SoftStax™

MICROWARE®

Software Components • Consulting Services • Development Tools
Embedded Solutions for Successful Products™

Microware's SoftStax™
Networking Solution

the
superior
networking communications
solution

fielded • proven • low-risk • easy-to-use • unified architecture

Microware's SoftStax™

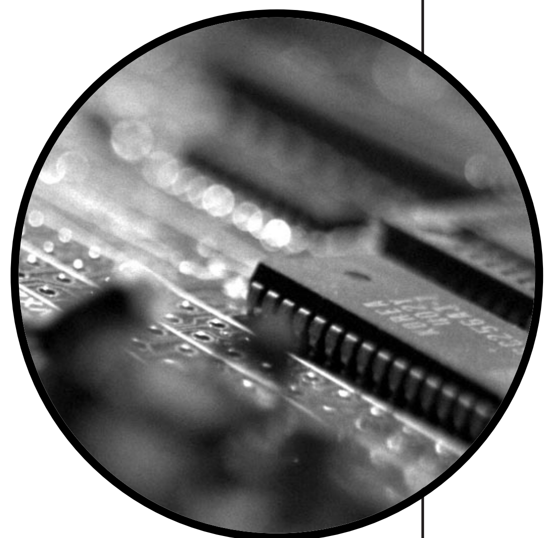
ABSTRACT

Communications software development is about more than application development. The baseline system software, which controls the microprocessor and network interface, must also be created. Baseline components include:

- A real-time operating system (RTOS)
- One or more network interface device driver(s)
- One or more protocol stack(s) and associated programming interfaces

Today's software developers must learn the design philosophy behind each component and forge a coherent software baseline before application development can be completed. Since each component is developed using a different design philosophy, creating this baseline can be extremely challenging. The resulting application environment is often clumsy, confusing, and unnecessarily large. To make the application environment simple and understandable, developers must rewrite most, if not all, of the components under one unified design philosophy. This paper describes Microware's SoftStax™ networking solution, the integrated communications framework for OS-9®. This solution:

- Is a completely open and specified framework
- Ensures all baseline components for the OS-9® Real-Time Operating System (RTOS) use one optimal design philosophy
- Eliminates interworking each baseline component and provides an application environment that makes development simple and more understandable

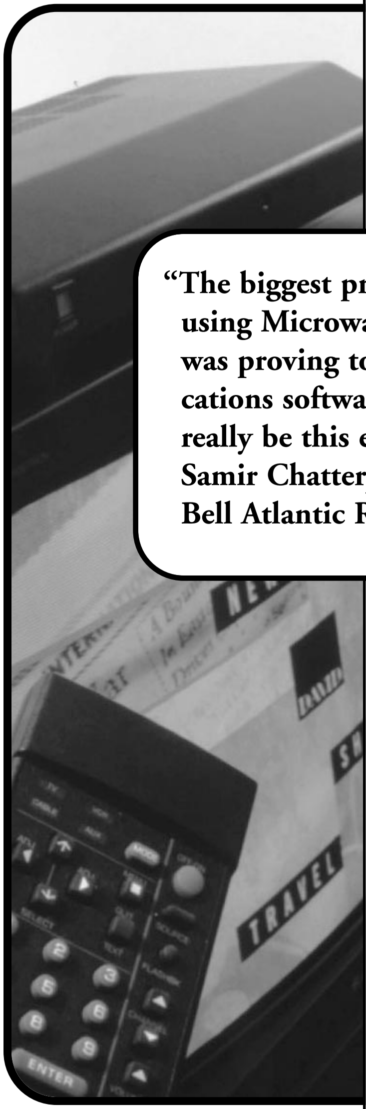


INTRODUCTION

Communications software developers face many challenges:

- Short development time
- Not enough developers
- Product reliability concerns
- Lack of effective tools to fix real-time bugs
- Cost/performance pressures
- Rapid technology advances/changes

These challenges reflect one main obstacle — the lack of an optimal software base-line that provides a simple and understandable application environment with the ability to "snap in and out" underlying network technologies as they evolve without disturbing the application. Microware's networking solution provides a simple and understandable application environment that enables underlying network technologies to "snap in and out" without disturbing the application. More finished code is included, making it easy to write network-independent applications. This paper describes the technical detail behind Microware's networking solution: the architecture, design philosophy, application environment, and protocol stack framework that overcomes the obstacles faced during communications software development.

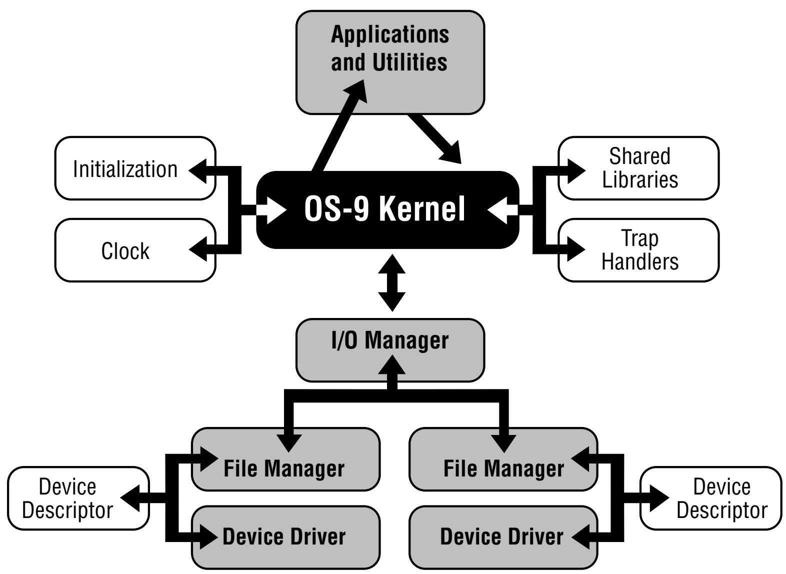


ARCHITECTURE AND DESIGN PHILOSOPHY

Samir Chatterjee and the Bell Atlantic team members each had over ten years of experience developing communications software. The tendency was to assume that services that needed to be provided had to be developed. To the Bell Atlantic team's surprise, the Microware SoftStax networking framework already implemented all the details, so the team could get to the critical tasks—creating new multimedia service network custom protocols and applications interoperable across multiple network topologies.

