BITS ZG629T: Dissertation

Remote Management of Wireless Gateway

Dissertation work carried out at

Mphasis Technologies Ltd, Bangalore

by

Dinesh D N

2004HZ12158

Dissertation submitted in partial fulfillment of M.S. (Software Systems) Degree under the Supervision of Baba Saheb, Associate General Manager, Mphasis Technologies Ltd, Bangalore

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE
PILANI (RAJASTHAN)
March 2006

CERTIFICATE

This is to certify that the Dissertation entitled “Remote Management of Wireless Gateway” submitted by Dinesh D N having ID-No. 2004HZ12158 for the partial fulfillment of the requirements of M.S. Software Systems degree of BITS, embodies the bonafide work done by him under my supervision.

_____

Signature of the Supervisor

Place : Bangalore

Date : _____

Baba Saheb,
Associate General Manager,
Mphasis Technologies Ltd
Bangalore

Birla Institute of Technology & Science, Pilani

Distance Learning Programmes Division

Fourth Semester 2005-2006

BITS ZG629T: Dissertation

ID No.	**: 2004HZ12158**
NAME OF THE STUDENT	**: Dinesh D N**
EMAIL ADDRESS	**: dinesh.nagaraj@mphasis.com**
EMPLOYING ORGANIZATION	**: Mphasis Technologies Ltd**
SUPERVISOR'S NAME	**: Baba Saheb**
SUPERVISOR'S EMAIL ADDRESS	**: baba.saheb@mphasis.com**
DISSERTATION TITLE	**: Remote Management of Wireless Gateway**

ABSTRACT :

The remote management of wireless gateway is about monitoring, controlling and managing wireless gateway products. The external device management interfaces are supported through common set of services, routines to access the MIB. This would provide seamless flexibility to enhance the MIB in future, if required.

Key words: Windriver, WMIT, SNMP, MIB, AGENT, Managed Device, Wireless, Wireless Gateway, SNMP Agent, SNMP Manager, Backplane, Web, CLI, Telnet, NMS

_____	_____
Signature of the Student	**Signature of the Supervisor**
Name: Dinesh D N	**Name: Baba Saheb**
Date: 21-March-2006	**Date: 21-March-2006**
Place: Bangalore	**Place: Bangalore**

Acknowledgements

The author is grateful to Mr Baba Saheb, Associate General Manager, Mphasis Technologies Ltd and Mr Denis Perelyubskiy, Software Engineer for their technical help and assistance throughout the project.

The author thanks Mphasis Technologies Ltd for providing access to all the vital hardware, software and tools that were needed to work on the project.

Table of Contents

List of Figures

List of Tables

Introduction

The *Remote Management of Wireless Gateway* is about remotely controlling, configuring, monitoring and managing the network devices, a wireless gateway, using external management interfaces like Web, CLI and SNMP Browser. The core protocol this technology is built around is SNMP, *Simple Network Management Protocol*. The SNMP is an application layer protocol that facilitates the exchange of management information between network devices.

Wireless Communication is a term used to describe telecommunications in which electromagnetic waves (rather than some form of wire) carry the signal over part or the entire communication path.

The *Wireless Gateway* is an edge network device, meant for internetworking, a system that joins two networks together.

Presently, the three external device management interfaces, CLI, Web and SNMP Browser, adopt different technique to configure and manage the managed device. This makes the task of enhancing the managed features, MIB, tedious, as it requires enhancement to all the three interfaces. The WMIT, Windmanage Integration Tool, by Windriver System, converge CLI and Web access to the MIB using WMIT Backplane services to provide a common set of APIs to access the MIB. This simplifies the work of enhancing the MIB in future and the user could seamlessly upgrade the MIB with negligible effort.

Chapter 1 – SNMP Protocol

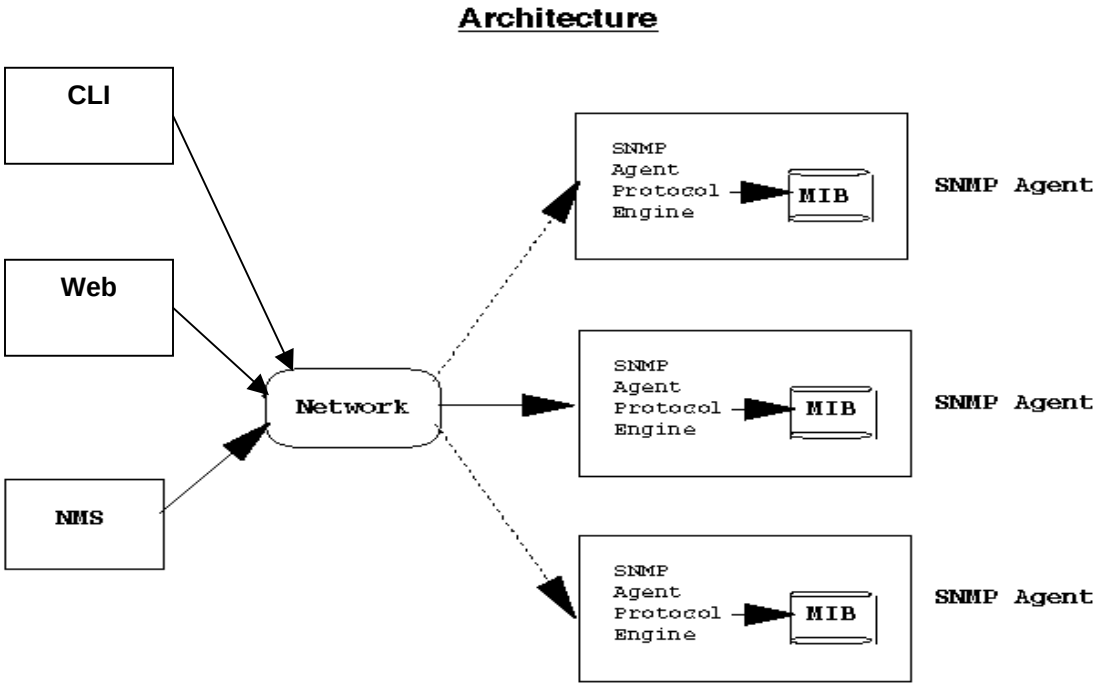


Figure 1– SNMP Operation

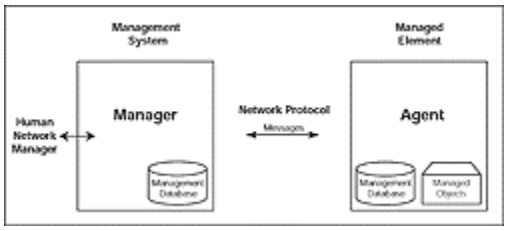
The *Simple Network Management Protocol (SNMP)* is an application layer protocol that facilitates the exchange of management information between network devices. Two versions of SNMP exist: SNMP version 1 (SNMPv1) and SNMP version 2 (SNMPv2). Both versions have a number of features in common, but SNMPv2 offers enhancements, such as additional protocol operations. Managed devices are monitored and controlled using four basic SNMP commands: **read**, **write**, **trap**, and **traversal** operations. SNMP lacks any authentication capabilities, which results in vulnerability to a variety of security threats.

A Management Information Base (MIB) is a collection of information that is organized hierarchically. MIBs are accessed using a network-management protocol such as SNMP. They are comprised of managed objects and are identified by object identifiers. A managed object is one of any number of specific characteristics of a managed device. Managed objects are comprised of one or more object instances, which are essentially variables. Two types of managed objects exist: scalar and tabular. *Scalar objects* define a single object instance. *Tabular objects* define multiple related object instances that are grouped in MIB tables.

The MIB/ SNMP Browser is an NMS application that could monitor and control managed device, Wireless Gateway in our case. A *managed device* is a network node that contains an SNMP agent and that resides on a managed network. Managed devices collect and store management information and make this information available to NMSs using SNMP. Managed devices, sometimes called network elements, can be gateways, routers and access servers, switches and bridges, hubs, computer hosts, or printers.

SNMP Agent is a network-management software module that resides in a managed device. An agent has local knowledge of management information and translates that information into a form compatible with SNMP.

SNMP protocol is based on the manager/agent model consisting of a manager, an agent, a database of management information, managed objects and the network protocol. The manager provides the interface between the human network manager and the management system. The agent provides the interface between the manager and the physical device(s) being managed.



SNMP is based on the manager/agent model of a network management architecture.

Figure 2– SNMP Manager-Agent Model

The manager and agent use a Management Information Base (MIB) and a relatively small set of commands to exchange information. The MIB is organized in a tree structure with individual variables, such as point status or description, being represented as leaves on the branches. A long numeric tag or object identifier (OID) is used to distinguish each variable uniquely in the MIB and in SNMP messages.

SNMP uses five basic messages (GET, GET-NEXT, GET-RESPONSE, SET, and TRAP) to communicate between the manager and the agent. The GET and GET-NEXT messages allow the manager to request information for a specific variable. The agent, upon receiving a GET or GET-NEXT message, will issue a GET-RESPONSE message to the manager with either the information requested or an error indication as to why the request cannot be processed. A SET message allows the manager to request a change be made to the value of a specific variable in the case of an alarm remote that will operate a relay. The agent will then respond with a GET-RESPONSE message indicating the change has been made or an error indication as to why the change cannot be made. The TRAP message allows the agent to spontaneously inform the manager of an ‘important’ event.

