NET+OS with GNU Tools Programmer's Guide

Making DEVICE NETWORKING easy™

NET+Works with Green Hills Programmer's Guide

Operating system/version: 6.3
Part number/version: 90000725_B
Release date: March 2005
www.digi.com

©2006 Digi International Inc.

Printed in the United States of America. All rights reserved.

Digi, Digi International, the Digi logo, the Making Device Networking Easy logo, NetSilicon, a Digi International Company, NET+, NET+OS and NET+Works are trademarks or registered trademarks of Digi International, Inc. in the United States and other countries worldwide. All other trademarks are the property of their respective owners.

Information is this document is subject to change without notice and does not represent a committment on the part of Digi International.

Digi provides this document “as is,” without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of, fitness or merchantability for a particular purpose. Digi may make improvements and/or changes in this manual or in the product(s) and/or the program(s) described in this manual at any time.

This product could include technical inaccuracies or typographical errors. Changes are made periodically to the information herein; these changes may be incorporated in new editions of the publication.

Contents

Chapter 1: NET+Works Introduction 1

System components 2

NET+Works runtime software 2

HTML-to-C compiler 3

Advanced Web Server (AWS) PBuilder utility 3

Address Configuration Executive (ACE) 3

System requirements 4

Chapter 2: Using the HTML-to-C Compiler 5

Overview 6

Web content 6

Static and dynamic content and forms processing 7

Preparing to use the HTML-to-C compiler 10

How the HTML-to-C compiler works 10

Using the HTML-to-C compiler 12

Creating a new project 13

Removing obsolete data 14

Adding or removing source files 14

Specifying the location of files 15

Generating C source files 15

Setting or changing a project's home page 15

Editing URL files 16
Opening the url.c file 16
Adding and deleting URLs 17
Editing a URL 17
Setting the user and password of a URL 18

Chapter 3: Using the MIBMAN Utility 19

Overview 20
Terms and concepts 20
SNMP 20
Scalar MIB objects 21
MIB tables 21
Traps 22
Action routines 23
Implementing a MIB: an example 23
Converting an SNMP MIB into C code 23
Step 1: Using SMICng 23
Step 2: Using MIBMAN 24
Step 3: Final integration 36
Writing action routines 37
Action routines for scalar objects 37
Action routines for tables 42
SNMP OID and string index values 51

Chapter 4: Using the Advanced Web Server PBuilder Utility 53

Overview 54
The PBuilder utility 54
About the Advanced Web Server Toolkit documentation 55
Running the PBuilder utility 55
Linking the application with the PBuilder output files 57
security.c file 57
cgi.c and file.c files 57

Comment tags 57
Creating Web pages 58
AWS custom variables 58
Data types 60
Displaying variables 60
Changing variables 61
Security 63
Exceptional cases 64
Controlling the MAW module 64
Setting the semaphore timeout 64
Array subscripts 65
Error handling 66
Building the application 67
Phrase dictionaries and compression 67
Maintaining and modifying Web content 68
Sample applications 68

Chapter 5: Troubleshooting 69

Diagnosing errors 70
Diagnosing a fatal error 70
Diagnosing an unexpected exception 70
Reserializing a development board 71
Observing the LEDs 71
Preparing to reserialize 72
Assigning a MAC address to the NET+Works board 72
Restoring the contents of flash ROM 74
Step 1: Configure the development board and the MAJIC 74
Step 2: Building the bootloader 75
Step 3: Building the application image and starting naftpapp 75
Step 4: Sending rom.bin to the development board 75
Step 5: Verifying the boot ROM image on the development board 76

Using This Guide

Review this section for basic information about this guide, as well as for general support contact information.

About this guide

This guide describes NET+OS 6.3 with GNU Tools and how to use it as part of your development cycle. Part of the NET+Works integrated product family, NET+OS is a network software suite optimized for the NET+ARM.

Software release

This guide supports NET+OS 6.3. By default, this software is installed in the `C:/NETOS63_GNU/` directory.

Who should read this guide

This guide is for software engineers and others who use NET+Works for NET+OS.

To complete the tasks described in this guide, you must:

- Be familiar with installing and configuring software.
- Have sufficient user privileges to do these tasks.
- Be familiar with network software and development board systems.

Conventions used in this guide

This table describes the typographic conventions used in this guide:

This convention	Is used for
italic type	Emphasis, new terms, variables, and document titles.
bold, sans serif type	Menu commands, dialog box components, and other items that appear on-screen.
Select **menu name→ menu selection name**	Menu commands. The first word is the menu name; the words that follow are menu selections.
`monospaced type`	File names, pathnames, and code examples.

What's in this guide

This table shows where you can find information this guide:

To read about	See
The NET+Works components	Chapter 1, "NET+Works Introduction"
Converting HTML and related Web page files to C code	Chapter 2, "Using the HTML-to-C Compiler"
Using the MIBMAN utility to implement management information bases (MIBs)	Chapter 3, "Using the MIBMAN Utility"
Using the PBuilder utility to convert HTML Web pages into compilable C source code	Chapter 4, "Using the Advanced Web Server PBuilder Utility"
Diagnosing problems; reserializing a development board	Chapter 5, "Troubleshooting"

Related documentation

- *NET+Works Quick Install Guide* describes how to install the hardware.
- *NET+Works with GNU Tools Tutorial* provides a brief, hands-on exercise
- *NET+Works with GNU Tools BSP Porting Guide* describes how to port the board support package (BSP) to a new hardware application with GNU tools.
- The NET+Works online help describes the application program interfaces (APIs) that are provided with NET+OS.

For information about third-party products and other components, review the documentation CD-ROM that came with your development kit.

For information about the processor you are using, see your NET+Works hardware documentation.

Documentation updates

Digi occasionally provides documentation updates on the Web site. Be aware that if you see differences between the documentation you received in your NET+Works package and the documentation on the Web site, the Web site content is the latest version.

Customer support

To get help with a question or technical problem with this product, or to make comments and recommendations about our products or documentation, use the contact information listed here:

- United State telephone: 1 877 912-3444
- International telephone: 1 952 912-3444
- email: digi.info@digi.com
- Web site: http://digi.com

NET+Works Introduction

CHAPTER 1

This chapter provides an overview of NET+Works.

Overview

The NET+Works products offer an embedded solution for hardware and networking software that are being implemented into product designs.

NET+Works is designed as an alternative to a PC for products that must be connected to Ethernet for Transmission Control Protocol (TCP) access and for Internet/Intranet access.