> **"The biggest problem we had to overcome using Microware's OS-9® and SoftStax™ was proving to ourselves that communications software development could really be this easy."**
> **Samir Chatterjee**
> **Bell Atlantic Research**

Microware's networking environment follows gracefully from the overall architecture and design philosophy of OS-9 itself. OS-9 implements a unified Input/Output (I/O) system. The programming interface used by the application is identical whether the application is using a hard drive, serial device, or network interface. This programming interface consists of calls to open, close, read, write and set/get I/O configuration information (called setstats and getstats).

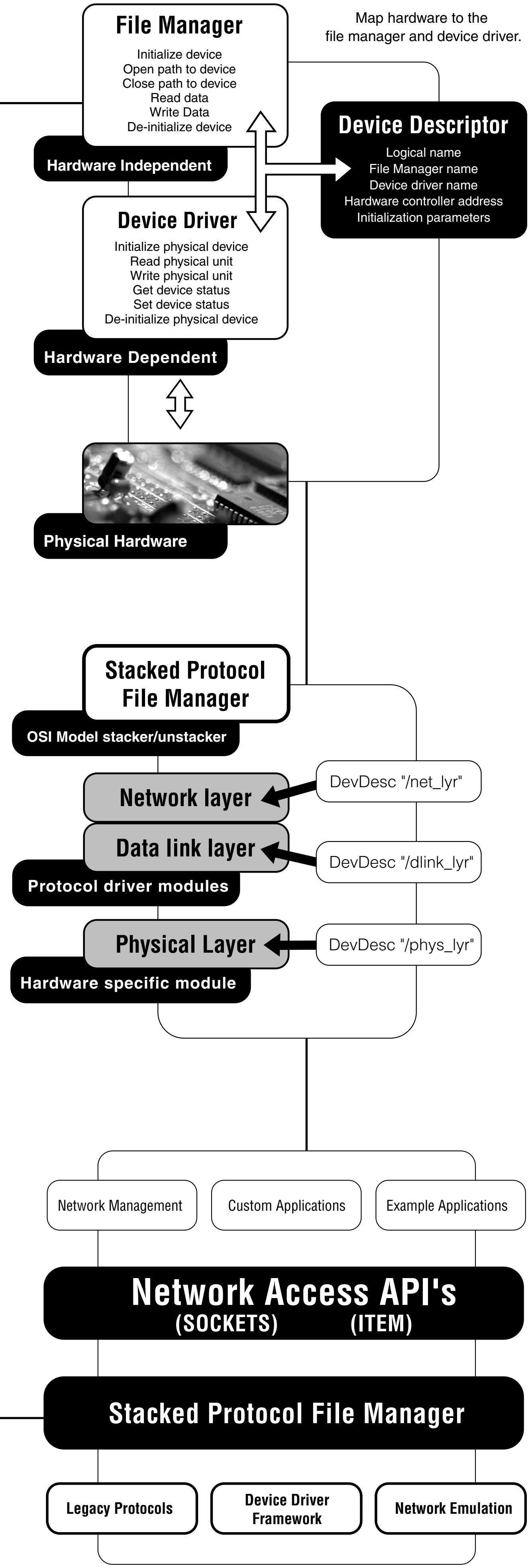


Every I/O system for OS-9 consists of a file manager, device driver, and device descriptor. The file manager performs all logical features of the specific I/O system, implementing the Hardware Abstraction Layer (HAL) for the system. The device driver controls the specific hardware, distilling driver creation down to hardware initialization, termination, and an interrupt service routine. The device descriptor is identifiable by the application that dynamically links all the modules. The application opens a path using a device descriptor module name. Then OS-9 uses the information contained in the device descriptor as a roadmap to create a link between the application, file manager, and device driver. The link created by OS-9 for the application is called a path. The application uses the resulting path to access the services provided by the I/O system. All modules in the system are fully re-entrant and position independent, two very important characteristics of an RTOS if dynamic download and upgrade facilities are to be available.

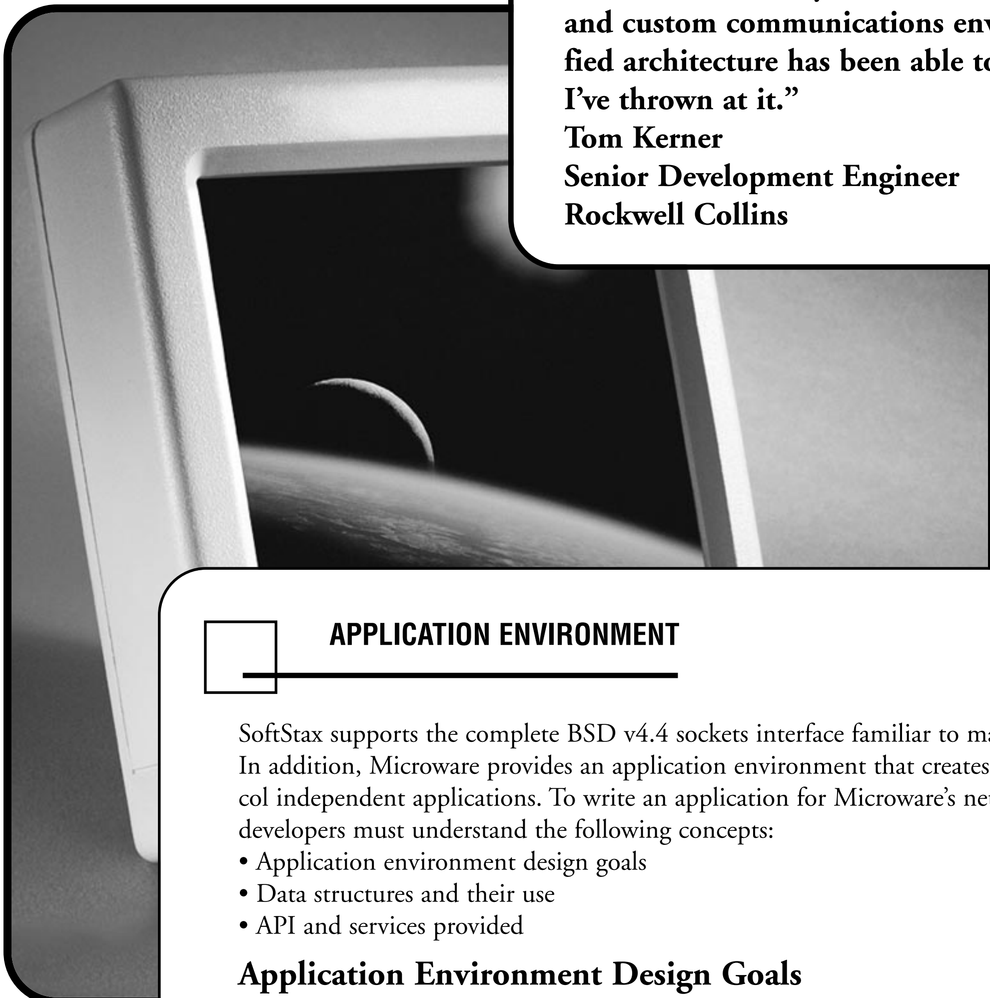
Microware's SoftStax networking solution extends the I/O system philosophy by enabling the mapping of not just one driver on a given path, but allows multiple drivers to be stacked on one another. This extension represents the implementation of the OSI Model as defined by the International Standards Organization. The OSI Model specifies abstractly the services provided within seven stackable layers and is used as the foundation design philosophy for all protocol specifications. SoftStax represents a concrete implementation of the OSI Model specification.