As you can see, most of the messages (GET, GET-NEXT, and SET) are only issued by the SNMP manager. Because the TRAP message is the only message capable of being initiated by an agent. This notifies the SNMP manager as soon as an alarm condition occurs, instead of waiting for the SNMP manager to ask.

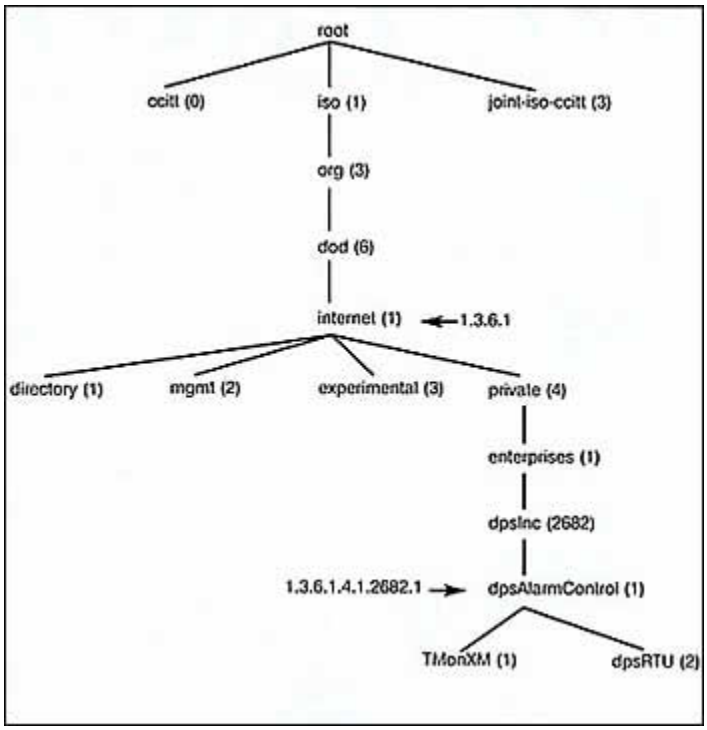
The small number of commands used is only one of the reasons SNMP is "simple." The other simplifying factor is its reliance on an unsupervised or connectionless communication link. This simplicity has led directly to its widespread use, specifically in the Internet Network Management Framework. Within this framework, it is considered ‘robust’ because of the independence of the managers from the agents,

e.g. if an agent fails, the manager will continue to function, or vice versa. The unsupervised communication link does however create some interesting issues for network alarm monitoring.

Each SNMP element manages specific objects with each object having specific characteristics. Each object / characteristic has a unique object identifier (OID) consisting of numbers separated by decimal points (i.e., 1.3.6.1.4.1.2682.1). These object identifiers naturally form a tree as shown below. The MIB associates each OID with a readable label (i.e., dpsRTUAState) and various other parameters related to the object. The MIB then serves as a data dictionary or code book that is used to assemble and interpret SNMP messages.

When an SNMP manager wants to know the value of an object / characteristic, such as the state of an alarm point, the system name, or the element uptime, it will assemble a GET packet that includes the OID for each object / characteristic of interest. The element receives the request and looks up each OID in its code book (MIB). If the OID is found (the object is managed by the element), a response packet is assembled and sent with the current value of the object / characteristic included. If the OID is not found, a special error response is sent that identifies the unmanaged object.

When an element sends a TRAP packet, it can include OID and value information (bindings) to clarify the event. Well-designed SNMP managers can use the bindings to correlate and manage the events. SNMP managers will also generally display the readable labels to facilitate user understanding and decision-making.



The branch of the MIB object identifier tree.

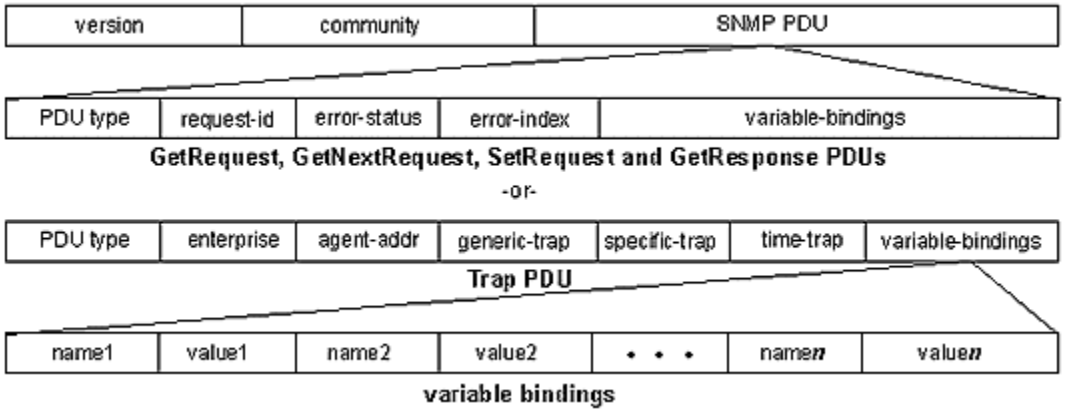
Figure 3– MIB Tree

The SNMP packets contain header, data and checksum bytes. SNMP is also packet oriented with the following SNMP v1 packets (Protocol Data Units or PDUs) used to communicate:

1. Get
2. GetNext
3. Set
4. Trap

The manager sends a Get or GetNext to read a variable or variables and the agent's response contains the requested information if managed. The manager sends a Set to change a variable or variables and the agent's response confirms the change if allowed. The agent sends a Trap when a specific event occurs.

The image below shows the packet formats. Each variable binding contains an identifier, a type and a value (if a Set or response). The agent checks each identifier against its MIB to determine whether the object is managed and changeable (if processing a Set). The manager uses its MIB to display the readable name of the variable and sometimes interpret its value.



SNMP Packet Formats

Figure 4– SNMP Packet Format

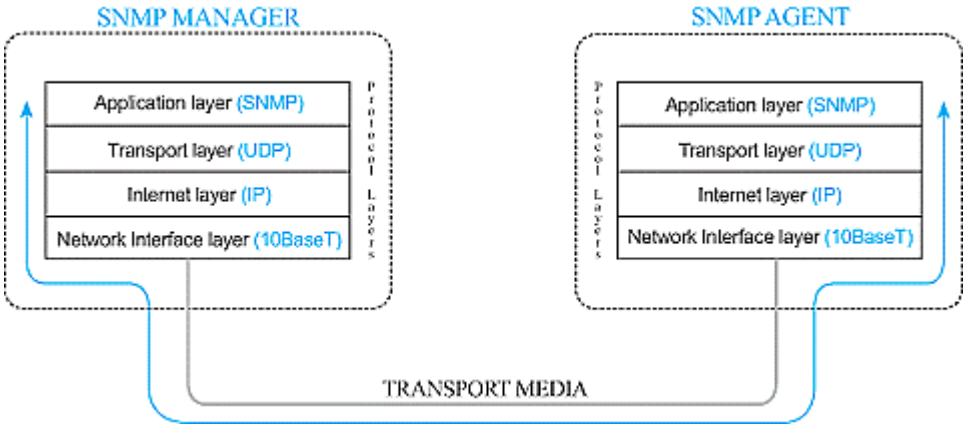
We continue to examine the Simple Network Management Protocol (SNMP) focusing specifically on the layered communication model used to exchange information. However an SNMP message is not sent by itself. It is wrapped in the User Datagram Protocol (UDP), which in turn is wrapped in the Internet Protocol (IP). These are commonly referred to as layers and are based on a four-layer model.

The SNMP resides in what is called the Application layer, UDP resides in the Transport layer and IP resides in the Network or Internet layer (somewhat obvious). The fourth layer is the Network Interface layer or Data Link Layer where the assembled packet is actually interfaced to some kind of transport media (i.e., twisted pair copper, RG58 co-axial or fiber).

Traversing the Layers

To illustrate the function of this layered model, let's look at a single SNMP GET request from the agent's perspective. The SNMP manager wants to know what the Agent's System Name is and prepares a GET message for the appropriate OID. It then passes the message to the UDP layer. The UDP layer adds a data block that identifies the manager port to which the response packet should be sent and the port on which it expects the SNMP agent to be listening for messages. The packet thus formed is then passed to the IP layer. Here a data block containing the IP and Media Access addresses of the manager and the agent is added before the entire assembled packet gets passed to the Network Interface layer. The Network Interface layer verifies media access and availability and places the packet on the media for transport.

After working its way across bridges and through routers (the modern equivalent of over the rivers and through the woods) based on the IP information, the packet finally arrives at the agent. Here it passes through the same four layers in exactly the opposite order as it did at the manager. First, it is pulled off the media by the Network Interface layer. After confirming that the packet is intact and valid, the Network Interface layer simply passes it to the IP layer. The IP layer verifies the Media Access and IP address and passes it on to the UDP layer where the target port is checked for connected applications. If an application is listening at the target port, the packet is passed to the Application layer. If the listening application is the SNMP agent, the GET request is processed as we have discussed in previous articles. The agent response then follows the identical path in reverse to reach the manager.