The NET+Works package includes:

- A NET+Works processor
- A development board and board support package
- Either a MAJIC/MAJICO debugger or a Raven debugger
- The networking firmware
- The object code with application program interfaces (APIs)
- Development tools
- Sample code
- Documentation

For information about the NET+ARM devices, see the hardware documentation.

System components

This section describes the components that make up the NET+Works software.

NET+Works runtime software

NET+Works software provides the building blocks to help you create your custom applications. You create your application with calls to APIs for:

- The board support package (BSP)
- ThreadX RTOS kernel
- Basic Internet protocols
- Higher-level protocols and services

Board support package

The NET+Works BSP is a collection of ARM object code and C source-code drivers and bootloader. The BSP initializes hardware and software, and it provides power-on self test (POST).

The BSP includes a set of APIs that you use to incorporate device peripheral functionality into your application. In addition, the BSP provides the drivers for your NET+ARM processor or Digi Connect module development board, including Ethernet, Serial, SPI, Flash, USB host, USB device, LCD, PCI/CardBus, and others.

ThreadX RTOS kernel

The RTOS is based on a high-speed picokernel architecture. ThreadX helps you manage complex event synchronization and memory using threads, queues, application timers, semaphores, and event flags.

HTML-to-C compiler

The toolkit provides a compiler that translates your HyperText Markup-Language (HTML) code into C code. You can then compile the C code as part of the project, which allows you to create the pages your menu system needs to perform the tasks necessary for your device. Sample code also is provided.

Advanced Web Server (AWS) PBuilder utility

Using the PBuilder utility, you create or maintain Web pages and recomplile the application program to generate updated images. The PBuilder provides support for HTML, multiple Web object sources, object compression, and advanced security.

Address Configuration Executive (ACE)

NET+Works provides services such as the ACE, which lets you acquire IP parameters at startup from multiple prioritized sources, including BOOTP, RARP, Auto IP, and others.

System requirements

To run NET+Works with GNU Tools, your system must meet these requirements:

- Intel architecture (i586/i686) PC running one of these Microsoft operating systems:
  - Windows XP Professional
  - Windows 2000 Professional

  Be aware that Windows 9x is no longer supported because of limited system resources in the operating system.
- CPU clock rate: 200 MHz or higher
- Minimum system RAM: 128 MB
- Free disk space: 900 MB

Using the HTML-to-C Compiler

CHAPTER 2

This chapter describes the HTML-to-C compiler, which you use to convert HTML and related Web page files to C code.

Overview

This chapter describes the HTML-to-C Compiler, which is one of two Web authoring tools supplied with NET+Works. The HTML-to-C Compiler allows you to develop simple web pages. Another web authoring tool, the Advance Web Server, allows you to create more sophisticated Web content, and provides a way to translate web pages into foreign languages. For more information about the Advanced Web Server, see Chapter 4.

The HTML-to-C compiler converts HyperText Mark-up Language (HTML) and related Web page files to standard C code so you can compile and link the Web pages for an application.

The HTML-to-C compiler provides an easy way to integrate Web pages and content into the Web server. Components converted to C code are easily integrated into NET+Works software applications.

The next sections provide background information about HTML and describe how to use the HTML-to-C compiler.

Web content

In response to HyperText Transport Protocol (HTTP) requests from Web browsers, devices send *Web content*, which consists of

- HTML pages
- Images (such as `.gif` and `.jpeg` files)
- Java applets
- Audio files

To incorporate Web content into an embedded device, you first build an HTTP server into the device by using the application program interfaces (APIs) provided with NET+Works (described in the online help). The server processes HTTP requests and responds with Web content.

The next step incorporates the Web content into the HTTP server. Commercial Web servers installed on UNIX or Windows NT systems have storage disks with large file systems. Incorporating Web content is fairly routine because pages and images are added to an existing directory, making the files Web-accessible. Embedded devices normally have read-only memory (ROM) without a file system. The Web content in such cases must be incorporated directly into the embedded device application stored in ROM.

When you write HTML from scratch, you develop pages by adding HTML markup tags to the text content, using either a text editor or Web authoring tool. You can add the same pages to an embedded device by writing application code to physically return an HTML page. The page is stored in a large character buffer in the device and returned through a network API such as sockets.

As tools for generating HTML pages become more advanced, webmasters do not generate HTML pages by hand. Web authoring tools are more efficient and reduce the amount of typing needed in markup tags.

Static and dynamic content and forms processing

Embedded devices must be able to incorporate *static* content, *dynamic* content, and *forms processing* into the embedded HTTP server source code.

The HTML-to-C compiler automatically converts Web content into application source code:

- *Static* pages are converted into the necessary program calls to send back HTML, image, and applet content that does not change over time. Nothing needs to be added after the page is converted. Static pages are a small part of the content provided by a Web server.
- *Dynamic* content, which changes over time, is necessary for status reporting. This type of content has proprietary non-HTML markup tags inserted with an HTML editor into the HTML source code. The HTML-to-C compiler recognizes these tags and produces shell routines and calls to the routines in the application source code. The embedded designer is then responsible for implementing the routines so that the appropriate dynamic content is returned when the routines are called.
- *Forms processing*, which accepts user input and acts on it, also is necessary for making configuration changes in an embedded device.

An HTML generation or Web authoring tool does not solve the problem of providing dynamic content or forms processing. Application code in the commercial Web server (written in PERL, C++, or Java) is necessary for these types of Web content.

The HTML-to-C compiler also recognizes HTML form tags and adds the shell routines to be called when forms data is sent back to the embedded Web server.

The embedded Web server typically has an API that makes it easy to retrieve the common gateway interface (CGI) data supplied by the browser in response to a forms submission. The embedded designer is responsible for filling in the shell routine with the code necessary to handle the data and send back the reply.

Dynamic content example

Sometimes you want a Web page to look different every time it is accessed. For example, you may want to write a page that provides weather or traffic reports. Dynamic content is useful in such a case. All you have to do is to embed the `_NZZA_` tag into your HTML file. The HTML-to-C compiler understands this tag and generates an empty routine at the end of the generated C file.

This example shows a portion of a Web page that reports the temperature at Logan Airport:

```
<h1> The temperature at Logan is _NZZA_get_temp </h1>
```

This is translated into statements:

```
HSSend (handle, “<h1> The temperature at Logan is”);
na_get_temp(handle);
HSSend (handle, “</h1>”);
```

Note that `_NZZA_get_temp` is being converted into the `na_get_temp` function call, which is located at the end of the file:

```
void na_get_temp(unsigned long handle)
    {
    }
```

You need to add real code to get and format the temperature, and then send it back to the browser. For example:

```
void na_get_temp(unsigned long handle)
    {
        char buf[100];
        int temperature;

        temperature = get_airport_temp();
```

```
        sprintf (buf, "%4d", temperature);
        HSSend (handle, buf);
    }
```

Input form example

One use of an embedded Web server is collecting data (for example, configuration information) from a user. The easiest way to do this is with an input form. The HTTP Server library API `HSGetValue` is used to capture data collected in this form.