Since Microware's SoftStax models the OSI Model instead of "fighting" it, protocol layer implementation is easy, understandable, and interoperable with other protocol layer (or protocol driver) implementations for OS-9. It is also important to note that since the implementation for OS-9 is a natural extension to the core OS-9 kernel, Microware's integrated solution maximizes performance while minimizing footprint and CPU utilization.

Microware's environment consists of an Application Programming Interface (API) called ITEM (Integrated Telephony Environment for Multimedia), the Stacked Protocol File Manager (SPF), a template protocol driver (spproto), a network emulation driver (sploop), and various HDLC driver implementations. Network-independent application examples are also provided and ready to run for quick familiarization with the environment and providing a guideline for application development.



"We've been using Microware's SofStax networking solution for more than two years in multi-protocol TCP/IP, ISDN and custom communications environments. Microware's unified architecture has been able to handle everything I've thrown at it."
Tom Kerner
Senior Development Engineer
Rockwell Collins



APPLICATION ENVIRONMENT

SoftStax supports the complete BSD v4.4 sockets interface familiar to many programmers. In addition, Microware provides an application environment that creates network and protocol independent applications. To write an application for Microware's networking solution, developers must understand the following concepts:

- Application environment design goals
- Data structures and their use
- API and services provided

Application Environment Design Goals

ITEM defines the application environment of Microware's SoftStax framework. SoftStax is an optimal software baseline, providing a simple and understandable application environment with the ability to snap in and out underlying network technologies without disturbing the application.

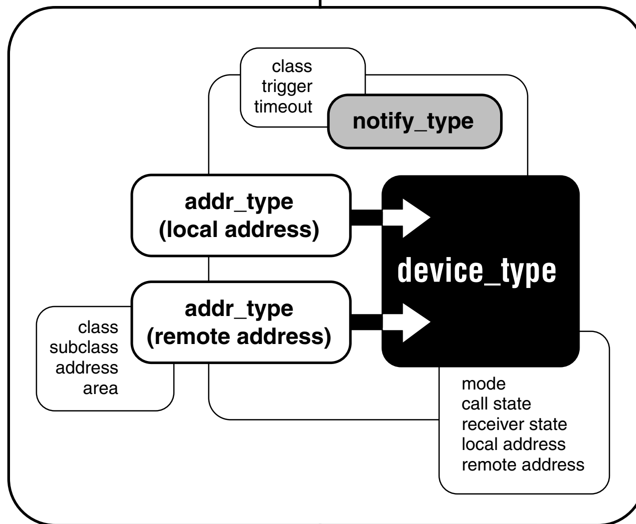
Simple application development — One design goal of ITEM is to eliminate the complexities involved with an application using network services. With ITEM, the applications are not forced to build pieces of network-specific messages and pass them through the API to perform call control. For example, an ATM application is not required to pass in the channel ID, bearer capability, and low-layer compatibility information elements as parameters in order make a connection. This simplifies the application, frees the application from being network specific and does not require the programmer to be "ATM-literate."

Easily understandable applications — A second design goal of ITEM is to use an API paradigm familiar and intuitive to the programmer while not reflecting a specific state machine. ITEM achieves this. For example, applications written using an ISDN API wouldn't just disconnect. The application would also release after a far end disconnect or release complete after initiating disconnection. This is a reflection of the ISDN state machine transitioning from active to disconnect, release and release complete.

Network independence — The third design goal of ITEM is to specify an API and data structures, enabling applications to be network independent. ITEM enables application binaries to run across multiple network topologies without recompiling or relinking. Network independence is achieved by abstracting properties of the network.

Data Structures and Their Use

To achieve application environment design goals, data structures were created to enable the application to remain network independent. Abstracting application visible aspects of any network is the key to making network independence a reality. Abstractions for the network device and network addressing were created using structures called device_type and address_type. The third data structure in ITEM abstracts the asynchronous notification method called a notify_type structure. This provides a level of operating system independence.



The descriptor automatically initializes all of the parameters in the device_type and addr_type structures when the path is created. Since automatic initialization occurs as an implicit kernel service, applications need not be aware of these two structures. This enables applications in their most simple form to still operate with ITEM. If required, ITEM contains API calls to get and set all variables within the device_type and addr_type structures. Applications use the notify_type structure for network event registration and removal. Notification requests can be set through the ITEM API for:

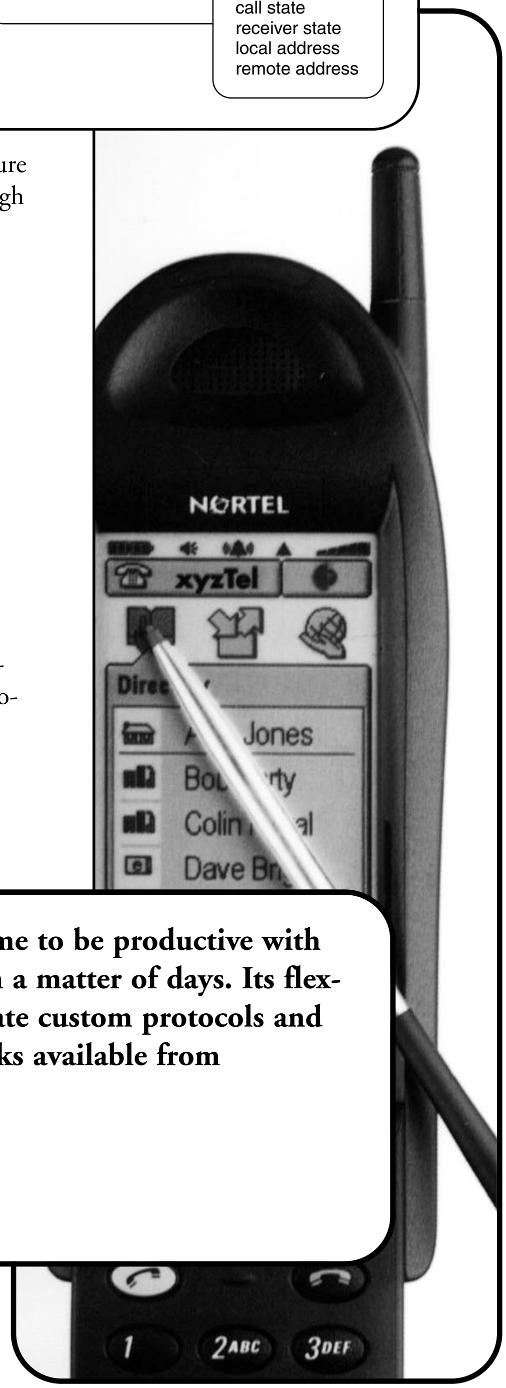
- Link down/link up
- Incoming call
- Connection active/far-end hang up
- Data available to be read
- End of MPEG-II program
- Flow control on/off
- Custom protocol or device driver network events