An SNMP message passes through the protocol layers at both the manager and the agent. Each layer addresses a specific communication task.

Figure 5– SNMP Packet Flow Path

Since its creation in 1988 as a short-term solution to manage elements in the growing Internet and other attached networks, SNMP has achieved widespread acceptance. SNMP was derived from its predecessor SGMP (Simple Gateway Management Protocol) and was intended to be replaced by a solution based on the CMIS/CMIP (Common Management Information Service/Protocol) architecture. This long-term solution, however, never received the widespread acceptance of SNMP.

MIB

Every SNMP manageable device contains a Management Information Base (MIB) that stores management information for that device. The MIB consists of a collection of managed objects.

A specific instance of a managed object is known as a MIB *variable* or *object instance*. Objects can be one of two types; *scalar* or *columnar*. A scalar object has only one instance. A columnar object has potentially multiple instances.

You specify an object instance (MIB variable) using an object identifier - **oid**. The oid consists of a *base MIB object* with the *instance* appended (see below for examples).

Scalar Objects
Consider the 'sysName' MIB object (1.3.6.1.2.1.1.5). This is the name of a system, and thus only has a single value. To access scalar objects such as 'sysName', you specify the instance to be **zero**.

i.e. To access the 'sysName' object, you would specify the object identifier as 1.3.6.1.2.1.1.5.0

In the example above, the base MIB object is 1.3.6.1.2.1.1.5, the instance is 0.

Columnar Objects
Columnar objects can be thought of as tables. The base MIB object defines the row of the table. Each instance represents a given column. Thus the instance is effectively an index into a table. (The instance can be an integer or an object-id).

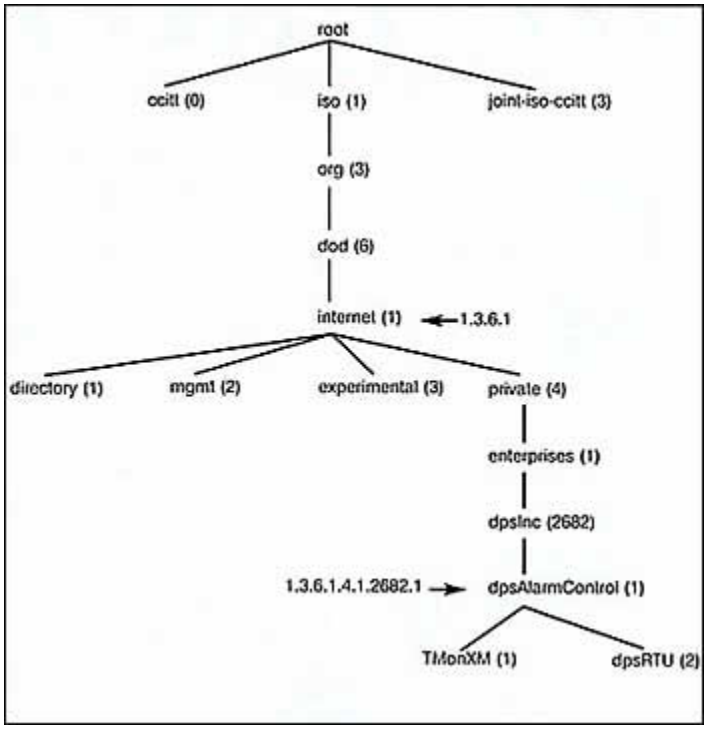
Consider the 'ifDescr' MIB object (1.3.6.1.2.1.2.2.1.2). This is an entry in a table that contains information on the interfaces of a device. On a machine with 3 interfaces, there would be 3 instances (or columns) of the 'ifDescr' object. To access information on each interface, you specify the instance you wish to query.

i.e. To access the 'ifDescr' object for the first interface, you would specify the instance as 1. The object identifier would then be 1.3.6.1.2.1.2.2.1.2.1

In the example above, the base MIB object is 1.3.6.1.2.1.2.2.1.2, the instance is 1.

Note: The instance (or index) does not have to be a simple integer. The instance can itself be an object-id

Each SNMP element manages specific objects with each object having specific characteristics. Each object / characteristic has a unique object identifier (OID) consisting of numbers separated by decimal points (i.e., 1.3.6.1.4.1.2682.1). These object identifiers naturally form a tree as shown below. The MIB associates each OID with a readable label (i.e., dpsRTUAState) and various other parameters related to the object. The MIB then serves as a data dictionary or code book that is used to assemble and interpret SNMP messages.



The branch of the MIB object identifier tree.

Figure 6– MIB Organization

When an SNMP manager wants to know the value of an object / characteristic, such as the state of an alarm point, the system name, or the element uptime, it will assemble a GET packet that includes the OID for each object / characteristic of interest. The element receives the request and looks up each OID in its code book (MIB). If the OID is found (the object is managed by the element), a response packet is assembled and sent with the current value of the object / characteristic included. If the OID is not found, a special error response is sent that identifies the unmanaged object.

In a managed device, specialized low-impact software modules, called *agents*, access information about the device and make it available to the NMS. Managed devices maintain values for a number of variables and report those, as required, to the NMS. For example, an agent might report such data as the number of bytes and packets in and out of the device, or the number of broadcast messages sent and received. In the Internet Network Management Framework, each of these variables is referred to as a *managed object*. A managed object is anything that can be managed, anything that an agent can access and report back to the NMS. All managed objects are contained in the *Management Information Base (MIB)*, a database of the managed objects.

An NMS can control a managed device by sending a message to an agent of that managed device requiring the device to change the value of one or more of its variables. The managed devices can respond to commands such as **set** or **get** commands. The **set** commands are used by the NMS to control the device. The **get** commands are used by the NMS to monitor the device.

Instead of defining a large set of commands, SNMP places all operations in a *get-request*, *get-next-request*, *get-bulk-request*, and *set-request* format. For example, an SNMP manager can get a value from an SNMP agent or store a value in that SNMP agent. The SNMP manager can be part of a network management system (NMS), and the SNMP agent can reside on a networking device such as a router. If SNMP is configured on a router, the SNMP agent can respond to MIB-related queries being sent by the NMS.

SNMP Manager Operations	**Description**
get-request	Retrieve a value from a specific variable.
get-next-request	Retrieve the value following the named variable. Often used to retrieve variables from within a table[1].
get-response	The reply to a get-request, get-next-request, get-bulk-request, and set-request sent by an NMS.
get-bulk-request	Similar to get-next-request, but fill the get-response with up to max-repetition number of get-next interactions.
set-request	Store a value in a specific variable.
Trap	An unsolicited message sent by an SNMP agent to an SNMP manager indicating that some event has occurred.

With this operation, an SNMP manager does not need to know the exact variable name. A sequential search is performed to find the needed variable from within the MIB.

Table 1- SNMP Operations

Chapter 2 – Wireless Technologies and Devices

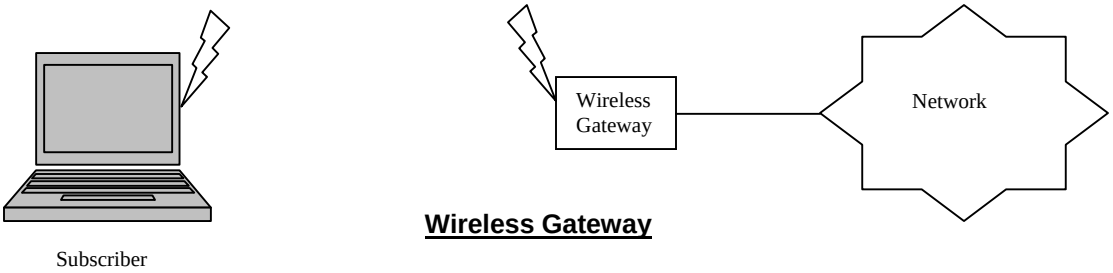


Figure 7– Wireless Device Connectivity

Wireless Technologies represent a rapidly emerging area of growth and importance for providing ubiquitous access to the network for all of the campus and corporate community. There is interest in creating mobile computing labs utilizing laptop computers equipped with wireless Ethernet cards. Recently, industry has made significant progress in resolving some constraints to the widespread adoption of wireless technologies. Some of the constraints have included disparate standards, low bandwidth, and high infrastructure and service cost. Wireless technologies can both support the institution/corporate mission and provide cost-effective solutions. Wireless is being adopted for many new applications: to connect computers, to allow remote monitoring and data acquisition, to provide access control and security, and to provide a solution for environments where wires may not be the best solution.

There are numerous applications for all the different wireless technologies. Applications of wireless technologies are divided into the following:

- Voice and messaging
- Hand-held and other Internet-enabled devices and
- Data Networking

This way of categorizing wireless technologies also includes their differences in cost models, bandwidth, coverage areas, etc.

Voice and Messaging

Cell phones, pagers, and commercial two-way business radios can provide voice and messaging services. These devices may be based on analog or digital standards that differ primarily in the way in which they process signals and encode information. The analog standard is the Advanced Mobile Phone Service (AMPS). Digital standards are Global System for Mobile Communications (GSM), Time Division Multiple Access (TDMA), or Code Division Multiple Access (CDMA).