To use forms, you need to include an action field in your form; for example:

```
<FORM ACTION="ip_config" method ="POST">
```

This field allows the HTML-to-C compiler to generate an empty function that must be filled in to collect the data sent from the browser. `Post_` is prefixed to the name of the action.

The generated function name must be a valid C identifier, so your action name must not contain any illegal character s. For example, `Post_ip_config` is added at the end of the file:

```
void Post_ip_config (unsigned long handle)
{
}
```

To extract data sent with the form, use `HSGetValue`, as shown next:

```
void Post_ip_config (unsigned long handle)
{
            char ipaddr[32];

    /* assume the data is a text input, with name
      attribute set to IP_ADDR */
    HSGetValue (handle, "IP_ADDR", ipadddr, 32);
    …
    /* at this point, whatever the user
        input will be stored in ipaddr   */
}
```

Preparing to use the HTML-to-C compiler

The file name of an HTML uniform resource locator (URL) is its full URL name. Because slash (/) is not a valid character for a file name, slashes are converted to underscores (_). For example, `/abc/def.html` is generated as `_abc_def.c`.

To incorporate Web content into an embedded device, you first build an HTTP server into the device using the APIs provided with NET+Works.

▶ **To build an HTTP server into a device:**

1. Create Web pages by using an HTML editor.
2. Use the HTML-to-C compiler to convert the Web pages into C code.
3. If the Web pages contain dynamic content or forms, add your own code to the generated empty C routines.
4. Put all your C source files into your application's `makefile`.

How the HTML-to-C compiler works

The HTML-to-C compiler recognizes two file types:

- **Text files.** Any file with an extension of `.html`, `.htm`, or `.txt`. A text file causes the compiler to generate a .c file with a similar file name.
- **Binary files.** All binary files are converted and stored in the `bindata.c` file. All the URL information, including security, is stored in the `url.c` file.
  The file names `bindata.c` and `url.c` are default names you can change.

For example, assume your Web server contains four files — `home.htm`, `x.htm`, `y.jpg`, and `z.gif` — and you want to organize your Web hierarchy (URLs) like this:

```
/home.htm
/x.htm
/images/y.jpg
/images/z.gif
/~John/home.htm
/~John/y.jpg
```

These C files are generated:

- `_home.c` (from `url /home.htm`)
- `_x.c` (from `url /x.htm`)

- _~John_home.c (from url /~John/home.htm)
- bindata.c (contains data for y.jpg and z.gif)
- url.c

These entries are added to the URL table in the url.c file:

```
URLTableEntry URLTable[] = {
“/home.htm”, 1, Send_function_0, (HSTypeFnHtml)HSTypeHtml, 0,
     Unprotected,
“/~John/home.htm”, 1, Send_function_1, (HSTypeFnHtml)HSTypeHtml, 0,
     Unprotected,
“/~John/y.jpg”, 0, Send_function_2, (HSTypeFnBinary)HSTypeJpeg,
      453, Unprotected,
“/images/y.jpg”, 0, Send_function_2,
     (HSTypeFnBinary)HSTypeJpeg, 453, Unprotected,
“/images/z.gif”, 0, Send_function_3, (HSTypeFnBinary)HSTypeGif,
     499, Unprotected,
“/x.htm”, 1, Send_function_5, (HSTypeFnHtml)HSTypeHtml, 0,
     Unprotected,
NULL  /* last entry MUST be NULL */
};
```

Although there is only one home.htm file, two files are generated because two URLs (home.htm and /~John/home.htm) refer to it. This lets you use different code to serve the same page (home.htm) under different absolute URLs.

Binary data, however, is treated differently because it does not need to be customized. Only one copy of the data is stored in bindata.c for y.jpg, even though it is referred to in both /~John/y.jpg and /images/y.jpg.

The URL table is used in the AppSearchURL function. When a user clicks a URL link from a Web browser, the Web server does a search on the URL table. If the server finds the entry for the URL, it calls the corresponding function (for example, Send_function_1) to send back the page.

Here is an example of the Send_function_1 in the file _~John_home.c function:

```
void Send_function_1(unsigned long handle)
{
    HSSend (handle, “<html>\n”);
    HSSend (handle, “\n”);
    HSSend (handle, “<head>\n”);
    HSSend (handle, “<meta http-equiv=\”Content-Type\“\n”);
    HSSend (handle, “content=\”text/html; charset=iso-8859-
        1\“>\n”);
```

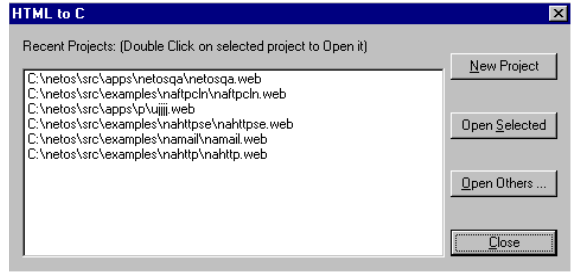
```
HSSend (handle, "<meta name=\"GENERATOR\
    "content=\"Microsoft FrontPage Express 2.0\">\n");
HSSend (handle, "<title>John's Home Page</title>\n");
HSSend (handle, "</head>\n");
HSSend (handle, "\n");
HSSend (handle, "<body bgcolor=\"#FFFFFF\">\n");
HSSend (handle, "\n");
HSSend (handle, "<h3>Welcome to John's Home </h3>\n");
HSSend (handle, "\n");
    HSSend (handle, "</body>\n");
    HSSend (handle, "</html>\n");
    }
```

Using the HTML-to-C compiler

Files for the same Web server application are organized as a *project*. A project file has a file extension of `.web`.

