extracts command-line arguments and is unnecessary if the program has alternatives to decode command-line arguments. The aforementioned global variables and `XtGetApplicationResources` nicely store the information in a line such as `wt -win 1234 -pixmap 5678 -pixmap_width 400 -pixmap_height 300 -datafile fname` into `myAppData`.

2. In `main()`, `app_shell` is first initialized the Xt way by using `XtInitialize`, which opens a connection to the X server and creates a top-level widget. `XtGetApplicationResources` gets the application resources as in step 1. The next section conveniently uses the `myAppData.win` variable to find out if the Weblet should run as a DHOE server or a stand-alone program. For a DHOE server, the program adds the `handle_client_msg` function from the DHOE implementation, `protocol_lib.c`, as the handler of the X client message event. The subsequent lines call three more DHOE functions: `register_client`, to initiate a handshake with the DHOE client; `register_client_msg_callback`, to register `myQuit()` as the callback function of the message `XtNexitNotify`; and `send_client_msg`, to send a `XtNpanelStartNotify` message, telling the DHOE client that the server is ready. The program then enters the conventional `XtMainLoop()`.

3. Two more functions must be modified. The drawing routine (`myDraw`) needs to copy the drawn picture (`myPixmap` in this case) onto `myAppData.pixmap`, the client's `pixmap`. The function then should send an `XtNrefreshNotify` message to the client, informing it of the change. The `myQuit()` function registered in

WebWish: Our Wish is Your Command

Sun's announcement of the adaptation of the Java language to the Web in 1995 was received enthusiastically by the entire Internet community as a welcome means for increasing the interactivity of Web-based content. Despite much of the publicity surrounding Java, which described it as an "interpreted" language, Java code must be compiled to a "virtual machine," which is then emulated on various platforms. A Web browser that supports the Java emulator is not enough to develop Java-based applications—the applet developer must purchase a compiler from Sun or its licensees at considerable cost.

Fully interpreted languages like Tcl/Tk or Basic are extremely useful, partly because they don't require a compiler for application development, just the language interpreter and any ASCII text editor. In choosing a programming language to adapt to the Web API, we decided early on that a fully interpreted programming language would be vital to quick, widespread Weblet implementation. We chose Tcl/Tk because of its robust capabilities and widespread use.

By the time you read this article, Eolas' WebWish Tcl/Tk interpreter should be available for both WebRouser under UNIX and Netscape Navigator 2.0 on Windows and Macintosh (see <http://www.eolas.com/eolas/webrouse/tcl.htm>). It supports Tcl 7.5 and Tk 4.1, as well as the Tcl-DP and EXPECT extensions. A new security feature has been added that exploits PGP-style digital signatures in order to authenticate scripts from trusted sources and to prevent unwanted execution of scripts from untrusted sources. This Weblet application turns WebRouser and Navigator into complete application-development

environments, without the need for expensive compilers. All that is needed to develop a WebWish-based application is WebWish, a WAPI-compliant Web browser, and a good text editor. Developers can draw upon the vast existing resources of freely downloadable Tcl/Tk program source code, and the expertise of thousands of experienced programmers.

WebWish provides an easy-to-use rapid prototyping environment, with built-in support for socket-based communications, remote procedure calls (RPCs), and the ability to "remote control" existing text-based server systems without reengineering the server. WebWish can run either as a Weblet in a Web page, or in stand-alone mode on either the client or a server machine. WebWish running in a Web page can communicate directly with other copies of WebWish running on remote servers, either through sockets or RPCs. This allows WebWish to act as "middleware" for the Web, allowing Web-based interfaces to create state-aware graphical front ends to existing text-based legacy systems, without changing the operation of the legacy-server application.

Last November, Chicago's Rush Presbyterian St. Luke's Medical Center surgical department created both client and server WebWish applets for just such a purpose. The applets allowed physicians using WebRouser to interactively query and browse Rush's (Informix) SQL-based Surgical Information System, consisting of medical records on over 1.5 million patients. The entire project took one programmer exactly 12 hours from start to finish. Try that with Java!

—M.D., C.A., and D.M.

(continued from page 24)

`main()` needs to send an `XtNpanelExitNotify` message to the client, telling the client that the server is terminated.

This Weblet can be tested by putting it in your path and pointing your copy of WebRouser to <http://www.eolas.com/eolas/webrouse/office.htm>.

