

RomPager Advanced

*Web Server Engine and Web Application Toolkit
for Embedded Network Devices*



RomPager[®] Advanced is an **affordable, royalty-free** Web server engine and Web application toolkit designed to add Web-based management to any embedded network device. Adding RomPager to a device allows it to be configured and managed using standard Web browsers such as Netscape Navigator or Microsoft Internet Explorer.

RomPager is the **most widely-used** OEM embedded Web server software on the market today because of its features, flexibility, and performance. It effectively integrates with major operating systems and processors, supports all current and emerging **Internet standards**, allows Web page content to be easily added to devices, and enables the developer to balance performance needs and memory budgets.

Since Web browsers run on Windows (3x, 95, NT), DOS, OS/2, Unix and Macintosh clients, an embedded Web server means universal client support. This eliminates custom management application development, or the requirement for expensive SNMP management consoles.

In addition to enabling **Web-based management** of embedded devices, RomPager can improve customer service. Web pages located on the network device can point back to your Web site, improving customer support, providing current marketing information and enhancing the customer relationship.

Using RomPager for Web-based management means that accessing and controlling routers, gateways, switches, hubs, switches, print servers, RAID disk servers, controllers, Universal Power Supplies and other network devices is as easy as browsing the Net. Additionally, Allegro offers RomMailer[™] with **email notification** capabilities so that the same devices can be proactively managed using email "push".

Using the RomPager Web Server Engine and Application Toolkit provides significant advantages:

Your **time to market** will be much faster with less engineering expense than developing the capabilities in house. Since the compatibility testing for all versions of the major browsers has already been done, you get a solid, protocol-compliant implementation of the Web server specifications.

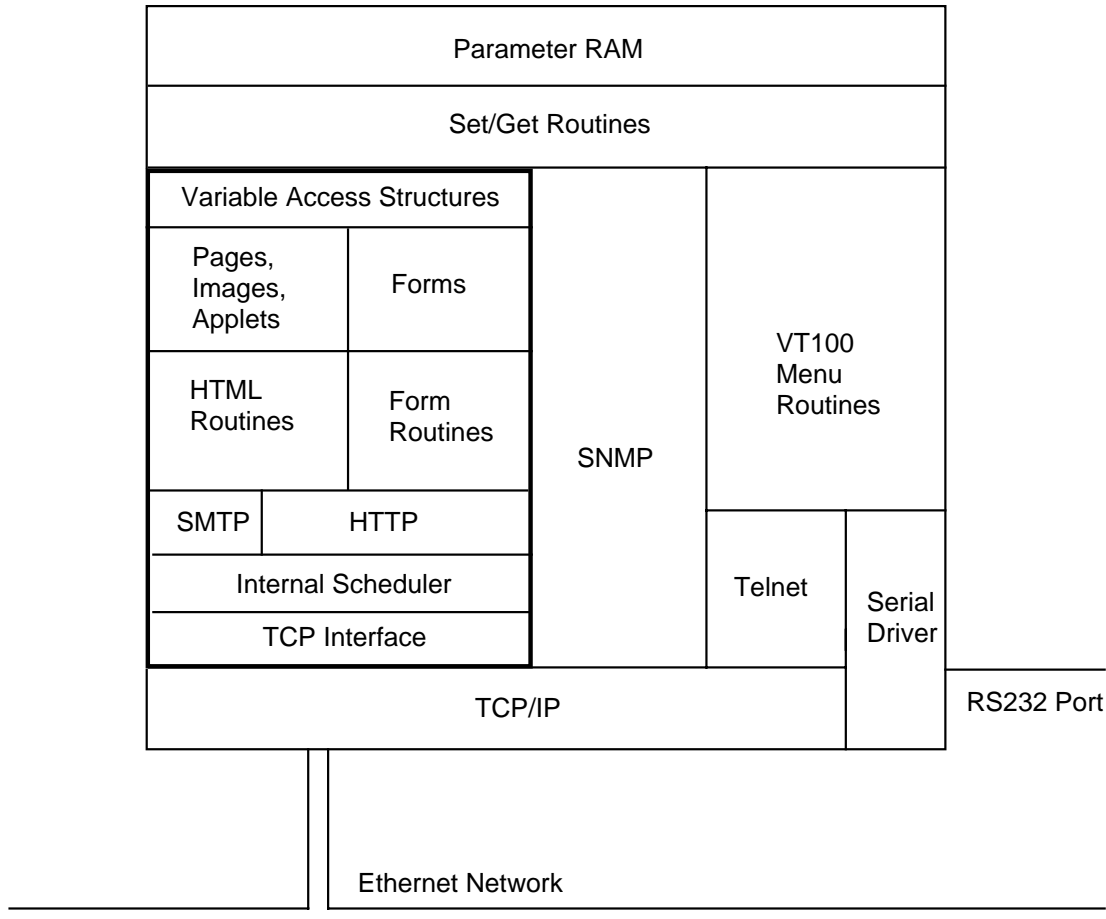
RomPager was designed from the ground up to support devices with ROM-based real-time environments, while the available public domain code was designed for disk-based Unix environments.