▶ **To start the HTML-to-C compiler:**

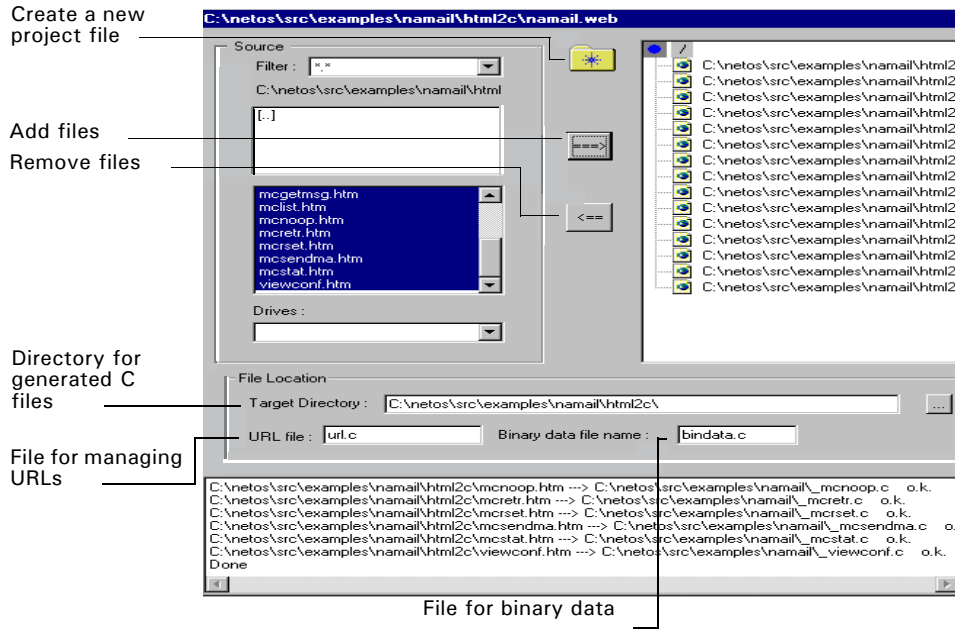
1 Select **Start → Programs → NET+OS63_GNU → HTML-to-C Compiler.** The **HTML to C** dialog box opens:



2 Do one of these steps:

- To open a project that appears in the list, select the project from the list and click **Open Selected.**
- To open a project that does not appear in the list, click **Open Others**, then locate and select the project.
- To create a new project, click **New Project**, then enter a name and select a location for the project.

This dialog box opens:



From this dialog box, you can:

- Create a new project.
- Remove obsolete data.
- Specify locations for generated C files and names for binary data and URL files.
- Generate C source files.
- Open and edit the `url.c` file.
- Set or change the project's home page.

Creating a new project

▶ **To create a new project:**

1. From the main dialog box, click the folder icon in the **Source** section.
2. Enter a name and select a location for the new project.

Removing obsolete data

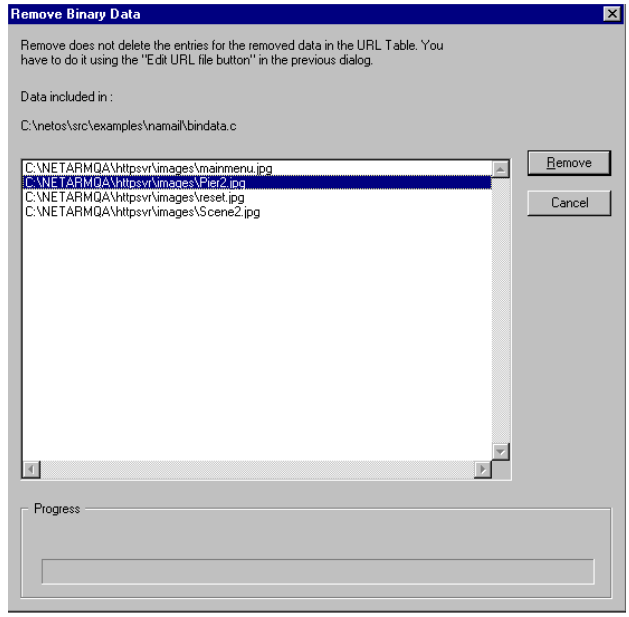
Removing obsolete data from a project file reduces compilation time. Use caution when you remove files; you may need the data for troubleshooting.

Be aware that removing data here does not delete the entries for the removed data in the URL table. You must use **Edit URL file** to remove the data from the table.

▶ **To remove obsolete data from a file:**

1 From the main dialog box, click **Remove Bin data**.

The **Remove Binary Data** dialog box opens, and displays the names of all binary files in the `bindata.c` file:



2 Select one or more files to delete from the list, and click **Remove**.

Adding or removing source files

You can add or remove source files at any time.

▶ **To add a source file:**

1. In the **Source** section of the main dialog box, locate and select the file to add to your project.
2. Click the right arrow.

▶ **To remove a source file:**

1. In the **Source** section of the main dialog box, locate and select the file to remove from your project.
2. Click the left arrow.

Specifying the location of files

From the main dialog box, you can specify the location for the generated C files, and the names of the URL and binary data files, as described here:

- To specify the directory in which to put the generated C files, enter the path name in the **Target Directory** input box.
- To specify a file name for the URL file, enter the name in the **URL file** input box. The default name is `url.c`.
- To specify a file name for the binary data file, enter the name in the **Binary data file name** input box. The default name is `url.c`.

  If your project does not have any binary files, you do not need to include this file in your `makefile`.

Generating C source files

From the main dialog box, you can generate C source files in two ways:

- To generate *only* the files that have changed since the last build, click **Build**.
- To generate all files, click **Build All**.

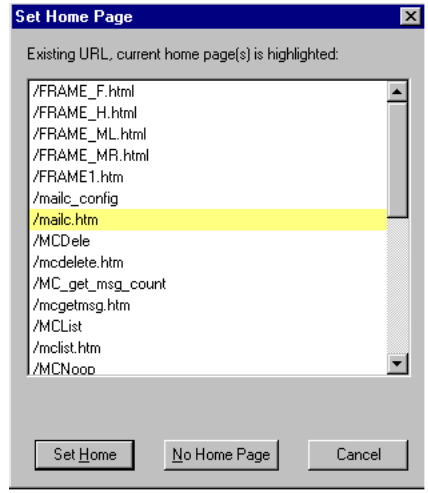
Setting or changing a project's home page

The home page is the page that appears in the browser when you type only the IP address of the server. The URL of a home page is always a forward slash (`/`).

▶ **To set or change the home page of a project:**

1 From the main dialog box, click **Home Page**.

The **Set Home Page** dialog box opens:



2 Select the URL of the page you want to use as home page and click **Set Home**.

If you do not want to specify a home page, click **No Home Page**.