The Eolas Web OS

In addition to the WebWish applet described in the text box, a Java interpreter Weblet application is planned for release by the end of March 1996. Java is a compiled language that produces binaries for a "virtual machine." The binaries are downloaded to the client and run on virtual-machine emulators that run on Macintosh, Windows, and UNIX platforms. Java applications tend to be smaller and more efficient than WebWish interpreted code, but they are far more difficult to develop. Eolas is developing a virtual operating system, the Web OS (planned for release late in 1996) that will allow far more robust, compact, and efficient compiled applets to be developed than is possible with Java. The Web OS is key to Eolas' long-term goal to transform the Web into a robust, document-centric, distributed-component application environment. It is a real-time, preemptive multitasking, multithreaded, object-oriented operating system that will run efficiently on low-end platforms, even on 80286-based systems and handheld PDAs.

The Web OS can run within Windows, Macintosh, and UNIX environments, or in stand-alone mode on machines with no pre-installed operating system. It supports dynamic memory management and linked libraries, and is both graphical and object oriented at the OS level. The OS kernel includes fully defined object classes, inheritance, and direct messaging. The OS includes sev-

eral building-block objects that allow sophisticated applications—WYSIWYG word processors, spreadsheets, databases, e-mail systems, and the like—to be developed with a minimum of code. These applications are created primarily by subclassing and combining various Web OS component objects. Since new applications are created by defining differences and additions to the constituent objects, this results in tiny, robust, efficient binaries that optimize both bandwidth usage and server storage requirements. This platform is so efficient that a complete WYSIWYG word processor can be created in less than 5K of compiled code. Applications developed for the Web OS are likely to be smaller than most of the inline GIF images found on average Web pages today.

The operating system employs a single imaging model for screen, printer, fax, and other output devices; an installable file system, for both local and remote file access; direct TCP/IP and socket support; distributed objects; and security through public-key encryption and "ticket-based" authentication.

As the Internet pervades more of our work environments, the Web OS will allow the Web to become the preferred environment for new and innovative productivity, communications, and entertainment applications for all hardware platforms. The concept of a machine-specific operating system will become irrelevant, since any application will be available to the average user, regardless of hardware platform. Much of the computational load for applications will be pushed off to remotely networked computational engines, allowing low-cost Web terminals to act as ubiquitous doorways to potentially unlimited computational resources. The Web will be your operating system and the Internet will be your computer.

DDJ

(Listing begins on page 91.)

IF YOU KNOW OS/2® YOU SHOULD KNOW INDELIBLE BLUE

Post Road Mailer by Innoval Systems

Make sending and receiving email easier! Built-in editor, quoted replies, MIME attachments, drag-and-drop filing, printing and shredding; put the world at your fingertips.

INO20 PRM (Green) Internet Edition
MSRP \$59.00 **\$49.00**

INO21 PRM Blue (PROFS)
MSRP \$79.00 **\$64.00**

Kopy Kat by Hilgraeve

Anything you can do on a PC with OS/2, you can do remotely by modem, serial cable or LAN. See the full desktop of the remote PC on your screen, control mouse and keyboard.

HGR48 KopyKat 2-user pack
MSRP \$199.00 **\$125.00**

HGR44 KopyKat 10-user pack
MSRP \$795.00 **\$495.00**

Your Single Source for OS/2 Solutions is also Your Single Source for Internet Solutions!

OS/2 WarpConnect IBM's Internet Connection Server

AntiVirus Hyperwise

DB2 World Wide Web Connection

Call, or check our web pages for all the info!

GammaTech Utilities by SoftTouch Systems

Buy them before you need them! Recover deleted files, files from a corrupted volume or partition, optimize HPFS and FAT partitions, mass deletions, edit sectors and much more.

GTU20 GammaTech Utilities
MSRP \$149.00 **\$99.00**

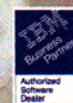
Site Licensing Available

Corporate Discounts: Volume Incentive Program

Save up to 45% on IBM software for your business! Call 1-800-901-VALU for details.

Visit our web site:

<http://www.indelible-blue.com/ib>



**THE SINGLE SOURCE
FOR OS/2 SOLUTIONS**

**CALL FOR NEW 64 PAGE
FULL COLOR CATALOG!**

1-800-776-8284

OS/2 and IBM are registered trademarks of IBM Corporation. All other trademarks belong to their respective owners. Prices are subject to change. This ad composed on a PC running on OS/2 Warp, using native OS/2, DOS and Windows apps.