Ongoing support for new browser revisions and the new versions of the **evolving HTTP and HTML specifications** is provided by Allegro to save you engineering time.

RomPager has been shipping since early 1996 and is available for your devices today.

RomPager Architecture

RomPager is designed to complement the management architecture of the typical embedded device. An embedded device is normally managed through the network using SNMP and/or Telnet protocols which in turn use Set/Get routines to access the parameters being managed. Some devices also have an “out-of-band” serial port to manage the device parameters. The RomPager software can easily be integrated into existing device management environments as illustrated by the following block diagram. In this diagram, the RomPager components needed to add both Web browser management as well as Email notification capabilities to an embedded device are outlined in bold.



A manufacturer can add Web-based management to their device with RomPager using these steps:

- 1) Port the RomPager Web Server Engine (HTTP, TCP Interface, scheduler) and Web Application Toolkit runtimes (HTML and Form routines) to the embedded device’s operating system. Since the Engine has already been ported to all major RTOS/ CPU combinations, this is a straight-forward task.
- 2) Design the HTML pages and forms for the Web-based management application using standard HTML page design tools. Web page layout tools such as Microsoft’s FrontPage or Adobe’s PageMill do an excellent job of producing HTML code, but even a simple text editor can be used.
- 3) Use the RomPager Web Application Toolkit to integrate the HTML pages and forms with the device parameter access (Set/Get) routines. The Web Application Toolkit includes the PageBuilder compiler, which generates standard ANSI-C source code from the HTML. The C code is then compiled and linked with the rest of the device’s code.

RomPager Features

RomPager is compatible with all standard browsers including Internet Explorer, Netscape Navigator or Communicator, Hot Java and Mosaic, on all platforms, allowing any network-connected user to easily access your device.

Support for HTML levels 2.0, 3.2 and 4.0 including Netscape and Microsoft extensions so that your pages can present information effectively. This includes page and forms processing support for <INPUT>, (Button, Checkbox, Hidden, Form Password, Radio, Reset, Submit, and Text types) <TEXTAREA>, <SELECT>, and Image Map elements.

HTTP 1.1 Support

The HTTP 1.1 support complies with current specifications (RFC 2616) and can provide improved page loading times and network efficiency. RomPager provides full interoperability with Microsoft Internet Explorer 4.0/5.0 HTTP 1.1 implementations. Digest Authentication (RFCs 2069 and 2617) is also supported, although not all browser vendors are supporting this capability.

An interface to SNMP-style set and get routines is provided to maximize use of your existing software investment. Direct access of SNMP variables via MIB Object ID pointers is supported for the SNMP Research EMANATE™ product line.

Minimum Application Memory Footprint through Intelligent Compression

The PageBuilder HTML compiler pre-processes and compresses your Web pages and images into compilable ANSI-C code. This allows pages to be developed using standard HTML tools, and stored internally in a patent-pending memory efficient format. Both standard HTML and user phrases are tokenized to use runtime phrase dictionaries. Common page elements are stored once and shared across multiple pages. Shared and nested pointer techniques are used for additional memory savings. These various techniques yield a smaller Web application footprint than any other vendor.

Dynamic HTML creation includes support for index-based variables, so that tables and other repeating groups of HTML may be created easily in a memory-efficient manner.

Efficient Internationalization Support

Dynamically selectable user phrase dictionaries make for easy support of internationalized pages. Since a page is stored only once with multiple phrase libraries, your application can provide comprehensive international support while using significantly less memory than other approaches. RomPager comes with standard error messages included for French, German, Italian, Portuguese and Spanish as well as English.