Application Programming Interface (API)

The API is another important characteristic of the application environment. The ITEM API is modeled after the telephone, a paradigm that everyone is familiar with. ITEM provides scalable capabilities and is simple to use. However, in cases where applications require complex network-specific services, add-on communications paks come with APIs that expose detailed access to particular network topologies. For example, SoftStax also includes a BSD4.4 compatible socket library. The advantage of this approach is that the developer knows the level of network independence for the libraries used by the application.

The ITEM API contains five main categories of service:

- Device oriented
- Path oriented
- Call control
- Data manipulation
- Asynchronous notification



"The product and training enabled me to be productive with Microware's networking solution in a matter of days. Its flexible architecture enabled me to create custom protocols and drivers that plug into protocol stacks available from Microware and third parties."
Alan Granum
Development Engineer
Nortel

Device-oriented calls – These calls manipulate individual protocol layers or device drivers. They include calls to initialize and terminate individual layers, get and set permissions for a layer, get the layer name, and get the type of service the layer provides.

Call	Parameters	Description
ite_dev_attach	Name String, Mode, Handle	Initialize the device(layer)
ite_dev_detach	Handle	De-initialize the device(layer)
ite_dev_getmode ite_dev_getname	Path ID, Mode Path ID, Name String	Get permissions (read,write...) Get device(layer) name
ite_dev_gettype	Path ID, Input Type, Output Type	Get device(layer) type (OOB signaling, MPEG, ...)
ite_dev_setmode	Path ID, Mode	Set permissions (read, write...)

Path-oriented calls – These calls manipulate entire protocol stacks for a given path. Calls to open and close incarnations of a protocol stack and to dynamically add and remove protocol layers are also available. Profiles are used to simplify the correct quality of service for connections by the applications. These profiles are identified by the application as primitives (i.e. VOICE, DATA, MPEG, IP, etc). This way, applications can request connections based on a service profile primitive. The protocol layer maps the primitive to the specific connection messages required to create the correct type of connection for the service desired.

Call	Parameters	Description
ite_path_open	Name String, Mode, Path ID pointer, Address Pointer	Open protocol stack instance
ite_path_close	Path ID	Close protocol stack instance
ite_path_push	Path ID, Name String	Add layer to stack
ite_path_pop	Path ID	Remove layer from stack
ite_path_profileget	Path ID, Conn Type, Profile Size, Profile Buffer	Get connection service profile
ite_path_profileset	Path ID, Conn Type, Profile Size, Profile Buffer	Set connection service profile

Call-control calls – This group of calls provides call-control services required for connection-oriented networks. The framework of Microware's SoftStax allows these calls to be made successfully even if the application is running over a connectionless network, providing true portability across all types of network topologies.

Call	Parameters	Description
ite_data_read	Path ID, Buffer, Size	Read data
ite_data_write	Path ID, Buffer, Size	Write data
ite_data_avail_asgn	Path ID, Notify_type	Request notification when data available to be read
ite_data_avail_rmv	Path ID	Remove data available request
ite_data_ready	Path ID, Data Count	Return number of bytes data available to be read
ite_data_readmbuf	Path ID, mbuf container	Zero copy read API call
ite_data_writembuf	Path ID, mbuf container	Zero copy write API call

Data manipulation calls – The data manipulation calls enable synchronous or asynchronous reading and writing operation. Zero copy across the user interface is available, not just with TCP/IP, but with all Microware networking protocols through the read and write mbuf calls. Data can also be read by packets or individual bytes for the convenience of the application.

Call	Parameters	Description
ite_ctl_addrset	Path ID, Local Addr_type, Remote Addr_type	Set local/remote abstract addressing
ite_ctl_connstat	Path ID, Device_type	Get device/addr information
ite_ctl_connect	Path ID, Local Addr_type, Remote Addr_type, Notify_type	Make a call (notify on connect)
ite_ctl_disconnect	Path ID	Hang up a call
ite_ctl_answer	Path ID, Notify_type	Answer call (notify on connect)
ite_ctl_suspend	Path ID	Put caller on hold
ite_ctl_resume	Path ID, Notify_type	Resume call previously on hold (notify on resumption)

Call	Parameters	Description
ite_fehangup_asgn	Path ID, Local Addr_type, Remote Addr_type	Notify on far-end hang up
ite_fehangup_rmv	Path ID, Device_type	Remove FE hang up notification
ite_linkdown_asgn	Path ID, Local Addr_type, Remote Addr_type, Notify_type	Notify on link down
ite_linkdown_rmv	Path ID	Remove link down notification
ite_linkup_asgn	Path ID, Notify_type	Notify on link up
ite_linkup_rmv	Path ID	Remove link up notification

Asynchronous notification calls – Far-end hang up and protocol stack status change can also be registered by the application in addition to the asynchronous calls defined by the previous sections. The facility also allows layer-specific notifications, if required.

Using the Application Environment

Below is an example of a stack consisting of an ISDN driver, LAP-D data link layer, and Q.931 network layer.