Listing One

```

#include "protocol_lib.h"
...
/* X-way to define resources and parse the cmdline args */
/* WebRouser 2.6-b2 gives the embedded window information through these args */
typedef struct
int win;
int pixmap;
int pixmap_width;
int pixmap_height;
char *datafile;
} ApplicationData, *ApplicationDataPtr;
static XtResource myResources[] = {
{"win", "Win", XtRInt, sizeof(int),
XtOffset(ApplicationDataPtr, win), XtRImmediate, 0},
{"pixmap", "Pixmap", XtRInt, sizeof(int),
XtOffset(ApplicationDataPtr, pixmap), XtRImmediate, 0},
{"pixmap_width", "Pixmap_width", XtRInt, sizeof(int),
XtOffset(ApplicationDataPtr, pixmap_width), XtRImmediate, 400},
{"pixmap_height", "Pixmap_height", XtRInt, sizeof(int),
XtOffset(ApplicationDataPtr, pixmap_height), XtRImmediate, 400},
{"datafile", "Datafile", XtRString, sizeof(char*),
XtOffset(ApplicationDataPtr, datafile), XtRImmediate, NULL},
};
static XrmOptionDescRec myOptions[] = {
{"-win", "*win", XrmoptionSepArg, 0},
{"-pixmap", "*pixmap", XrmoptionSepArg, 0},
{"-pixmap_width", "*pixmap_width", XrmoptionSepArg, 0},
{"-pixmap_height", "*pixmap_height", XrmoptionSepArg, 0},
{"-datafile", "*datafile", XrmoptionSepArg, NULL},
};
ApplicationData myAppData;

void myDraw()
{
/* do your drawing... */
...
/* if you draw into your own drawables (myPixmap in this case) */
if (myAppData.win) {
/* copy from myPixmap to the "shared" pixmap */
XCopyArea(display, myPixmap, myAppData.pixmap, myGC, 0, 0, WIN_WIDTH,
WIN_HEIGHT, 0, 0);
/* tell WebRouser to update the drawing window */
send_client_msg(XtNrefreshNotify, display, myAppData.win);
}
}
void myQuit()
{
/* tell WebRouser you are exiting... */
if (myAppData.win)
send_client_msg(XtNpanelExitNotify, display, myAppData.win);
/* Motif way of exiting */
XtCloseDisplay(XtDisplay(any widget));
exit(1);
}
...
main()
{
Widget app_shell;
...
/* XtInitialize does XOpenDisplay, as well as creates a toplevel widget */
app_shell = XtInitialize("wt", "Wt", myOptions, XtNumber(myOptions),
&argc, argv);
...
/* This func fill up myAppData with user specified values/default values */
/* We get the embedded window's info this way */
XtGetApplicationResources(app_shell, &myAppData, myResources,
XtNumber(myResources), NULL, 0);
...
/* if we have an external window to display the image... */
if (myAppData.win) {
XtAddEventHandler(app_shell, NoEventMask, True, handle_client_msg, NULL);
register_client(app_shell, display);
/* register the func to be called when WebRouser exits */
register_client_msg_callback(XtNextNotify, myQuit);
/* tell WebRouser you have started fine */
send_client_msg(XtNpanelStartNotify, display, myAppData.win);
}
...
XtMainLoop(); /* Motif's event loop */
}
/* End of program listing */

```

KERMIT

Listing One

```

STATIC int KSendPacketFromCache(KERMIT_PRIVATE *pK, long sequence, int addToList)
{
int slot = sequence & 63;
long prev, next;

if ((pK->exchange[slot].myPacket.type == 0)
|| (pK->exchange[slot].myPacket.data == NULL))