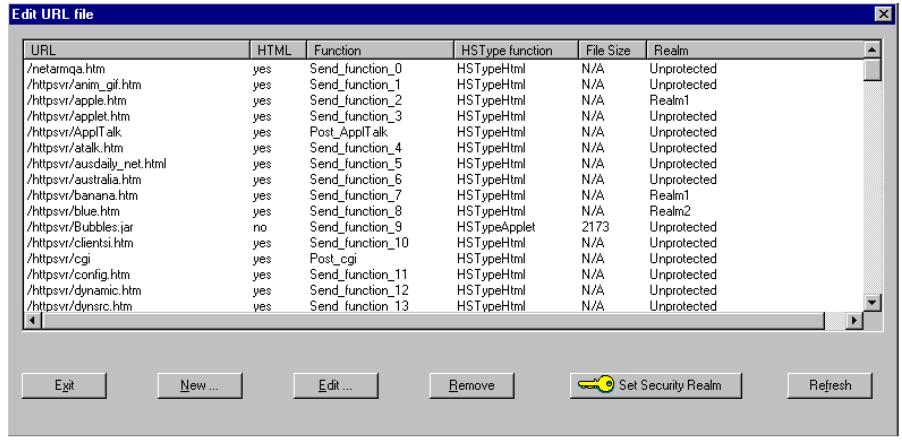
Editing URL files

The `url.c` file contains the URL table, from which you can:

- Add or delete entries.
- Edit entries.
- Set or change security realms.

Opening the url.c file

To open the `url.c` file, from the main dialog box, click **Edit URL file**. The **Edit URL file** dialog box opens:



Adding and deleting URLs

▶ **To add a new URL:**

1. From the **Edit URL file** dialog box, click **New**.

   The **Edit URL** dialog box opens.

2. Provide the necessary information, and click **OK**.

▶ **To delete a URL:**

1. From the **Edit URL file** dialog box, select the URL to delete.
2. Click **Remove**.

   Be aware that the deletion happens immediately; you do not get a prompt to confirm the deletion.

Editing a URL

▶ **To edit a URL:**

1. From the **Edit URL file** dialog box, select the URL you want to edit.
2. Click **Edit**.

   The **Edit URL** dialog box opens.

3. Make the changes you want, and click **OK**.

Setting the user and password of a URL

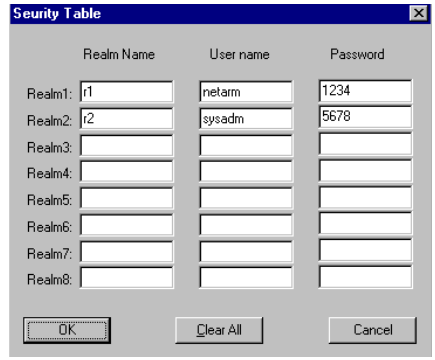
The HTTP server supports the basic access authentication scheme in the HTTP protocol. The combination of the username, password, and list of Web pages is a *realm*. You can create up to eight realms.

You must set up a security table before you start the HTTP server.

▶ **To set up or change realm information:**

1 From the **Edit URL file** dialog box, click **Set Security Realms**.

The **Security Table** dialog box opens:



2 For each realm, type the realm name, user name, and password, and then click **OK**.

Using the MIBMAN Utility

CHAPTER 3

This chapter describes the MIBMAN utility, which you use to implement management information bases (MIBs).

Overview

MIBMAN is a utility that translates Simple Network Management Protocol (SNMP) Management Information Bases (MIBs) into C code that contains:

- Templates for action routines that you implement
- Management API declarations for MIB objects that correspond to management variables

You control which action routines and management variable declarations are generated through configuration files.

This chapter describes how to convert an SNMP MIB into C code and how to write action routines.

Terms and concepts

The next sections provide some terminology to be familiar with as you use MIBMAN.

SNMP

SNMP, which defines a system for managing network devices, is implemented in multiple modules. SNMP consoles run on network workstations and provide the user interface. SNMP agents run on managed devices and handle communications with remote SNMP consoles. When a console sends a request to an agent, the agent decodes the request and calls subroutines in the managed device that handles the request. You implement the subroutines, which are called *action routines*.

A MIB defines the view of a managed device that the device's agent presents to an SNMP console.

The MIB defines a set of objects the console can read and write. These objects can either correspond to variables that exist on the device or be synthesized by the agent:

- MIB objects that represent variables that exist are called *real objects*.
- MIB objects that do not exist are called *virtual objects*.

For example, a MIB might define one object that is the count of Ethernet packets received so far, and another object that indicates the health of the system. The count of Ethernet packets received is probably stored in a real variable. On the other hand, the agent can determine the health of the system on-the-fly by examining the state of several system variables and processes.

Scalar MIB objects

A *scalar object* represents a single item with a simple type. If a scalar object is real, MIBMAN can generate most of the code needed to implement it.

You can create a declaration for the variable in the management API and use generic action routines to read and write the variable. Then you can have the application set an initial value for the variable and update it as necessary.

MIBMAN cannot always use generic action routines. For example, a MIB object could represent the state of an LED, and special purpose action routines might be needed to turn the LED on and off when a user changes the value of the object. In such a case, you can configure MIBMAN to generate templates for the action routines and a declaration for a management variable to represent it. If the object is virtual, you can use MIBMAN to generate templates for the action routines without generating a management variable declaration.

MIB tables

You can arrange MIB objects in tables. A *table* is a set of MIB objects repeated in multiple rows. Objects in a row are *columnar objects*. Columnar objects all have simple types, like scalar objects. An SNMP table is similar to an array of C structures, where each row represents a C structure, and each columnar object represents a field in the structure. Like scalar objects, tables can represent either:

- Real tables of information stored in memory
- Virtual objects that the agent creates on-the-fly

Tables are read and written one columnar object at a time. When a columnar object is accessed, the object is identified with:

- An object identifier (OID) that determines which columnar object in the row is being accessed
- An index that determines which row in the table is being accessed

Although MIBs describe the elements that make up a table index, they do not describe how the index works; you implement the table's indexing scheme. For more information, see the comments in the MIB and the RFCs that describe the MIB.

Tables can have relationships with other tables; for example:

- One table can expand another table by defining additional columnar objects. In this case, the two tables have a one-to-one relationship.
- One table can be a different view of another table. For example, one table might consist of the same rows in another table sorted into a different order.
- One table can be a sub-table of another. Implementing this relationship requires complex indexing schemes.

These relationships can cause the fate of rows in one table to affect rows in another table. For example, assume one table expands another. When a row in one table is deleted, the corresponding row in the other table also must be deleted. These types of relationships, which you implement, are not described by the MIB, but in comments in either the MIB or the RFCs that describe the MIB.