There are three ways the application can invoke this configuration:

Explicitly, 1 call

ite_path_open("/isdn0/lapd/q931", READ | WRITE, &pathID, NULL);

Explicitly, 3 calls

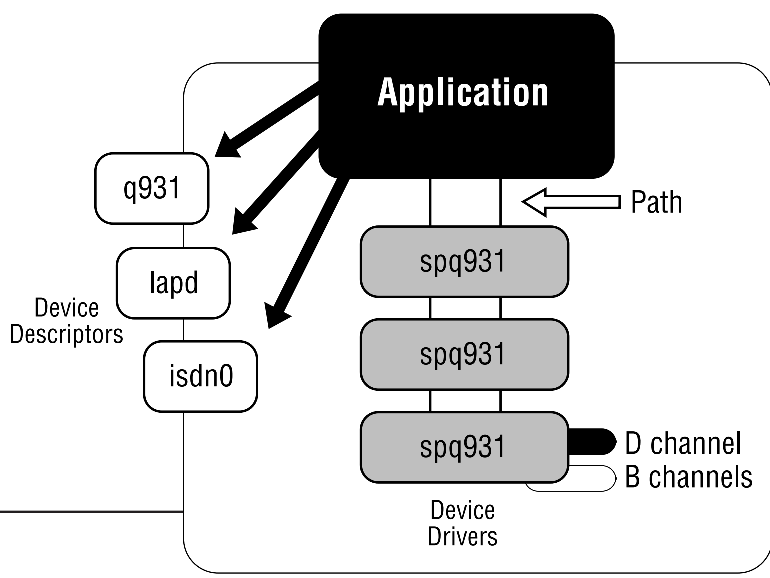
ite_path_open("/isdn0", READ | WRITE, &pathID, NULL);
ite_path_push(pathID, "/lapd");
ite_path_push(pathID, "/q931");

Implicitly

ite_path_open("/network", READ | WRITE, &pathID, NULL);

In this case, the isdn0 descriptor is configured to contain an implicit push of the /lapd/q931 stack. This descriptor is then named /network. In this manner, the application simply opens /network. New descriptors containing different protocol stacks can be loaded into the OS-9 system. This method enables the application to run over different network topologies without disruption. Addressing can be defined by using the '#' delimiter when opening each layer. Using the ISDN example above, spisdn uses D channel, splapd uses TEI/SAPI {00}, and spq931 uses 515-223-8000 for their respective addresses. The open call would look like:

ite_path_open("/isdn0#D/lapd#00/q931#5152238000", READ|WRITE, &pathID, NULL);



Developers must define the following to write an application for Microware's *Soft* Stax :
• Design goals
• Driver architecture
• Optimized driver services
• Data and control flow through the architecture

Design goals

SoftStax defines the communications software framework. Microware provides the optimal software baseline that provides a simple and understandable application environment with the ability to snap in and out underlying network technologies without disturbing the application.

> **"Microware's SoftStax networking solution reduced the risk to our project by providing an integrated communications framework, excellent training, and quality support throughout the project. It also enabled us to build a high performance IP over ATM multimedia service network with a lower memory footprint."**
> **Derek Noble**
> **Product Development Manager**
> **Nortel**

Optimal software baseline

The Core SoftStax environment is 20Kb RAM and 25Kb ROM for all processor architectures. The architecture was not designed for portability across operating systems. Microware's solution is a kernel extension that utilizes services unique to OS-9 to provide a run-time communications architecture that maximizes performance and minimizes footprint andCPU utilization.

Open architecture — Microware's SoftStax is completely documented and specified to allow all third-party protocol stack companies, hardware driver providers, and SoftStax users to efficiently implement their technologies for OS-9.

Protocol stack and layer interoperability – SoftStax provides one universal framework for every protocol layer. This enables protocols implemented by multiple parties to be interoperable.

> **"Its universal framework enabled our technologists to complete Universal Serial Bus development for Microware in just four weeks. Microware's networking solution is an easy and efficient solution for implementing Universal Serial Bus protocol technology for OS-9."**
> **Thierry Giron**
> **Director of Engineering**
> **Award Software International**

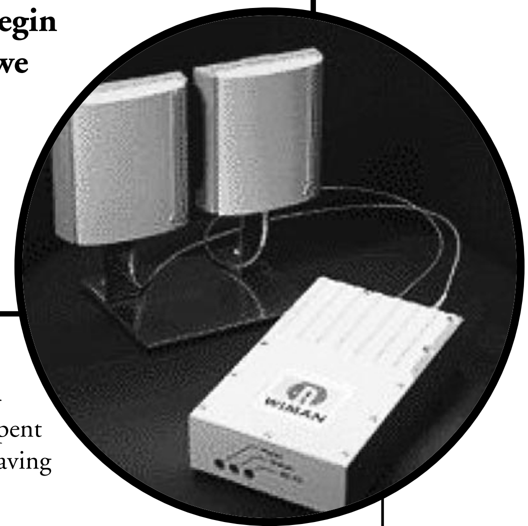
Simple protocol stack development

SoftStax provides an easy to learn and use framework that includes a protocol layer template driver, network emulation driver, timer services, and buffer management services. The template driver provides a "null layer" implementation to which a protocol state machine can be immediately added. The network emulation driver enables validation of protocol stacks without requiring access to the network. Timer services and buffer management services are also provided.

Easy understanding of protocol stack add-ons – Communications software development requires integration of an RTOS, application, one or more protocol stacks, and device drivers, all written to different frameworks. Microware's SoftStax enables developers to immediately understand a common baseline regardless of the Microware networking solutions product add-on.

Effective debugging of real-time problems – Microware's networking solution provides a facility for tracing the events leading up to real-time bugs. This facility is provided through a debugging library for real-time execution capture.

"Microware's networking solution simplifies complex communications equipment problems. The off-the-shelf Communications Paks eliminate wasted effort of porting protocol stacks, enabling us to begin application development quickly. The technical support we were given was excellent. Microware provided timely and knowledgeable answers and experience to the project."
Sharon Harris
Software Design Engineer
PowerSys



Protocol stacks available in source and binary form

Providing protocol stack binaries eliminates development effort spent porting the protocol stack to a particular RTOS environment. Having access to source code enables control over the implementation.