```

```

return kOK;
prev = pK->exchange[slot].previousPacket; /* Unlink from list */
next = pK->exchange[slot].nextPacket;
if ((pK->lastPacket & 63) == slot) pK->lastPacket = prev;
if (prev >= 0) pK->exchange[prev & 63].nextPacket = next;
if (next >= 0) pK->exchange[next & 63].previousPacket = prev;
pK->exchange[slot].nextPacket = -1;
pK->exchange[slot].previousPacket = addToList ? pK->lastPacket : -1;
if (addToList) {
if (pK->lastPacket >= 0) /* Add to end of list */
pK->exchange[pK->lastPacket & 63].nextPacket =
pK->exchange[slot].sequence;
pK->lastPacket = pK->exchange[slot].sequence;
}
pK->exchange[slot].tries++; /* Count number of sends */
pK->exchange[slot].sendTime = SerialTime(pK->initTime);
/* Stamp time of send */
return StsWarn (KSendPacket (pK, slot, pK->exchange[slot].myPacket.type,
/* Send it */
pK->exchange[slot].myPacket.data,
pK->exchange[slot].myPacket.length));
}
STATIC int KSendPacketReliable(KERMIT_PRIVATE *pK, BYTE type,
const BYTE *pSendData, unsigned long sendDataLength,
unsigned long rawDataLength)
{
int blocked = FALSE;
int err;
int slot = pK->sequence & 63;
int timeout = pK->my.timeout;
{ /* Put packet into cache */
EXCHANGE *pThisExchange = &(pK->exchange[slot]);
if (pThisExchange->myPacket.data == NULL) {
if (pK->minCache < pK->minUsed) {
SwapSlots (pK->minCache, slot);
/* Move free exchange to end of window */
pK->minCache++;
pThisExchange->yourPacket.type = 0;
} else return StsWarn (kFail); /* Internal consistency failure */
}
if (pSendData == pK->spareExchange.myPacket.data) {
/* In the reserved slot? */
BYTE *pTmp = pThisExchange->myPacket.data; /* Just swap it in */
pThisExchange->myPacket.data = pK->spareExchange.myPacket.data;
pK->spareExchange.myPacket.data = pTmp;
} else /* copy it */
memcpy (pThisExchange->myPacket.data, pSendData, sendDataLength);
if (pK->sequence > pK->maxUsed) pK->maxUsed = pK->sequence;
/* Update end of window */
pThisExchange->sequence = pK->sequence;
/* Finish initializing this exchange */
pThisExchange->myPacket.length = sendDataLength;
pThisExchange->myPacket.type = type;
pThisExchange->rawLength = rawDataLength;
pThisExchange->tries = 0;
pK->txPacket.data = pK->spareExchange.myPacket.data;
pK->txPacket.length = 0;
}
StsRet (KSendPacketFromCache (pK, pK->sequence, TRUE)); /* Send packet */
if (pK->minUsed <= pK->minCache) blocked = 1; /* Are we blocked? */
if (pK->maxUsed - pK->minUsed + 1 >= pK->currentWindowSize)
/* How blocked are we? */
blocked = (pK->maxUsed - pK->minUsed + 1) - pK->currentWindowSize + 1;
err = KReceivePacketCache (pK, 0); /* Get a packet if one's ready */
do { /* Until we're not blocked and there are no more packets pending */
switch (err) {
case kBadPacket: /* Didn't get a packet */
case kTimeout:
break;
default: /* Unrecognized error, pass up to caller */
return StsWarn (err);
case kOK: /* Got one! */
{
EXCHANGE *pThisExchange = &(pK->exchange[pK->rxPacketSequence & 63]);
switch (pK->rxPacket.type) {
case 'N': /* Got a NAK */
if (pThisExchange->myPacket.type != 0) /* Resend packet */
StsRet (KSendPacketFromCache (pK, pK->rxPacketSequence, FALSE));
if ((pK->currentWindowSize > 1) || (pK->maxUsed > pK->minUsed))
break; /* Don't generate implicit ACKs for large windows */
pThisExchange = &(pK->exchange[(pK->rxPacketSequence - 1) & 63]);
pThisExchange->yourPacket.type = 'Y';
case 'Y': /* Got an ACK */
if (pThisExchange->rawLength > 0) { /* ACKed before? */
if (pThisExchange->tries == 1) { /* Update round-trip stats */
long now = SerialTime (pK->initTime);
long thisDelay = now - pThisExchange->sendTime;
if (pK->roundTripSamples++ == 0) { /* First sample? */
pK->roundTripDelay = thisDelay;
pK->roundTripDelayVariance = 0;
} else {
long oldAverage = pK->roundTripDelay;
long diffSquared;
if (pK->roundTripSamples > 30) /* Average first 30 */
pK->roundTripSamples = 30;
/* Then decaying average */
pK->roundTripDelay += (thisDelay - pK->roundTripDelay)
/ pK->roundTripSamples;
diffSquared = (thisDelay - oldAverage) *
(thisDelay - oldAverage);
pK->roundTripDelayVariance += (diffSquared -
pK->roundTripDelayVariance) /
pK->roundTripSamples;
pK->roundTripDelaySD = (pK->roundTripDelaySD +
pK->roundTripDelayVariance /
pK->roundTripSamples) / 2;

```

(continued on page 92)