Normally, devices operate within networks that provide metropolitan, statewide, or nationwide coverage. These large and costly networks are operated by carriers such as local phone companies, etc. and operate in different frequency bands. Throughput depends on the standard being used, but presently these networks operate at throughput rates up to 16 kilobits per second (Kbps). New digital standards, also referred to as "Third-Generation Services" or 3G will provide 30 times faster transfer rates and enhanced capabilities. Because of the many standards, there are

interoperability issues between networks, carriers, and devices. Generally, charges are based on per minute utilization or per number of messages.

Hand-held and Internet-enabled devices

Internet-enabled cell phones and Personal Digital Assistants (PDAs) have emerged as the newest products that can connect to the Internet across a digital wireless network. New protocols, such as Wireless Application Protocol (WAP), and new languages, such as WML (Wireless Markup Language) have been developed specifically for these devices to connect to the Internet. However, the majority of current Internet content is not optimized for these devices; presently, only email, stock quotes, news, messages, and simple transaction-oriented services are available. Other limitations include low bandwidth (less than 14 Kbps), low quality of service, high cost, the need for additional equipment, and high utilization of devices' battery power. Nevertheless, this type of wireless technology is growing rapidly with better and more interoperable products.

Data Networking

We differentiate between pure data applications in wireless local area networks (WLANs) and data, voice, and video converged in broadband wireless. We also briefly discuss Bluetooth, an emerging wireless technology.

Wireless Local Area Networks

Wireless Local Area Networks (WLAN) are implemented as an extension to wired LANs within a building and can provide the final few meters of connectivity between a wired network and the mobile user.

WLANs are based on the IEEE 802.11 standard. There are three physical layers for WLANs: two radio frequency specifications (RF - direct sequence and frequency hopping spread spectrum) and one infrared (IR). Most WLANs operate in the 2.4 GHz license-free frequency band and have throughput rates up to 2 Mbps. The new 802.11b standard is direct sequence only, and provides throughput rates up to 11 Mbps. Currently the predominant standard, it is widely supported by vendors such as Cisco, Lucent, Apple, etc. A new standard, 802.11a, will operate in the 5 GHz license-free frequency band and is expected to provide throughput rates up to 54 Mbps.

WLAN configurations vary from simple, independent, peer-to-peer connections between a set of PCs, to more complex, intra-building infrastructure networks. There are also point-to-point and point-to-multipoint wireless solutions. A point-to-point solution is used to bridge between two local area networks, and to provide an alternative to cable between two geographically distant locations (up to 30 miles). Point-to-multi-point solutions connect several, separate locations to one single location or building. Both point-to-point and point-to-multipoint can be based on the 802.11b standard or on more costly infrared-based solutions that can provide throughput rates up to 622 Mbps (OC-12 speed). In a typical WLAN infrastructure configuration, there are two basic components:

1. Access Points - An access point/base station connects to a LAN by means of Ethernet cable. Usually installed in the ceiling, access points receive, buffer, and transmit data between the WLAN and the wired network infrastructure. A single access point supports on hundreds of users and has a coverage varying from 20 meters in areas with obstacles (walls, stairways, elevators) and up to 100 meters in areas with clear line of sight. A building may require several access points to provide complete coverage and allow users to roam seamlessly between access points.

2. Wireless Client Adapter - A wireless adapter connects users via an access point to the rest of the LAN. A wireless adapter can be a PC card in a laptop, an ISA or PCI adapter in a desktop computer, or can be fully integrated within a handheld device.

Broadband Wireless

Broadband wireless (BW) is an emerging wireless technology that allows simultaneous wireless delivery of voice, data, and video. BW is considered a competing technology with Digital Subscriber Line (DSL). It is generally implemented in metropolitan areas and requires clear line of sight between the transmitter and the receiving end. BW comes in two flavors: Local multi-point distribution service (LMDS) and Multi-channel multi-point distribution service (MMDS). Both operate in FCC-licensed frequency bands.

LMDS is a high bandwidth wireless networking service in the 28-31 GHz range of the frequency spectrum and has sufficient bandwidth to broadcast all the channels of direct broadcast satellite TV, all of the local over-the-air channels, and high speed full duplex data service. Average distance between LMDS transmitters is approximately one mile apart.

MMDS operates at lower frequencies, in the 2 GHz licensed frequency bands. MMDS has wider coverage than LMDS, up to 35 miles, but has lower throughput rates. Companies such as Sprint and WorldCom own MMDS licenses in the majority of U.S. metropolitan areas. Broadband wireless still involves costly equipment and infrastructures. However, as it is more widely adopted, it is expected that the service cost will decrease.

Bluetooth

Bluetooth is a technology specification for small form factor, low-cost, short-range wireless links between mobile PCs, mobile phones, and other portable handheld devices, and connectivity to the Internet. The Bluetooth Special Interest Group (SIG) is driving development of the technology and bringing it to market and it includes promoter companies such as 3Com, Ericsson, IBM, Intel, Lucent, Motorola, Nokia, and over 1,800 Adopter/Associate member companies. Bluetooth covers a range of up to ten meters in the unlicensed 2.4GHz band. Because 802.11 WLANs also operate in the same band, there are interference issues to consider. Bluetooth technology and products started being available in 2001, but interoperability seems to be a big problem.

Standards

A major obstacle for deployment of wireless networks is the existence of multiple standards. As it was mentioned previously, there are analog and digital standards in wireless telephony. While GSM is the only widely supported standard in Europe and Asia, multiple standards are in use in the U.S. As a result, the U.S. has lagged in wireless networks deployment. Just recently, organizations have been formed to ensure network and device interoperability. For example, the adoption of the 802.11b standard has made wireless data networks one of the hottest newcomers in the current wireless market.

Chapter 3 – External Device Management Interfaces

Web Browser

The HTTP compliant Web browser is an NMS application and one of the most preferred device management interface sought. The Web browser displays device HTML based configuration, management screens to enable user configure, manage the remote managed device. The data sent by the browser/ user would be handled by HTTP Server and then passed on to the SNMP agent for further processing to GET or SET a managed devices configuration variable.

CLI, Command Line Interface

The Serial Interface and Telnet are also widely used in industry to run commands to configure and manage the managed device. They provide neatly organized commands to enable the user to issue different commands to the managed devices, to GET or SET a device configuration variable. Scripts could be written and run if the command shell supports it to do large amounts of configuration work.

MIB/ SNMP Browser

Is an NMS application that could configure, monitor and control managed device, a Wireless Gateway. A *managed device* is a network node that contains an SNMP agent and that resides on a managed network. Managed devices collect and store management information and make this information available to NMSs using SNMP. Managed devices, sometimes called network elements, can be gateways, routers and access servers, switches and bridges, hubs, computer hosts, or printers.

Chapter 4 – WMIT Architecture

Device management has emerged as a crucial requirement for todays networked devices, from carrier class switches and connected consumer electronics to industrial control and automotive telematics devices. The range of products requiring device management is extremely diverse, with different requirements existing within each market segment. WindManage enables one to implement remote management capabilities into their products for virtually any networked device.