Plug-In Capabilites:

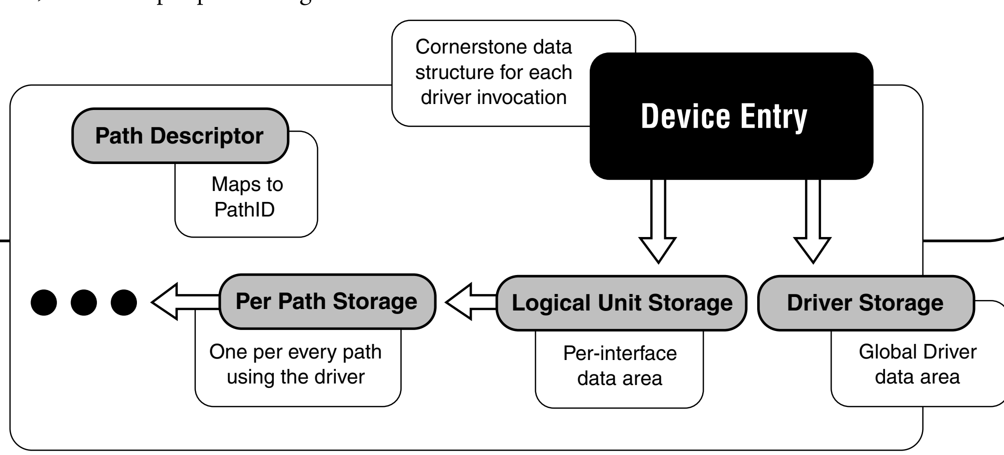
BSD v4.4 TCP/UDP/IP	USB Host/Peripheral
ISDN (Q931, Q921)	Voice Over IP
X.25 (X.25, LAP-B)	Web/HTTP server
SNMP	SMTP/POP3 e-mail client
ATM Signaling	Web Browsers (Java and Native)
IPoA	IEEE 1394

Driver Architecture

Protocol driver data structures – The OS-9 kernel provides automatic allocation and initialization of data structures for a driver. This service is used by SoftStax to allocate and initialize data areas for protocol drivers without requiring creation of code to allocate and initialize data areas. OS-9 automatically creates four data structures for a driver, including the following:

- Device entry
- Driver storage
- Logical unit storage
- Path descriptor

A library is also provided to create a per path data structure for the driver, called the per path storage.



Entry points of a protocol driver:

Entry Point	Parameters	Description
dr_iniz	Device entry	Initialize protocol state machine/HW
dr_term	Device entry	De-initialize protocol state machine/HW
dr_getstat	Device entry, param blk	Retrieve control information
dr_setstat	Device entry, param blk	Set control information
dr_updata	Device entry, buffer	Incoming PDU for processing (going up)
dr_downdata	Device entry, buffer	Downgoing PDU for encapsulation

All ITEM API calls are realized at the driver layer as DrGetstat and DrSetstat calls. Parameter blocks are formatted with the ITEM service request and associated parameters. For device drivers, the DrUpdata entry point is not used, and an interrupt service routine, which can be considered as the incoming data entry point for a device driver, is implemented.

Inter-driver communication primitives:

Macro Call	Parameters	Description
SMCALL_UPDATA	Device entry, drvr above device entry, buffer	Pass PDU up
SMCALL_DNDATA	Device entry, drvr below device entry, buffer	Pass PDU down
SMCALL_GS	Device entry, adjacent drvr device entry, param blk	Pass control request up/down
SMCALL_SS	Device entry, adjacent drvr device entry, param blk	Pass control request up/down

Macro Call	Parameters	Description
DR_FMCALLUP_PKT	Device entry, drvr above device entry, buffer	Queue packet for incoming data processing

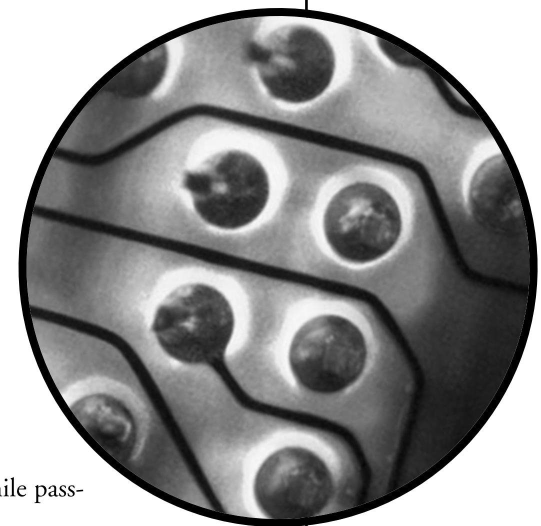
Inter-driver communication primitives are implemented not as inter-process communication, but as direct jumps to the entry point of the driver above and below. This aspect is the key to a high performance system.

The DR_FMCALLUP_PKT macro minimizes the amount of time spent in an interrupt service routine by queuing the data on a receive queue for processing by the receive process.

Optimized Driver Services

Some of the issues that have a negative effect on protocol processing performance include:

- Copying data packets during transmission and reception
- Data container facility
- Timer services



Eliminating copies – Microware's networking solution has facilities to eliminate data copying for three common scenarios:

- Application reads
- Data packet storage for retransmission
- Moving or shifting data for data encapsulation

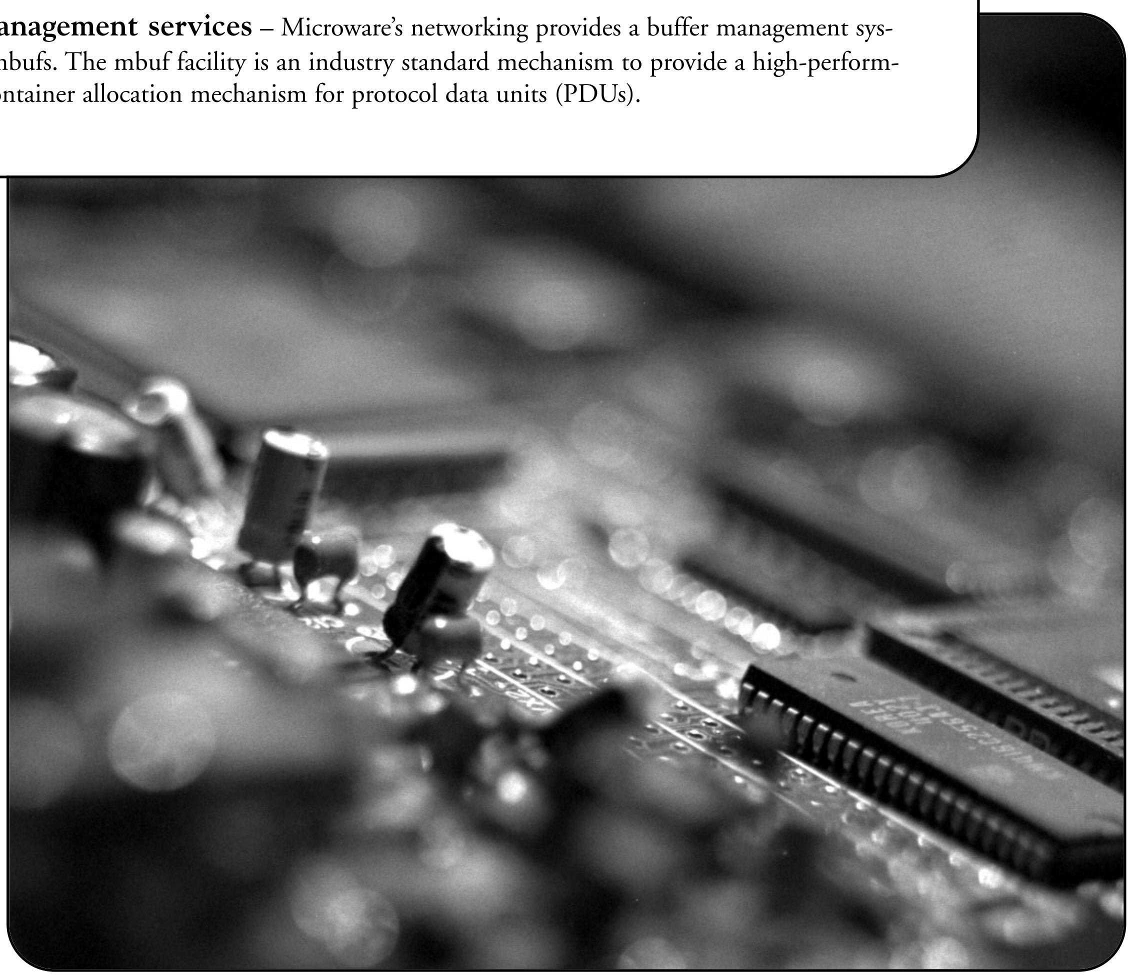
The ITEM API provides read and write mbuf calls that provide what the industry calls zero copy facilities for application reading. The Microware environment also provides other "zero copy" facilities.