MIBMAN can create management variable declarations and action routine templates for MIB tables that represent real tables in memory. You must initialize the tables and update the data in them as necessary. You also must complete the action routines by coding the table's indexing scheme and fate relationships to other tables. If the table is virtual, MIBMAN can generate templates for its action routines.

Traps

A *trap* is a message that an agent sends to a console when certain events occur. Although the contents of traps are defined in MIBs, the MIBs do not specify the events that cause the agent to send the trap. These events are explained by comments in the MIB and in the RFC that describes the MIB.

You can use MIBMAN to generate a set of data structures that applications use to generate and send traps.

Action routines

Action routines are functions that an SNMP agent calls to read or write MIB objects. MIBMAN generates templates of these functions that you complete to implement the MIB. The agent passes parameters to the functions that identify which object is being accessed, and the functions perform the operation and return the result.

Implementing a MIB: an example

`namib`, a sample application on the NET+Works installation CD, demonstrates how to implement a MIB. The sample MIB allows you to read and change the state of the LED on the board from a MIB browser. `namib` also demonstrates how to implement a MIB with tables.

Converting an SNMP MIB into C code

To convert an SNMP MIB into C code, you use this general procedure:

1. Use a MIB compiler to convert the MIB definition into an intermediate file.

   To compile the MIBs, you can use the SMICng MIB compiler. SMICng, an industry-standard compiler, is shipped with the SNMP agent.

2. Run MIBMAN to convert the intermediate file generated by SMICng into three C source files.

   These source files contain the C variable definitions and action routine templates needed to implement the MIB.

3. Do the final integration.

The next sections provide details about the three steps.

Step 1: Using SMICng

Because the developers of SNMP tried to design a format for SNMP MIB definitions that would be readable by both machines and people, MIB definitions are difficult for anyone or anything to read. The SMICng compiler, however, converts a MIB definition into a form that is easier for software to read.

For more information about SMICng, see the SMICng online documentation.

Here are the basic steps for using SMICng:

1. Strip off all extraneous text.

   MIB definitions are usually contained within text files (RFCs) that contain other text besides the MIB definition itself. If the MIB uses definitions from other MIBs, you also need to clean up the text files that contain the definitions for these MIBs.

2. Create an include file for SMICng that lists all the MIBs used by the one that is being processed.

   Include the MIB itself as the final file.

3. Run SMICng with this command line:

   `smicng -z -cm` *`yourMib.inc > yourMib.out`*

   where:

   - *`yourMib.inc`* is the name of the include file you created in step 2 of this procedure.
   - *`yourMib.out`* is the name of the intermediate file the compiler creates.

Step 2: Using MIBMAN

The MIBMAN utility operates on the intermediate files that SMICng creates. MIBMAN accepts up to four arguments, as shown here:

`MIBMAN` *`list-file configuration-directory output-directory var-filename`*

where:

- *`list-file`* is the file that contains a list of intermediate files generated by SMICng.
- *`configuration-directory`* is the directory in which configuration files are located.
- *`output-directory`* is the directory in which output files are generated.
- *`var-filename`* is the file that MIBMAN creates to contain a list of the variables defined by the MIB and their OIDs.

The *`list-file`* argument is required; the other three arguments are optional. If you do not specify the optional arguments, MIBMAN uses the current directory and does not generate a list of MIB variables. In addition, the *`list-file`* must list the MIB files so that MIBs that have dependencies are listed after the MIBs that resolve the dependencies.

For example, assume RFC1213-MIB is being processed. This MIB depends on items defined in RFC1155-MIB and RFC1212-MIB. So, a list file for RFC1213-MIB would be:

```
RFC1155-MIB.OUT
RFC1212-OUT.OUT
RFC1213-OUT.OUT
```

Generated files

MIBMAN generates two C files and a header file (.h) for each MIB it processes that defines variables or traps. Here are the naming conventions:

- By default, the name of the MIB module determines file names.
- MIBMAN converts the MIB module name into legal Windows file names by changing hyphens and periods in the name into underscores and appending the extensions .c and .h.
- The suffix *Action* is appended to the template file name.
- If the module name starts with a digit, an underscore is added to the beginning of the file name.

One C file —the *definition file* — contains the definitions for variables and structures that the management API and the SNMP agent use, including:

- An array of manVarType elements used to create the variables in the management API database
- An array of struct variable elements used to register the variables with the SNMP agent
- Information about SNMP traps that is needed to implement the traps

The other C file (the template file) contains templates for action routines you use to implement the MIB.

The header file contains declarations for functions and variables defined in the two C files, including declarations for:

- The manVarType and struct variable arrays defined in the definition file
- The action routine templates in the action file that read variables
- The constants needed to implement the action routines

By default, MIBMAN generates the C code by:

- Creating management variables to represent all scalar objects
- Using generic action routines to access all scalar objects
- Creating management table variables to represent all MIB tables
- Creating templates or action routines that read and write the tables

None of the management variables is protected by semaphores. Indexing for management table variables is left undefined. You can change these default settings through a configuration file.

Data types

SNMP data types are represented by similar management variable data types, as shown in this table:

These SNMP data types	Are represented as
Textual conventions, octet strings, and opaque objects	`MAN_OCTET_STRING_TYPE` variables.
`INTEGER` and `Integer32`	`INT32` types.
`Unsigned32`, `Gauge`, `Gauge32`, `Counter`, `Counter32`, and `TimeTicks`	`WORD32` types.
`Counter64`	`WORD64` type.
`OBJECT IDENTIFIER`	Arrays of 129 `WORD32` elements. The length of the OID is encoded in the last element of the array.
`IPaddress` and `NetworkAddress`	An array of four `WORD8` elements.
`BITS`	Octet strings in which each element contains 8 bits. SNMP bit label `0` is the least significant bit in the first byte of the string. The number of bit labels defined in the MIB determines the string's length.

Tables

MIBMAN generates code to register tables with the management API by using the MAN_TABLE_TYPE data type. Individual rows are represented as C structures where each columnar element is represented as a field in the structure.

Preparing to reserialize

Before you reserialize the development board, start a HyperTerminal session. Keep the HyperTerminal window open during all your testing.

If your system resources are limited, keep the HyperTerminal window open only when you power cycle the board.

Assigning a MAC address to the NET+Works board

Each device on the network must have a unique Ethernet media access controller (MAC) address. The NET+Works development board comes preconfigured with a factory-set MAC address that is printed on a sticker on the board.