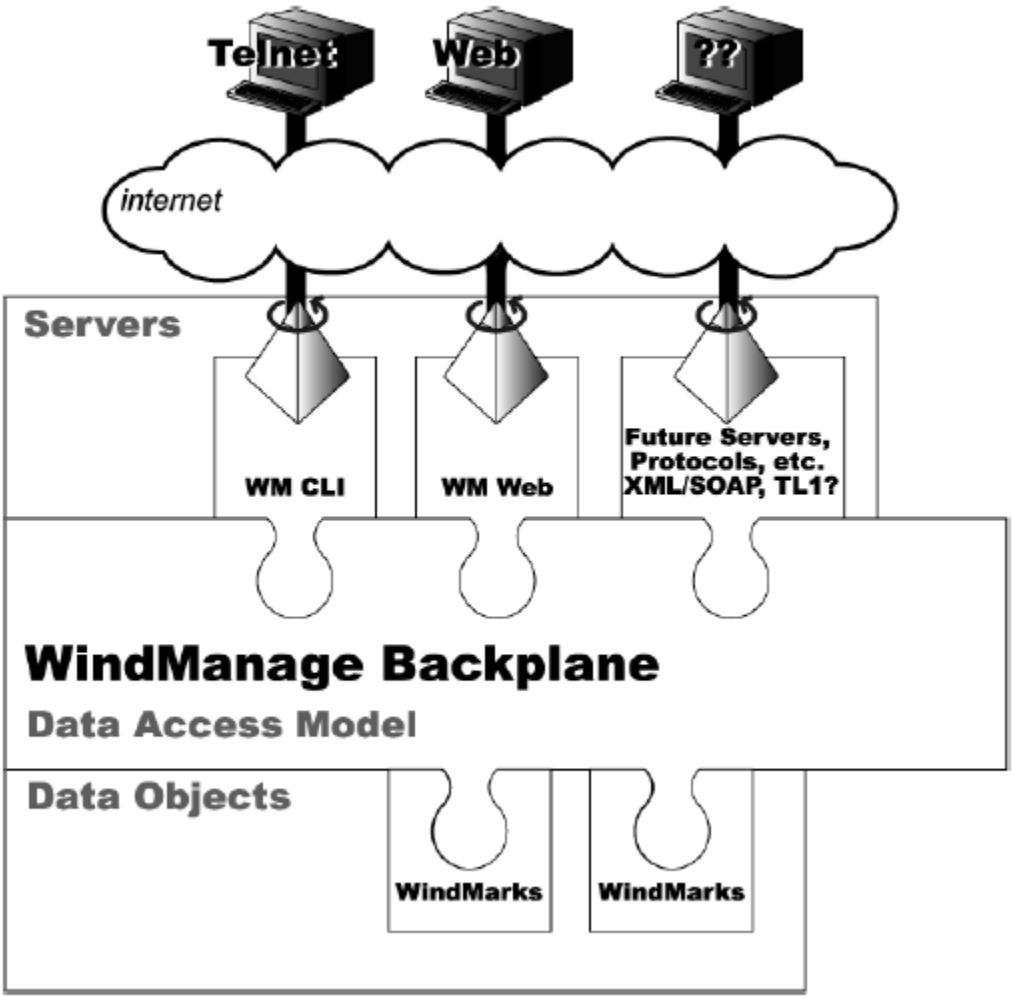


Figure 8– WMIT Architecture

As networks grow in size and complexity, the need for strong device management capabilities is essential. Building reliable and advanced device management capabilities into new products from scratch is both costly and time consuming. Development teams do not have the luxury of developing in-house expertise with technologies outside of their core business competencies, including device management. WindManage accelerates the development process by enabling manufacturers to build a variety of management layers quickly and to systematically upgrade and customize management interfaces across multiple product releases and product lines.

The WindManage open API will allow us to create custom interfaces for devices using Wind Rivers VxWorks and BSD/OS operating systems, and extend the WindManage backplane to new management standards. Icon Labs SNMP IQ Application Builder and Iconfidant SSH security software leverages WindManage and provides an optimum programming environment to create SNMP, Web and CLI-managed network devices.

WindManage integrates Wind Rivers EnvoyTM simple network management protocol (SNMP) implementation and RapidControlTM device management product lines into a single, unified device management framework. It supports Wind Rivers VxWorks, VxWorks AE and BSD/OS operating systems, Tornado integrated development environment and its extensive suite of network protocols. In addition, the suite is bound together by an IDE-based graphical interface and comprehensive tool set that speed development time through ease of use. Other key WindManage features include:

- Integrated device management technology Bundles external management interfaces including SNMP, Web server, command line interface (CLI), as well as custom interfaces, with an easy-to-use development framework.
- Unified Access and Security Provides a single set of data objects that normalize data access and security for multiple external management technologies simultaneously. Unified Tools - WindManage Integration Tool ties all of the device management products together and makes unified access a reality.
- Extensibility Enables the addition of custom and future management technologies through an open architecture.

Carrier class networking companies are able to use the WindManage solution to extend their currently implemented SNMP management solution to seamlessly add Web and CLI management capabilities, extending the wealth of SNMP management information bases (MIBs) already implemented to these other interfaces. Maintenance personnel of carrier class systems use Web and CLI as interfaces to interact with network equipment using a laptop computer. By using WindManage, maintenance personnel can use the same control parameters as the SNMP console in the carrier control and monitoring site without re-engineering more management interfaces.

In addition to carrier class equipment, networked consumer products such as printers and DSL routers can use WindManage to implement Web-based management systems. Consumer device manufacturers have discovered that a Web browser can be used as a universal management console, allowing graphical, easy-to-use management capabilities without having to build and deploy a proprietary application. Some high-end printers require SNMP management as well, making the WindManage unified access architecture indispensable. When new management technologies, such as XML-based management, become a requirement in printers, the WindManage open API will allow management capabilities to be extended quickly and easily, without re-engineering from the ground up.

The CLI commands and Web pages are built by WIND MANAGE., a software development toolkit that provides target runtime code and a graphical toolkit you can use to build management interfaces for your embedded equipment. WIND MANAGE provides tools that allow you to build Web-based and command line

management interfaces that both use a common backplane to access device data. Additionally, when you use WIND MANAGE MIBway, the Web-based and command line interfaces you design with WIND MANAGE can also access and modify SNMP management data.

Understanding WMIT

To understand what WIND MANAGE offers, first consider an example of what device management is. The following scenario highlights two examples of external device management implemented by WIND MANAGE. *Web-based* device management is one example of *external* device management, and *command line* device management is the other. Device management consists of configuring, controlling or monitoring the device. *External* device management indicates that the product's end user manages the device by a means external to the device. First let's clarify internal versus external management schemes a little bit further.

An internal management scheme is one where you physically connect to the device (perhaps with a serial line). Once connected —and only while connected— you can control, configure, or monitor the operation of that device. When the serial line is unplugged, you can no longer control the device or monitor its operation. This type of management has some limitations. One in particular is that you must be physically near the device. In some cases that may be a reasonable limitation, but in many cases, the person responsible for the device *cannot* be physically near the device.

An external management scheme is one that accommodates this limitation. As the name implies, the product's end user can manage the device externally via a network connection and some application layer running over the network. Web-based management is one particular type of external management. In Web-based management, the person responsible for the device does not need to physically connect to the device. Instead that person uses a Web browser to connect to an HTTP server embedded within the device. The person managing the device can be on another part of the local network or the internet.

Another example of external management is management from a command line interface. A command line interface is similar to a UNIX or DOS command shell. The product's end user enters control or configuration commands and parameters directly on a command line. In this example of external management, the end-user uses a telnet client to connect to a telnet server and command interpreter embedded within the device. As with Web-based management, the person managing the device can be on another part of the local network or the internet. Simple Network Management Protocol (SNMP) is another example of external device management. SNMP is a standard that defines network device management where a network administrator using a *network management station (NMS)* connects to a network management agent (SNMP agent) embedded within the device. It is similar to the Web and command line examples, however SNMP is defined by a set of Internet Engineering Task Force (IETF) standards. WIND MANAGE SNMP, a development toolkit for creating SNMP agents, is also part of the WIND MANAGE Collection.

WIND MANAGE provides the client developer with a unified solution for developing a device management application that leverages Web, command line, and SNMP interfaces. WIND MANAGE consists of target code and development

tools to create the embedded management server components, interfaces, and the *manageable elements* that your product's end users will configure and control.

WIND MANAGE provides code and tools to support the development of the following:
_ Command Line Interfaces, using WIND MANAGE CLI.
_ Web-based Interfaces, using WIND MANAGE WEB.
_ Web or CLI interfaces that can execute SNMP operations on the embedded SNMP agent, using WIND MANAGE MIBway.

Command Line Interfaces

Virtually every piece of commercial embedded network equipment provides a management interface via typed commands entered in an ordinary Telnet shell or serial application like HyperTerminal. This is a Command Line Interface (CLI). Despite this ubiquity, the CLI has often been one of the most poorly engineered aspects of a given device architecture. This poor engineering is often due to the perceived lack of importance. Commands are often designed and implemented in a quick and dirty fashion merely to test a newly developed feature. It is also due to real shipping pressures, because the CLI is not the purpose for the device—it's merely a window into its functioning. In short, CLIs are rarely developed with sound engineering principles. They often lack an architecture that is both efficient at runtime, yet easy to modify and extend, and they are almost never designed in the context of other management paradigms, and how they might interrelate with them.

WIND MANAGE CLI is one part of a family of device management products that are designed to work together through a common device data access model—the backplane. WIND MANAGE CLI provides you with a set of tools to quickly define a robust and extensible CLI into your hardware, so that your customers can manage the device from a simple Telnet shell or serial application.

WIND MANAGE CLI consists of a body of ANSI C source code that implements the CLI engine, and a set of development tools that you use to configure the CLI engine code for your hardware, and build the custom management command tree for your device.

Web Interfaces

Web-based management has some key advantages over other common approaches to device management. In a web-based scheme, the only real requirement to remotely manage the device is a standards-compliant Web browser. Given that browsers are installed on nearly every PC or workstation, virtually any networked PC or workstation can remotely manage your device.

Another key advantage with Web-based management is how the user experience compares with other management schemes. In the example where the person responsible for the device needs to physically connect via a serial cable, device configuration probably consists of entering commands via a console. Again, this is completely valid, and for some people quite useful. It generally requires that the operator has knowledge of the commands and parameters, and is comfortable with managing the device via console. This might be a reasonable assumption for an IT operator configuring an interior router. But is probably an *unreasonable*

assumption for an office manager installing a small office network printer. With a Web-based device management scheme, you can create a richer, more intuitive interface than a command line interface allows. Then the office manager can point and click through the configuration of the printer's network setup without worrying about command usage and syntax. Your screens (Web pages) could include recommended procedures, tips, or other types of interactive help that would be difficult, if not impossible with a command line interface.

All of this is not meant to imply that command line interfaces are not useful (they are tremendously useful to those who are used to them), it just means that you now have the ability to provide a simpler, more intuitive interface for your device for those users who can benefit from it—just as GUIs have for PCs and workstations.