1. Microware's networking solution provides the SPF_NOFREE facility, enabling protocol drivers to keep a pointer to a single data container while passing the same buffer down for transmission instead of making a copy.

2. Microware's networking solution uses the SPF_GS_UPDATE facility to collect protocol stack header and trailer requirements when the stack is created or modified.

This ensures that there is reserved space for headers and trailers in the data container, eliminating the extra copy or data shifting to make room for headers and trailers by each layer.

Buffer management services – Microware's networking provides a buffer management system called mbufs. The mbuf facility is an industry standard mechanism to provide a high-performance data container allocation mechanism for protocol data units (PDUs).



Feature	Benefit
Microware's Networking Application Environment	
Network independent	API Applications portable across network topologies
3 data structure abstractions	Network independent applications
Based on telephone paradigm	Easy to learn and use
Network specific APIs w/add-ons	Detailed network service access available
Sync & async data reads/writes	Smaller code, simplifies application development
Byte or packet oriented reading	Smaller code, simplifies application development
Zero copy through mbuf read/write calls	High performance
Dynamic protocol layer pushing/popping	Flexible application development
'#' delimiter	Smaller code, simplifies application development
Networking Components	
File manager extension to OS-9	Maximum performance, minimum footprint & CPU utilization
Application examples	Simple & understandable application development
Protocol driver template (spproto)	Ensures layer interoperability, less development effort
Network emulation driver (sploop)	Efficient application & protocol testing
HDLC driver sources & binaries	Less development effort
25 kilobyte footprint framework	Small footprint, modularly scaleable
Protocol stack add-ons in source & binary	Eliminates porting effort while enabling source modifications
Network Oriented OS-9 Properties	
Process model RTOS	Secure, resistant to attack
Re-entrant, position independent	Simple application development
Dynamic download & upgrade	100% availability even during maintenance
OS-9 alarms	High resolution timer service

Facilities

Structure auto-allocation & initialization	Smaller code, less development
Inter-driver communication macros	High performance
DR_FMCALLUP_PKT macro	Minimizes time spent in interrupt context
SPF_NOFREE "-1 copy"	High performance protocol stack operation
SPF_GS_UPDATE "-2 copy"	High performance protocol operation
Mbuf buffer management	Industry standard, high performance, less development effort
Timer services	High performance, less development effort

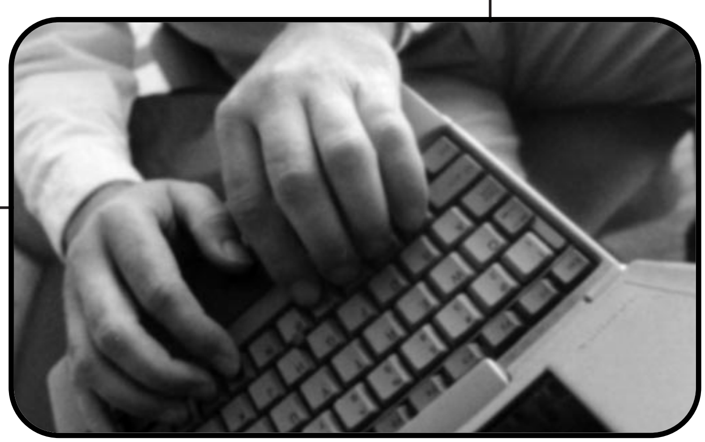
TECHNICAL SUMMARY

System Requirements	68K		PowerPC		StrongARM /ARM		X86		SH	
	ROM	RAM*	ROM	RAM*	ROM	RAM*	ROM	RAM*	ROM	RAM*
OS-9 kernel	27652	32K	80008	64K	84232	64K	60912	64K	72064	64K
SoftStax networking solution	18156	20K	22680	20K	21928	20K	17080	20K	18808	20K
Mbuf management system	2444	64K **	6728	128K**	6056	128K**	4000	128K**	5096	128K**
Template protocol	3444	2K	4832	2K	4424	2K	2896	2K	3456	2K
Network emulation	5784	3K	10184	3K	9480	3K	6944	3K	8220	3K
Serial console support	4010	8K	27080	20K	28336	20K	18004	20K	22632	20K
Floppy/hard drive support	10940	10K	44144	26K	53456	26K	34624	26K	60166	26K
IP stack Ethernet support	88772	20K	170K	20K	176K	20K	148K	20K	178K	20K
PPP/SLIP	82272	30K	99K	30K	142K	30K	100K	30K	113K	30K
Network File System client	—	—	—	—	—	—	—	—	—	—
Network File System server	—	—	—	—	—	—	—	—	—	—

*RAM sizes are estimates based on a single application's access to the I/O system.