The MAC address can be lost if NVRAM is corrupted by an application under test. In such a case, you must restore the MAC address to make sure that the board can communicate over the network. The development board ships with an application written in flash ROM that you can use to restore the MAC address. From the debugger, you also can use any sample application built with the configuration dialog enabled.

▶ **To restore a board's original Ethernet MAC address:**

1. Connect the board to a serial port on your system.
2. Start a HyperTerminal session on the serial port.
3. Power up the board.

   A message similar to this one appears after a brief pause:

```
NET+WORKS Version 6.2
Copyright (c) 2003, NETsilicon, Inc.
PLATFORM: net50bga_a
APPLICATION: Ram-based FTP Server Application
------------------------------------------------------------------
NETWORK INTERFACE PARAMETERS:
IP address on LAN is 1.2.3.4
LAN interface's subnet mask is 255.255.255.0
IP address of default gateway to other networks is 1.2.3.4
HARDWARE PARAMETERS:
Serial channels will use a baud rate of 9600
This board's serial number is N99999999
This board's MAC Address is 00:40:9D:00:43:35
After board is reset, start-up code will wait 5 seconds
Default duplex setting for Ethernet connection: Full Duplex
------------------------------------------------------------------
Press any key in 5 seconds to change these settings.
```

4 Enter the configuration dialog by pressing a key before the timeout expires.

5 At the prompt, enter the system password:

```
Netsilicon
```

6 Enter the values for the IP stack configuration settings and serial port baud rate.

7 At the prompt, enter the Ethernet MAC address that appears on the sticker on the board.

8 Respond to the prompts to set up the remaining configuration settings.

This is a sample dialog:

```
Enter the root password: **********
Reset configuration to default values (Y/N)? Y

For each of the following questions, you can press <Return> to select
the value shown in braces, or you can enter a new value.

NETWORK INTERFACE PARAMETERS:
Should this target obtain IP settings from the network? [N] y

SECURITY PARAMETERS:

Would you like to update the Root Password? [N[

HARDWARE PARAMETERS:

Set the baud rate of Serial channels [9600]?
The new baud rate is 9600
The baud rate will be changed on next power up
Please set the baud rate for your terminal accordingly

Each development board must have a unique serial number
Set the board's serial number [N99999999]? N12345678
The board's new serial number is N12345678

Each development board must have a unique Ethernet MAC address.
Set the board's Ethernet MAC Address [00:40:9D:BA:DB:AD]?
00:40:9D:12:34:56
This board's new Ethernet MAC address is 00:40:9D:12:34:56

How long (in seconds) should CPU delay before starting up [5]?

Normally the board will automatically negotiate with the network hub (or
switch) to determine the Ethernet duplex setting; however some hubs do
not support autonegotiation.
What duplex setting should be used in this case (Full or Half)? [Full
Duplex]

Saving the changes in NV memory...Done.
```

Restoring the contents of flash ROM

NET+Works development boards ship with a boot ROM program written in flash ROM. The boot ROM program implements support for debugging and provides an FTP server that you can use to update flash ROM.

You restore the original boot ROM program by using a procedure in which you:

1. Configure the target development board and in-circuit emulator (ICE).
2. Build the bootloader image.
3. Build the application image, and build the FTP Flash download program from the `naftpapp` application.
4. Send the `rom.bin` file in your platform directory and the `image.bin` file in the `naftpapp/32b` directory
5. Verify the boot ROM image on the target development board.
6. Verify the application.

The next sections provide details about each step in the procedure.

Note: Be aware that the order of the tasks for restoring the contents of flash ROM is important. You *must* do the tasks in the order in which this document presents them.

Step 1: Configure the development board and the MAJIC

▶ **To set up the development board and the MAJIC:**

1. Connect the MAJIC as described in the *NET+Works Quick Install Guide*.
2. Connect target serial port 1 to the communications port of your computer.
3. Power up the target development board.
4. Power up the MAJIC.
5. Disable flash on the board. Depending on the board you are using, you might disable flash using either a jumper or a switch. For details, see the jumpers and components guide for the board you are using.
6. Start a HyperTerminal session.

Step 2: Building the bootloader

▶ **To build the bootloader:**

1. Edit `bsp.h` and make sure the configuration settings are correct.

   For information about these settings, see the *NET+Works BSP Porting Guide* and the online help.
2. Using the `makefile` in the `C:/NETOS63_GNU/src/bsp` directory, build the BSP.
3. To select your platform, use the command-line option and enter:
   `make PLATFORM = platform-name`

Step 3: Building the application image and starting naftpapp

▶ **To build the application image and start the naftpapp application:**

1. Prepare the application image.
2. Edit the `appconf.h` file for `naftpapp`, and make sure the application is configured to generate a configuration dialog.

   To generate a dialog, set the constant `APP_DIALOG_PORT` in `appconf.h`.

   For details about `APP_DIALOG_PORT` in `appconf.h`, see the online help.
3. Rebuild the `naftpapp` application.
4. Start the debugger and load `naftpapp`.
5. Start the application.

   `naftpapp` prompts you with the standard NET+OS configuration dialog box (unless you have disabled this feature).
6. Verify that the network settings are correct, and change them if necessary.

Step 4: Sending rom.bin to the development board

▶ **To send `rom.bin` of the bootloader to the development board:**

1. Open a command shell.
2. Change to this directory:

   `C:/NETOS63_GNU/src/bsp/your-platform`

   where you replace `your_platform` with the name of your platform.
3. To start the Windows FTP client, enter this command and press Enter:

   `FTP` *`a.b.c.d`*

and press Enter.

where *a.b.c.d* is your unit's IP address.

4. When you are prompted for a username, enter:

   ```
   (none)
   ```

5. To select binary mode transfer, enter:

   ```
   bin
   ```

6. Enter this command:

   ```
   put rom.bin
   ```

7. When the transfer is complete, enter:

   ```
   quit
   ```

8. Exit from the debugger.

Step 5: Verifying the boot ROM image on the development board

At this point, the bootloader has been written into the boot sector of flash. Now you need to write the application into flash.

▶ **To write an application into flash:**

1. Build the application you want to write into flash.
2. Restart the `naftpapp` application in the debugger.
3. Change to this directory:

   ```
   C:/NETOS63_GNU/src/examples/naftpapp/32b
   ```

4. To start the Windows FTP client, type this command and press Enter:

   ```
   FTP a.b.c.d
   ```

   where *a.b.c.d* is your unit's IP address.