SNMP Interfaces

An Simple Network Management Protocol (SNMP) interface involves a network administrator using a network management station (NMS) to connect to an SNMP agent embedded within the device. SNMP is a specific management approach governed by a set of relevant IETF standards. Please see the *WIND MANAGE SNMP Programmer's Guide* for more information about SNMP, building SNMP agents, and relevant IETF standards.

About SNMP and MIB Inheritance

SNMP is a management paradigm for system administrators to keep track of network devices, the performance of the network, and to diagnose and correct network problems. Around since 1990, SNMP was designed by the Internet Advisory Board (IAB) to be a standardized protocol for network device management and to provide inter-operability, so one vendor's devices could be connected with another vendor's devices.

Although SNMP was initially intended only as a placeholder until a more robust protocol could be devised, SNMP has been quickly and widely adopted by the data communications industry—virtually 100% of shipping network equipment is manageable via SNMP. The rapid adoption of SNMP in the telecommunications industry ensures that SNMP will continue to be a default management standard in all packet-based networks. In the IAB Request For Comments process which allows the standards to evolve as the Internet evolves, the IAB recommended that all IP and TCP implementations be network manageable with SNMP being the preferred full internet standard (RFC 1157).

MIBway: Executing SNMP Commands from WIND MANAGE CLI and WEB

When you create an SNMP agent using WIND MANAGE SNMP *and* a Web or CLI management interface using WIND MANAGE, you can use WIND MANAGE MIBway as a bridge from the Web/CLI interface to the SNMP agent. Your product's end users can GET and SET SNMP managed objects from a Web browser/telnet session as if they were using an NMS.

The WIND MANAGE Backplane

The basic components of a WIND MANAGE application are one or more interface servers (the embedded HTTP server, or the embedded telnet server) and a WIND MANAGE backplane, which contains one or more repositories of WindMarks. The backplane also serves as a a data abstraction mechanism for WIND MANAGE WEB and WIND MANAGE CLI.

When an end-user initiates a request to GET (or SET) a WindMark value, the interface server (HTTP or telnet) passes the WindMark name (and new value in the case of a SET) to the backplane. The backplane then executes requested WindMark's handler. If the WindMark represents a scalar built-in datatype, the automatically generated *scalar handler* reads (or updates) the value. If the WindMark is a custom datatype or a member of a table, the backplane executes whatever functionality you have coded into the WindMark's handler.

Backplane Services

The backplane offers services beyond GET and SET requests with the data object. The backplane provides locking and transaction processing. Locking allows multiple clients access to the same object at the same time, but only one client can modify the information in the WindMark. Transaction processing is necessary so multiple SETs can be done simultaneously on a group of objects, such as a row in a table, and all attempted SETs are backed out of if one of the SETs is not possible. The backplane provides data validation and data conversion of WindMarks in a strongly typed manner as well as data view and data modifiability security. The backplane provides the following services:

_ *locking*
_ *transaction processing*
_ *data validation*
_ *data conversion*
_ *data object access control*
_ *language localization*

WIND MANAGE CLI Overview

You use the WIND MANAGE Integration Tool to create commands along with their parameters and handlers. In the WMIT these commands are displayed in a command tree in the project window. The command tree is a visual representation of the command hierarchy and relationships. Each command is shown as a node on the tree, command parameters, handler functions, and WindMarks associated with the command are displayed on the command tree also. The visual command tree lets you see the tree your users will have to navigate, and the tree structure prevents inconsistencies that might otherwise occur.

The WIND MANAGE Integration Tool (WMIT) is a GUI interface you can use to set preferences for your projects, create WindMarks, and create commands and all of their properties, parameters, and handlers. You can import information (such as commands and WindMarks) from your other WIND MANAGE projects.

The WMIT generates files that contain all of this project information; these files are listed below.

With the WIND MANAGE Integration Tool (WMIT) you can:
_ Quickly define some standard commands that come from pre-existing CLI models (such as the Unix or Cisco-like command set, for example)
_ Specify and define new CLI commands (that might be specific to your embedded system)
_ Specify and define WindMarks
_ Link the CLI commands with WindMarks in a visual and intuitive manner.
In addition, the code generation components of the WIND MANAGE Integration Tool code are written in either C or C++, in order to facilitate development of console tools usable in environments other than Win32. Specifically, Unix console tools have been developed to provide similar functionality to the WMIT.

The WIND MANAGE Integration Tool allows you to quickly create a CLI command set by creating the tree of command nodes. Each node, in turn, can be edited to add parameters and handler functions (for specifying unique functions). In any command the information that follows the keyword may be either parameters and their values, or more commands and those command's parameters and values.

WIND MANAGE CLI Features

The CLI engine serves as a command parsing module that reads in the message received from the client, performs simple parsing to tokenize the message, checks for valid data and/or error conditions, and determines the appropriate command handler to execute. Core keywords, such as **no** (which negates the sense of some commands), are checked for and handled here.

The CLI engine serves the following functions:

- Handling the initial authentication and authorization, using a standard *name:password* authentication mechanism. *Although* this process is handled by the CLI engine, you must write the code that performs the authentication and manages authorized users.
- A command interpreter (acts as a UNIX-like shell) with the following features:
- Command completion—typing TAB at the command prompt will initiate a search of the command tree for the first unambiguous match of a command to the text entered by the user to that point. For example, if the user enters **res<TAB>** and the only command that starts with **res** is reset, the command tree will return reset and the CLI engine will display that for the user.
- History—the CLI engine will record the last *x* commands issued by the user, where *x* is configurable by the user in **wm_options.h** (we'll cover **wm_options.h** in detail later on).
- Help—typing **help**, or **?** at any point will retrieve help info from the command tree based on whatever portion of a command the user has typed to that point.
- Source files—the CLI engine reads in a command, tokenizes it, assigns each token to a command node or parameter, and validates that the parameter information is correct.
- Intermediate modes support—intermediate mode temporarily moves the root node closer to a particular leaf node on the tree . The shell remains in intermediate mode until the user explicitly enters an exit commands such as **CTRL** + **Z**.
- Interfaces with the backplane to GET and SET device management data.

Client Interfaces

WIND MANAGE CLI implements the following client interfaces:

Packet Interface

This module provides support for the Telnet Protocol (RFCs 854-861), as well as some of the more modern options implemented in client shells (such as RFC 1416 and RFC 1572). It also supports native shell interfaces such as the one supplied with the VxWorks operating system.

Serial Interface

This block executes any code that is necessary to provide standard RS-232 access to the device.

Access Control

WindMarks, commands, and parameters can each have their own access control restrictions. Access is allowed or denied based on access level. In order to have access, the user must have an access level greater or equal to that of the resource being requested.

When you apply access control to commands and parameters, those commands and parameters will only be accessible to users with am equal or higher access level.

When you apply access control to WindMarks (and the command or parameter referencing the WindMark is not access restricted), the WindMark read or write will only succeed for users with an equal or grater access level.

WIND MANAGE Integration Tool Features

Command Node Editor

From this editor the user can create and edit nodes in the command tree. The user is able to add parameters and handler functions, modify help strings, set security levels, and other assorted items. You can enter into both the Handler Function Editor and the Parameter Editor from this component.

Handler Function Editor

After you create command nodes and generate the project files, you use this tool to edit the generated skeleton handler functions for your command nodes. You specify indicate which parameters are required, which are optional, whether the function is designed to handle *no* forms of the command, and so forth.

Parameter Editor

You use this tool to edit parameters for command nodes. You specify the name and variable type of the parameter WindMark to GET/SET, and so forth. You can also create custom parameter types.

Quick Command Create

This feature allows you to quickly specify a command, its parameter(s), and its

help string. The export/import feature lets you export command code (such as all prompt strings or all help strings), edit them in a text file, and then import them back into the project.

WIND MANAGE WEB Overview

WIND MANAGE WEB addresses some key difficulties of adding Web-based management to your device:

- Porting or writing a web server for *your* system, and
- Linking the manageable elements of your device to the HTML based management interface.
- Unifying Web, CLI, and SNMP based management on the device.

Linking the HTML Code to the Device's Manageable Elements

First of all, what is a manageable element? It is a generic term for any variable (or managed object in SNMP terms) resident in the target device that contains information about the state of the device. In some cases these may be read-only elements, like a system up-time variable. In other cases they may be configurable read-write elements, such as a network name or IP address.

In WIND MANAGE, manageable elements in the device are represented by WindMarks. A WindMark is a special piece of markup you put in your HTML code that is mapped to a handler function in the backplane. The WindMark's handler function actually reads the data from (or writes data to) the device-resident variable.

So when an HTTP request is parsed for WindMarks —and WindMarks are found— the backplane calls the registered handler for each WindMark, and whatever functionality you have written into each handler is executed.

The importance of this is that your interface code —the HTML— is separated from the device management executable code. Keeping the HTML interface separate from the handler implementation makes both the HTML and the handler code easier to write, understand, and maintain than if you used some combination of HTML and handler code written together. Using the WIND MANAGE model, your Web specialist can design the HTML interface without requiring knowledge of the embedded handler code, and your embedded expert can write the custom handler functions without needing to write any HTML. All each needs to know about what the other is doing is the names of WindMarks.