Flexible Security

Sensitive pages can be protected as a group (a session protection realm) rather than requiring a password per page. Eight separate realms are supported with optional overlapping, for superset realm creation. Security functions include realm timeout with password re-challenge support. User exit routines are provided for maintenance of the realmname, username and password variables so that these values can be managed from Web pages or other management services.

External Security Support

This feature provides support for security verification using an external verification source such as a RADIUS server rather than the RomPager internal security tables.

Standard HTTP Header Support

RomPager serves objects including HTML pages, standard images, and Java applets with the appropriate MIME types. Caching support is provided for static and dynamic objects using the appropriate HTTP headers. Date header generation is controlled using calendar clock, realtime clock or clockless environments.

Extended HTTP Header Support

Netscape-style dynamic pages (using Client-Pull or Server-Push techniques) are supported for automatically updated pages. Netscape-style persistent connections using the Keep-Alive header are available to reduce connection overhead.

Built-in numeric conversion routines between HTML text format and internal numeric formats to eliminate device-specific conversion routines. The format conversions include support for 8, 16, 32, and 64-bit signed and unsigned integers, ASCII text, Boolean, Hex display, separated Hex display and TCP/IP dotted decimal display.

File System support allows you to serve or receive any arbitrary data stream from RAM, flash, disk or the network. This allows additional HTML pages, graphics or Java applets to be served without using ROM memory resources. File System support also allows uploading and downloading files to and from your device from any browser. Upload support requires an HTML 3.2 compliant browser that supports RFC 1867. The File System support offers a generalized asynchronous I/O interface to any file system environment including support for all latency issues.

The RomPager code has **compilation options for speed/memory/feature tradeoffs** so that the package may be optimized for your application. Engine working memory may be allocated statically or dynamically. Compilation options provide control over buffer sizing, number of simultaneous HTTP requests, index depth, clock support, and security features such as session timeouts and password groups. Additional flags control inclusion/exclusion of image map support, individual HTML form element support, JavaScript support, MIME type checking and a variety of other options.

The RomPager toolkit comes with a rich set of **sample pages** consisting of over 100 HTML pages, images and Java applets. These demonstration pages show a complex application that illustrates the full set of RomPager capabilities. There is also a set of validation pages that each test a single HTML function, and may be used to learn about a particular capability. Since the sample pages use all the facilities of RomPager, they may also be used to test the robustness of the port to your environment.

Internet Printing Protocol (IPP) support is provided. RomPager recognizes and passes IPP packets through HTTP 1.1 to an IPP parser process. The parser process returns IPP responses and RomPager provides the appropriate HTTP 1.1 transport to the IPP client.

URL State Management Support provides support for state passing between pages using state variables in the URL.

Image mapping support is provided for rectangle, circle, and polygon elements.

The **Remote Host Support** feature uses an optional HTTP client to redirect a request from the embedded device to another server on the network to retrieve applets, graphics, and other large objects. This is sometimes called proxy services. This feature allows devices with limited memory to still present large graphics or applets to the user. This is particularly important for applets, since Java security requires that applets can only talk with the device that served the applet.

Email Notifications can be provided with the optional RomMailer™ SMTP mail client. Using SMTP allows an embedded device to send asynchronous messages and notifications to any Internet mail server which will then relay the message to any email reader. A key feature of the mail service is that the message body is a RomPager object, so that a message can contain dynamic information in the same way that a page can contain dynamic information. A message can be plain text, simple HTML, or HTML with embedded images (RFC 2112). The latest email readers from Netscape and Microsoft support this “HTML Mail” feature, as will many of the other mail readers.

Email Message Reading can be provided with the optional RomPOP™ POP3 mail client. Using POP3 allows an embedded device to receive asynchronous messages and notifications from any Internet mail server. A message can be plain text, or text with any kind of attachments. Using RomPOP, a device can receive configuration information, submitted print jobs, or messages from other devices.

RomPager Porting Requirements

Operating System Environments

The RomPager embedded web server will work with any OS/TCP package and includes interface files for most popular RTOS environments on the distribution diskettes. The environments RomPager has been ported to include uCOS, ATI Nucleus, Elmic ELX, Epilogue, Express Logic ThreadX, ISI pSOS, JMI C-EXEC, Lynx, Kadak AMX, Mentor VRTX, Microsoft Windows CE, Microware OS-9, Pacific Softworks Fusion, Perihelion Helios, Precise MQX, QNX, SNMP Research, Treck TCP and WindRiver VxWorks stacks. RomPager has also been ported to Windows, Macintosh, Linux and Unix development environments.