**Memory pool for buffer management minimum recommended size. This number is user configurable to any size.

— Information not available at time of publishing.



Documentation

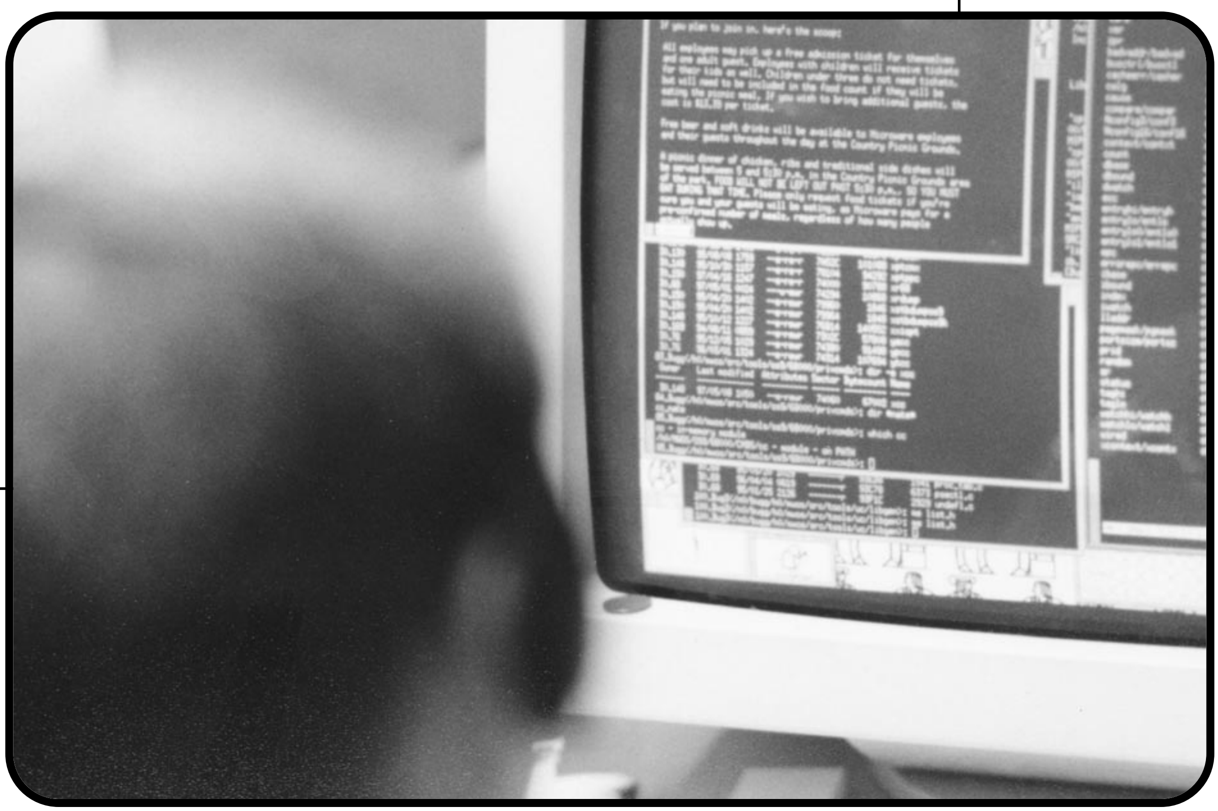
Documentation is provided in PDF format and is tutorial style for easier understanding of the product. The Microware networking solutions manual set includes:

- Getting Started Manual
- Programming Reference Manual
- Using SoftStax Manual (Microware's Networking Solution)
- Porting Guide

The Getting Started Manual explains the installation and initial configuration of the environment. The Programming Reference Manual provides syntax and semantics for all ITEM API calls. The Using SoftStax Manual provides information for the application developer. The Porting Guide targets the protocol stack or device driver developer.

CONCLUSION

The key to Microware's SoftStax networking solution is an optimal software baseline that provides a simple and understandable application environment with the ability to "snap in and out" underlying network technologies as they evolve, without disturbing the application. Microware's SoftStax enables developers to increase productivity and profitability while significantly differentiating their product in the market.



SoftStax™

Microware's SoftStax™
Networking Solution

technical
white
paper

CORPORATE OFFICE

NORTH AMERICA

Microware Systems Corporation
1500 N.W. 118th Street
Des Moines, IA 50325 USA
Tel: (515) 223-8000
Sales: (888) 642-7609
Toll-free: (800) 475-9000
Fax: (515) 224-1352
E-mail: info@microware.com
WWW: www.microware.com

MICROWARE®

Embedded Solutions for Successful Products™
SOFTWARE COMPONENTS • CONSULTING SERVICES • DEVELOPMENT TOOLS

INTERNATIONAL OFFICES

JAPAN

Microware Systems K.K.
5-2, Sotokanda 3-Chome
Chiyoda-Ku
Tokyo 101-002, Japan
Tel: +81 3-3257-9000
Fax: +81 3-3257-9200
E-mail: info@microware.co.jp
WWW: www.microware.com/japan

Osaka Office
4F Yotsuhashi Okawa Bldg.
1-6-23 Shinmachi
Nishi-Ku
Osaka 550-0013, Japan
Tel: +81 6-6535-6557
Fax: +81 6-6535-6558
E-mail: info@microware.co.jp

EUROPE AND ELSEWHERE:

France
(Southern Europe & Middle East)
Microware Systems France
LP 908 - Les Conquerants
1 Avenue de l'Atlantique
ZA de Courtaboeuf
F- 91976 LES ULIS cedex
France
Tel: +33 (0)1 60 92 36 70
Fax: +33 (0)1 60 92 36 79
E-mail: info@microware.fr

Germany
(Central & Eastern Europe)
Microware Systems Corporation
Haringstrasse 19
D-85635 Hoehenkirchen
Germany
Tel: +49 8102 7422 0
Fax: +49 8102 7422 99
E-mail: info@microware.de

United Kingdom
(Northern Europe & Far East)
Microware Systems Ltd.
1 Holly Court
3 Tring Road
Wendover
Buckinghamshire
HP22 6PE
United Kingdom
Tel: +44 (0)1296 628100
Fax: +44 (0)1296 628117
E-mail: info@microware.co.uk

The Netherlands
(Belgium, Luxembourg, and Scandinavia)
Microware Systems B.V.
Jan Ligtharstraat 1
1817MR Alkmaar
The Netherlands
Tel: +31 72 - 5143 510
Fax: +31 72 - 5143 512
E-mail: info@microware.co.uk

1044-0022 06-13-00