NOTE: Samples of the WindMark error and post reply routines are provided in *installDir* **/tutorials/src/wrn/wm/wmw/tutorial***[06-09]***/wmwUsrFuncs.c/h wmwUsrFuncs.c/h**. You *must* provide your own custom versions of these routines so that you can customize error handling and HTTP post reply functionality. In addition to customizing the functionality, you will also want to make the HTML displayed by these routines (if any) consistent with the rest of your HTML interface.

WIND MANAGE WEB Features

You can deploy WIND MANAGE WEB as the *only* management interface for a device, or as part of a management application that also includes a command line interface. If you use WIND MANAGE WEB as the only interface, the WIND MANAGE WEB application consists of the following components:

_ Embedded HTTP server

The server is a scalable embedded HTTP server. It has a modular architecture which allows you to include or exclude modules—called *Request Processing Modules*, or *RPMs*—so that you can build in all the features you need, and exclude those you do *not* need to keep the compiled application size as small as possible. You can also create *your own* request processing modules to extend the functionality of the server.

- WIND MANAGE Backplane

The backplane is the component that all of the WIND MANAGE blades work *through* to access manageable elements in the device.

- WindMark Resource Database

A sub-component of the backplane that contains all of the WindMarks you define for a project, along with their associated data, such as handler functions and access control information.

- The HTML-based management interface

This is the collection of files—HTML files, scripts, graphic files, and so on—that make up the visual management interface of the device; the personality of the device you might say.

WIND MANAGE MIBway Overview

The WIND MANAGE suite of products, designed around the WIND MANAGE backplane is designed to be flexible enough to inherit objects from other management schemes, such as SNMP, so their objects may be used by the management interfaces that you build with WIND MANAGE.

If you have a device managed with a Simple Network Management Protocol (SNMP) interface using WIND MANAGE SNMP, WIND MANAGE MIBway allows you to add a Web-based management interface, or a Command Line Interface (CLI). Developing a Management Information Base (MIB) for a device can take hundreds or even thousands of engineering hours to create, test and refine. Rather than discard that valuable engineering, MIBway inherits the MIB and converts if for use with the WIND MANAGE backplane.

The WIND MANAGE backplane is core to the WIND MANAGE suite of interface projects, it registers and de-registers all management objects for a device. WIND MANAGE MIBway provides routines so that the WIND MANAGE backplane can access and use all of the SNMP MIB objects on a device. This MIB inheritance allows the current management interfaces provided by the WIND MANAGE suite of product, and all future management interfaces that make use of the WIND MANAGE backplane.

In the WIND MANAGE architecture, WIND MANAGE MIBway is a specialized handler to access existing MIB objects. WIND MANAGE MIBway is intended to inherit SNMP data and structures for use with the WIND MANAGE suite of interface products. In the backplane-centric view, MIBway is a bridge to the SNMP agent. The bridge and the SNMP agent would be considered a producer to the backplane. The interface servers (Web or CLI) are considered consumers in that they change the values of objects, but do not create, destroy or otherwise modify the accessibility of the objects.

SNMP Overview

SNMP consists of three basic components — network management stations, the network devices which need to be managed, and the protocol itself, which is the means for the two to communicate with each other. Each device contains a

standardized information tree, so each type of device and their variables may be accessed by other non–proprietary network management consoles. This standardization allows different vendor's devices to be controlled by a single management console. Each managed device has an SNMP agent which reads information from —or writes information to —the network device. The function of the agent is to service the standardized information tree, known as the Management Information Base (MIB). The information from the MIB is passed between the network management console and the SNMP agent in packets called Protocol Data Units (PDUs).

SNMP only has four basic operations:

- Set (or change) a managed object.
- Get (or receive the value from) a managed object.
- Get Next, Get the next managed object in the database of managed objects, this operation allows you to traverse the database.
- Send an asynchronous notification from some sort of exceptional event that occurs on the device.

The information in the managed objects — variables — is altered or viewed to configure or monitor the device. The kind of data that can be used to manage a device can be limited to a small number of data types. SNMP has but four basic primitive types:

- INTEGER — a 32 bit 2's complement integer. (Range from –231 to 231 – 1)
- OCTET STRING — eight bits for each octet (Range from 0 to 255)
- OBJECT IDENTIFIER — a sequence of integers that are used to identify objects and groups of objects in a hierarchical data structure known as a MIB tree.
- NULL

Other data types are based on these data types with what are known as textual conventions.

From these concepts SNMP is based — four basic operations and four basic data types, a network management station (NMS) and managed devices with an agent that accesses management data.

The agent is basically a server to the MIB on the device. It has a server input loop which receives the PDU which is read into a buffer and called an SNMP packet. The packet is decoded into a structure, which is then passed around. The request packets are GETs, GETNEXTs, and SETs. The agent processes the request, finds the appropriate MIB object, and the MIB object's appropriate function. Once the

NOTE: The primitive types are shown in all caps because that is a formatting convention for ASN.1.

SNMP Agent – WIND MANAGE MIBway Interaction

In order to access the MIB objects, WIND MANAGE MIBway acts somewhat like an NMS. If a GET is done on a MIB object, the WMB identifies the object as needing the built–in WIND MANAGE MIBway handler to process the WindMark, rather than generate a handler for the customer–developer to modify (**wmmHandler** in the code and the WMIT GUI.)

However, rather than pass a packet which must be decoded, WIND MANAGE MIBway passes and receives the C structure directly to the WIND MANAGE

SNMP agent.

WIND MANAGE MIBway supports SNMP textual conventions. They do not have to be custom created in the WMIT, as the customer–developer would do for specialized custom types.

The WMB service of locking is unnecessary, since the SNMP agent will handle locking for its objects. The SNMP also conducts its own transaction processing.

In order for a network device to be manageable by SNMP, it must have an SNMP agent that is capable of receiving, parsing, building, and sending SNMP PDUs, and it must have a Management Information Base. The Management Information Base, in turn, must have a significant number of MIB objects, with their corresponding GET and SET routines, in order for the console to monitor and control the device in a meaningful fashion.

However, the core of SNMP, the design, definition, and description of a device's MIB objects, must be done for each device, in almost a unique fashion. The reason for this is that the MIB objects need to access real data, which means there must be GET and SET routines written that access real variables used in the device's code. These GET and SET routines for *each* MIB variable are (relatively) unique to each device code line.

For example, the WIND MANAGE SNMP agent requires a variation of the following system call to construct the MIB Object for **sysName**:
LEAF_NT_ASYS(l_sysName, 0, "sysName", VT_STRING,
READ_WRITE_ACCESS,
SCALAR, system_test, system_get, system_set, system_next, ((char *)0), 0x0000, 0xFF, 0xFF);

The developers at the network equipment vendor must *themselves* perform the task of writing the GET and SET routines (the **system_get( )** and **system_set( )** functions shown above) WIND MANAGE SNMP makes this process much easier, generating code that requires only minimal alteration for each object. However, the task is not limited to 5 or 10 MIB variables — network devices have, at an absolute minimum, the roughly 200 MIB objects described in MIB II.

So while SNMP is popular, and will remain the de-facto management paradigm for all packet-based networks into the foreseeable future, there is a non-trivial amount of engineering resources that need to go into developing and maintaining a viable SNMP implementation on a device.

The WIND MANAGE Handler (wmmHandler)

Unlike the generated or custom WindMark handlers that you generate, the backplane accesses MIBway WindMarks (which you map to SNMP MIB objects) by using a built a built–in handler called the **wmmHandler** (or the MIBway handler). The object action functions such as GET, SET, GETNEXT are handled by the WIND MANAGE MIBway handler. When attempting an SNMP transaction, the **wmmHandler** will check the SNMP error log for existing errors before the transaction and then check the log again after the execution of the **wmmHandler**.

Operation with Scalar MIB Objects

When the backplane finds an entry for **sysName**, it passes it off to the WIND

MANAGE MIBway handler. The OID of the WindMark is tied to the WindMark object in the WMB. WIND MANAGE MIBway formats the request for **sysName** and makes a call to the GET function. Once retrieved, the relevant data is sent back to MIBway, converted, and available for whichever interface server made the request.

Operation with Tabular MIB Objects

Since most SNMP MIB objects are in table format, this support is undoubtedly the most valuable and sophisticated feature of MIBway.

As shown in Figure 2-6, the Structure of Management Information (SMI) consists of a lot of groups and subgroups. However, as you wend your way through the SMI hierarchy, eventually you wind up at a leaf node. Leaf nodes are the members of the SMI that have actual GET and SET routines associated with them. Leaf nodes are where the data is stored in the structure.

The variable, **sysName**, is a Leaf Node, but it represents a scalar object. **sysName** is an object that has one and only one instance —hence the *instance* variable that was appended to the numerical specification of **sysName**, 1.3.6.1.2.1.1.5, was 0, for a final, unique object identifier (OID) of 1.3.6.1.2.1.1.5.0. Leaf nodes can represent table objects as well —objects that have more than one instance. In such cases, WIND MANAGE MIBway handles all the table instancing information.