Since RomPager comes with its own internal scheduler, any OS can be used to support the engine. In fact, some of our customers do not use an OS, and just run RomPager off the main device idle loop. For these environments, RomPager may be invoked with code similar to the sample below:

```
serverIsUp = false;
theServerData = RomPagerInit();
if (theServerData != (void *) 0) {
    serverIsUp = true;
}
while (serverIsUp) {
    /*
        Wait here for time or event message,
        and give up time to other threads.
    */
    if (!RomPagerMainTask(theServerData, &theHttpTasks, &theTcpTasks)) {
        RomPagerDeInit(theServerData);
        serverIsUp = false;
    }
}
```

RomPager uses a single processing thread with multiple TCP connections maintained in the TCP layer and multiple overlapping HTTP requests supported by the RomPager engine using internal request control blocks and a lightweight scheduler. The engine uses either a polling or message passing model to interface with the TCP/IP interface and timer services. If a message passing model is used, the thread is passed messages from the TCP interface for packet activity, and timer messages once per second. The HTTP protocol is not very time-sensitive, so the RomPager thread may be run at a low priority.

Compiler Requirements

RomPager is delivered as an integrated set of ANSI-C compliant source code that provides HTTP, HTML, and forms processing engines. The code has been compiled for Intel, Motorola, AMD, ARM, SPARC and MIPS processors in CISC and RISC environments.

Memory Requirements

The RomPager engine and toolkit runtime routines take between 9Kb to 25Kb of ROM code, with 4Kb of ROM data, 1Kb basic engine RAM and 3Kb to 5Kb of RAM per simultaneous HTTP request. Actual requirements depend on which compilation options and engine features are chosen.

The RomPager application toolkit uses a number of techniques to save page memory including dynamic HTML generation and text pointer sharing. As a result, the internal stored format of pages is about 60% of the external HTML page size. A typical application's pages and set/get routines will take between 25Kb and 35Kb of memory, depending on complexity and image use. Thus, a complete Web-based management application can be implemented in 50Kb of ROM and 10Kb of RAM.

TCP/IP Interface Requirements

RomPager will work with any TCP/IP package and has been ported to all the popular TCP environments. Interface files for these environments are included on the distribution disks. For other TCP environments, RomPager includes a simple interface shell to the TCP services which allows the porter to keep all the specific interface routines in a separately compiled module. The engine passes a connection index to the interface routines as an abstraction of the per connection data. The prototypes of the commands the porter needs to implement for the other services are shown below:

```
extern RpErrorCode StcpInit(void);
extern RpErrorCode StcpDeInit(void);
extern RpErrorCode StcpOpenPassive(StcpConnection theConnection);
extern RpErrorCode StcpConnectionStatus(StcpConnection theConnection,
                                        StcpStatus *theConnectStatus,
                                        StcpIpAddress *theRemoteAddressPtr,
                                        StcpIpAddress *theLocalAddressPtr,
                                        StcpPort *theLocalPortPtr);
extern RpErrorCode StcpOpenActive(StcpConnection theConnection,
                                  StcpIpAddress theRemoteAddress,
                                  StcpPort theRemotePort);
extern RpErrorCode StcpActiveConnectionStatus(StcpConnection theConnection,
                                              StcpStatus *theCompletionStatusPtr,
                                              StcpIpAddress *theLocalAddressPtr);
extern RpErrorCode StcpReceive(StcpConnection theConnection);
extern RpErrorCode StcpReceiveStatus(StcpConnection theConnection,
                                     StcpStatus *theReceiveStatus,
                                     char **theReceivePtrPtr,
                                     StcpLength *theReceiveLengthPtr);
extern RpErrorCode StcpSend(StcpConnection theConnection,
                            char *theSendPtr,
                            StcpLength theSendLength);
extern RpErrorCode StcpSendStatus(StcpConnection theConnection,
                                  StcpStatus *theSendStatus);
extern RpErrorCode StcpCloseConnection(StcpConnection theConnection);
extern RpErrorCode StcpCloseStatus(StcpConnection theConnection,
                                    StcpStatus *theCloseStatus);
extern RpErrorCode StcpAbortConnection(StcpConnection theConnection);
```