5. When you are prompted for a username, enter:

   ```
   (none)
   ```

6. To select binary mode transfer, enter:

   ```
   bin
   ```

7. Enter this command:

   ```
   put image.bin
   ```

8. When the transfer is complete, enter:

   ```
   quit
   ```

9. Exit from the debugger.

Index

Symbols

.gif and .jpeg images as Web content 6

A

accessor functions, setting 35
action routine
    called by SNMP agent 23
    defined 20
    generating 34
action routines 23
Advanced Web Server. See *AWS*.
agent 23
agent, SNMP 20
APP_DIALOG_PORT constant 75
appconf.h file 75
application samples
    naficgi 57, 68
    nahttp_pd 57, 68
    namib 23
array subscripts and MAW module 64, 65
audio files as Web content 6
AWS 54
    and comment tags 61
    and custom variables 58
    and customizable routines 59
    and MAW module 67
    data types of 60
    function type, specifying 64
    phrase dictionary technique 67
    security 63
    Web content compression 54

B

binary files and the HTML-to-C converter 10
bindata.c file 10, 14
board support package. See *BSP*.
boot ROM
    and restoring contents of flash ROM 74
    application 71
    image, verifying 76
BSP 3
    and NET+OS 3
bsp.h file 71, 75

C

C code, converting an SNMP MIB into 23
C source files, generating 15
cgi.c file 57, 68
columnar objects 21
comment tags 54, 57–58
comments in C files and header files 32
compression and phrase dictionaries 67
consoles, SNMP 20
constants, controlling names of 35
contents of flash ROM, restoring 74
controlling management variables 33
converting an SNMP MIB into C code 23
converting HTML and related Web pages to C code 6

D

data abort 70
data types, SNMP 26
default values MIBMAN uses, overriding 30
definition file 25
deleting obsolete data from project file 14
development board
  and LEDs 70
  reserializing 71
  restoring Ethernet MAC address 72
  sending rom.bin to 75
  verifying boot ROM image on 76
diagnosing errors 70
dictionary, adding new phrases 68
dynamic Web content, defined 7

E

embedded device, incorporating Web content into 6, 10
error
  handler, installing 66
error, fatal 70
Ethernet MAC address, restoring original 72
exceptions
  diagnosing 71
  types of 70

F

fast interrupt 70
file.c file 57, 68
filename conventions for HTML URL 10
files generated by MIBMAN 25
flash ROM, restoring contents of 74
forms processing Web content, defined 7
function stubs 54

G

generated files and MIBMAN 25
generating
  action routines 34
  C source file 15
generic action routines 21
getUsername function 59
gif and jpeg images as Web content 6
global variable prefixes, setting 33

H

home page, setting or changing for project 16
hooks. See *function stubs.*
HTML
  pages as Web content 6
  URL, filename conventions for 10
HTML-to-C converter
  file types recognized 10
  purpose of 6
  using 12
HTTP server, setting up security table for 18
HyperTerminal 72, 74

I

implementing traps 29
index information, setting 33
input form 9
integrating Web pages into the Web server 6

J

Java applets as Web content 6
jpeg and gif images as Web content 6

L

LEDs
  and troubleshooting 71
  observing during power cycle 71
list.bat file 56, 68

M

MAC address
  locating on development board 73
  restoring to a development board 72
magic cookies 54
maintaining and modifying Web content 68
makefile 10, 15, 75
management API 28
management API ID, setting 33
management variables and semaphores 26
management variables, controlling 33
MAW module
  and array subscripts 65
  error handling 66
  security 63
mawInstallErrorHandler function 66
mawInstallTimeoutFunction function 65
mawSetAccessTimeout routine 65
MIB
  action routines 20, 23
  and traps 29
  converting an SNMP MIB into C code 23
  definition file 25
  dependencies 24
  object identifier 21
  objects in tables 21
  real objects 20
  relationships between tables 22
  table index 21
  template file 25
  traps 22
  virtual objects 20

MIBMAN
and controlling comments in C and header files 32
and MIB module names 25
and SMICng intermediate files 24
configuration file 30
files generated by 25
include files 35
integration with Management API 28
overwriting files 31
register tables 26
MIBMAN utility 20
Microsoft HyperTerminal 72, 74

N

naficgi sampe application 57
naficgi sample application 57, 68
naftpapp application 74, 75, 76
nahttp_pb sample application 68
nahttp_pd sample application 57
namib sample application 23
NET+OS
security supported by 63
netarm1.c file 56
netarm1_v.c file 56

O

objects
real 20
scalar 21
virtual 20
observing LEDs during power cycle of board 71
obsolete data, removing from project file 14
original Ethernet MAC address, restoring 72
overriding MIBMAN default values 30

P

password
for URL, setting 18
PbSetUp.txt file 56
PBuilder Helper
described 54
linking the application 57
PBuilder Web Application Toolkit 55
sample applications 68
phrase dictionaries and compression
example 67
power cycling a board, LEDs and 71
prefetch abort 70
private structures and routines 55
project
creating new 13
defined 12
setting or changing home page of 15

R

real objects 20
removing obsolete data from project file 14
reserializing a development board 71
restoring the contents of flash ROM 74
rom.bin, sending to development board 75
RpFormInput tag 61

RpFormSingleSelect tag 62
RpFormTextAreaBuf tag 62
RpPages.c file 56
RpSingleSelectOption tag 63
RpUsrDct.txt file 56, 67

S

sample applications 57
  naficgi 57, 68
  nahttp_pd 57, 68
  namib 23
scalar object 21
security 63
security realms, defined 57
security table, setting up for HTTP server 18
security.c file 57, 68
semaphore timeout and MAW module 64
semaphores and management variables 26
semaphores, setting 34
Simple Management Network Protocol. See *SNMP*.
SMICng 23
SMICng intermediate files and MIBMAN 24
SMICng MIB compiler 23
SNMP
  agents and action routines 23
  consoles 20
  data types 26
  defined 20
SNMP_SET_ACTION routine 40, 45, 50
SNMP_SET_COMMIT routine 45, 50
SNMP_SET_FREE routine 46, 51
SNMP_SET_RESERVE routine 45, 50
SNMP_SET_UNDO routine 40, 46, 51
software interrupt 71
static Web content, defined 7
suppressing management variables 33

T

table 21
template file 25
templates for action routines 21
text files and the HTML-to-C converter 10
trap, defined 22
traps
  and MIBs 22
  implementing 29
types of Web content 6

U

undefined interrupt 71
URL, setting user and password for 18
url.c file 10, 13, 15

V

virtual objects 20

W

Web content
  maintaining and modifying 68
  types of 6
Web page
  file extensions 56
  integrating into the Web server 6
Web server libraries 54
Windows HyperTerminal 72

Digi®