Chapter 5 – Software Design

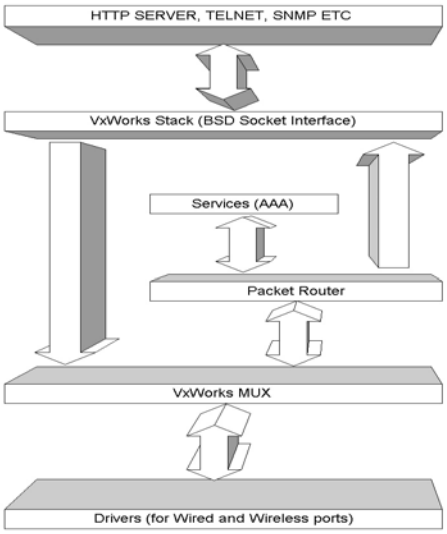


Figure 9 - Gateway Architecture

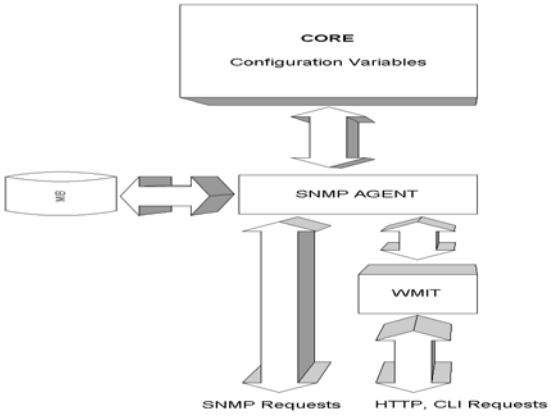


Figure 10 - SNMP Functional Overview

Gateway Architecture

Please refer to the Fig. 9., we have some modules above the TCP/IP stack that do not process the packets. User processes or applications like HTTP Server, Telnet Server falls in that category. They send/ receive packets using the TCP/IP stack.

The vxWorks MUX acts as a unifying interface between the TCP/IP Stack and the drivers. Packets travel from hardware network cards to the driver; driver sends them on to the MUX; MUX then processes the packets in accordance with a number of rules and hooks registered (calling muxBind()). For example, one could register a hook to capture all IP packets. By virtue of described hook mechanism, packets get delivered to the TCP/IP stack. Input and output hooks are registered with the MUX –

input hooks to receive incoming packets before they are received by the stack, and output hooks in order to receive packets sent by the stack before they leave the gateway.

The packet router is closely tied to services (AAA, DAT etc) and is responsible for routing packets among the various processing modules. It receives all packets from the MUX and either handles them or drops them. Packet router needs to make different decisions for different processing paths, depending on where it received the packet. Upon receiving a packet the packet router preprocesses the packet to extract relevant fields, build the priority list of modules to be executed and set the destination VIF to its default value. This list is determined by the direction of the packet (processing path) as well as by inspection of the packet header and the configuration of the system. The return code from a processing service will indicate whether to continue processing the packet or to drop it. If a packet is to be dropped, the remaining scheduled services are ignored. When the packet router has no more services scheduled to run (and the packet has not been dropped) it delivers the packet to the destination VIF to actually be sent.

Services go hand-in-hand with the Packet Router. The packet router will process a packet by traversing a priority list of services to be executed. As the packet router steps along the list of services, it executes the highest priority service during each iteration. A service may add a service to the list to be executed via the packet router. Each processing method is passed an object that describes the state of the packet as it is initially received. It is a job of the service to return information to the caller (packet router), indicating whether the packet needs to be dropped or passed on to the next service in the list.

As discussed above, our code sends packets to stack or network interfaces, it does not call appropriate routines directly. Instead, the VIF module is used. VIF abstracts out the type of interface (e.g. Ethernet versus Frame Relay) as well as its mapping to a physical interface. Multiple virtual interfaces may map to one physical interface and/or a virtual interface may map to multiple physical interface. We accomplish this via the Switch module. Virtual interfaces are responsible for placing the proper source and destination link layer addresses on the packet and delivering the packet to an actual device.

SNMP Architecture

The SNMP requests from Browser and CLI would pass through all the relevant layers to reach WMIT Backplane or MIBWay. The HTTP Server pre-processes the HTTP requests and the WMIT shell parses the CLI commands. They both call the MIBWay functions to perform SNMP GET or SET operation through SNMP Stack and SNMP Agent. The SNMP requests from the SNMP Manager reaches the SNMP Agent directly. This request doesn't pass through the WMIT Backplane.

The SNMP Agent, upon receiving the request would process the same to send back the results. During a typical Set operation, the data to be saved would come as part of SNMP request/PDU, along with the OID. During a typical Get operation, the OID for which the data has to be sent back would come along with the SNMP request/PDU. The SNMP agent uses the stubs or hooks in the Core to perform the data saving and retrieving operations.

In case of table GET/ SET operation, the row instance would be passed to the SNMP Agent. The same would be used to fetch appropriate row instance to carry out a SET or GET.

In the core, all the MIB Objects are defined as configurable variables or managed device attributes. There will be one-to-one mapping between them. The core provides a set of hook or stub functions to Get/ Set these configuration variables.

The MIB when compiled using a MIB complier, generates part of the SNMP Agent code. Inserting calls to the appropriate stub functions defined in the core would complete this implementation of SNMP Agent.

Summary

Remote device management has emerged as a crucial requirement for todays networked devices. The range of products requiring device management is extremely diverse, with different requirements existing within each market segment. Remote management capabilities could be built into products for virtually any networked device using SNMP and allied technologies. The efficient remote device management implementation is necessary to enhance the MIB seamlessly in future.

Conclusions and Recommendations

The remote device management is about monitoring, managing and controlling a single managed device remotely. The same technology could be extended to monitor, manage and control multiple managed devices centrally. This new generation remote management software would enable the network administrator to centrally enforce access control rules over multiple managed devices.

References

1. Computer Networks
Andrew S. Tanenbaum, Pearson Education , Fourth Edition

2. windmanage_api_reference_4.1.pdf
Windmanage API reference, Copyright 2003 Wind River Systems, Inc.

3. windmanage_cli_web_mibway_programmers_guide_4.1.pdf
Windmange CLI, WEB and MIBWAY programmers guide, Copyright 2003 Wind River Systems, Inc.

4. windmanage_cli_web_mibway_supplement_4.1.pdf
CLI, WEB and MIBWAY supplement, Copyright 2003 Wind River Systems, Inc.

5. windweb_2.0_windmanage_web_4.2_migration_faq_draft.pdf
WINDWEB to WINDMANAGE migration related FAQ, Copyright 2003 Wind River Systems, Inc.

6. WM_WEB_Tutorial_4.0.pdf
WINDWEB tutorial, Copyright 2003 Wind River Systems, Inc.

7. WM_WEB_user_guide_4.0.pdf
WINDWEB user guide, Copyright 2003 Wind River Systems, Inc.

8. windmange_cli_tutorial_4.1.pdf
Windmanage CLI tutorial, Copyright 2003 Wind River Systems, Inc.

9. windmange_web_tutorial_4.1.pdf
Windmanage WEB tutorial, Copyright. 2003 Wind River Systems, Inc.

10. http://www.snmp.org
SNMP conference papers, research articles and standards

11. http://www.cisco.com/warp/public/535/3.html
SNMP research papers published by CISCO

12. http://www.nomadix.com/aghelp/ag2000w/wwhelp/wwhimpl/js/html/wwhelp.htm
Nomadix user guide and online help

Checklist of items for the Final Dissertation Report

1.	Is the report properly hard bound. Spiral bound reports are not acceptable.	Yes / No
2.	Is the Cover page in proper format as given in Annuexure A.	Yes / No
3.	Is the Title page (Inner cover page) in proper format?	Yes / No
4.	(a) Is the Certificate from the Supervisor in proper format? (b) Has it been signed by the Supervisor?	Yes / No Yes / No
5.	Is the Abstract included in the report properly written?	Yes / No
6.	Does the Report contain a summary of the literature survey?	Yes / No
7.	Does the Table of Contents include page numbers? (i). Are the Pages numbered properly? (ii). Are the Figures numbered properly? (iii). Are the Tables numbered properly? (iv). Are the Captions for the Figures and Tables proper? (v). Are the Appendices numbered properly?	Yes / No Yes / No Yes / No Yes / No Yes / No Yes / No
8.	Is the conclusion of the Report based on discussion of the work?	Yes / No
9.	Are References or Bibliography given in the Report? Have the References been cited inside the text of the Report? Is the citation of References in proper format?	Yes / No Yes / No Yes / No
10.	A Compact Disk (CD) containing the softcopy of the Final Report and a copy of the Final Seminar Presentation made to the Supervisor / Examiner (both preferably in PDF format only) has been placed in a protective jacket securely fastened to the inner back cover of the Final Report.	Yes / No

Declaration by Student:

I certify that I have properly verified all the items in this checklist and ensure that the report is in proper format as specified in the course handout.

Place: Bangalore

Date: 21 March 2006

_____

Signature of the Student

Name: Dinesh D N

ID No.: 2004HZ12158

I have duly verified all the items in this checklist and ensured that the report is in proper format.

Place: Bangalore

Date : 23 March 2006

_____

Signature of the Supervisor

Name: Baba Saheb