Home

Getting Started with Hawk™

Version 2.5

RadiSys.
THE POWER OF WE

www.radisys.com
Revision A • July 2006

Copyright and publication information

This manual reflects version 2.5 of Hawk.

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.

Contents

1 Introduction to Hawk™

Hawk™ is the open Integrated Development Environment (IDE) for Microware's OS-9® real-time operating system. The Hawk IDE environment can be custom-tailored; you can add custom expert and productivity enhancement features by taking advantage of the open application interface. In addition, the Hawk development environment enables you to work in a seamless workspace integrating the following functions and tools:

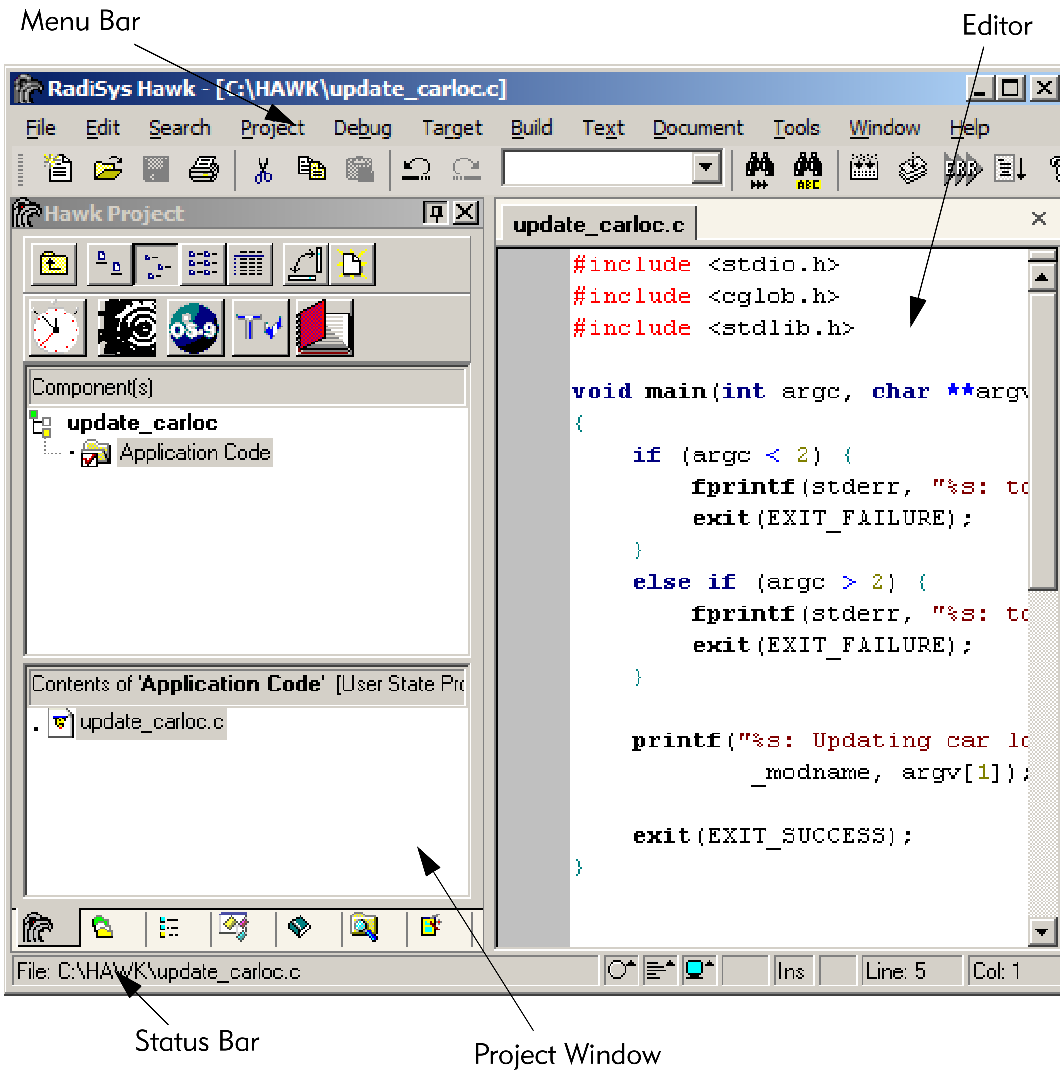
- Project Manager
- Editor
- Debugger
- Profiler
- Ultra C/C++ Compiler

This manual only contains information to help familiarize you with the Hawk IDE and its uses. For more detailed information about Hawk, refer to the *Using Hawk* manual.

Hawk Tools Overview

The Hawk development environment, as shown in Figure 1-1, contains a set of tools consisting of a project manager, source code editor, debugger, and Ultra C/C++ compiler.

Figure 1-1. Hawk Integrated Development Environment



```
#include <stdio.h>
#include <cglob.h>
#include <stdlib.h>

void main(int argc, char **argv
{
    if (argc < 2) {
        fprintf(stderr, "%s: to
        exit(EXIT_FAILURE);
    }
    else if (argc > 2) {
        fprintf(stderr, "%s: to
        exit(EXIT_FAILURE);
    }

    printf("%s: Updating car lo
            _modname, argv[1]);

    exit(EXIT_SUCCESS);
}
```

Project Manager

The Project Manager window is located in the left portion of the Hawk IDE environment. The Project Manager is responsible for the following tasks:

- organizing each software project within the project space
- identifying the dependencies between software components
- recording detailed build information
- controlling the build process

The Project Manager organizes source files, makefiles, libraries, and additional project files needed to build an application. Hawk saves your settings to a .PJT file and an .MPJ file. The .MPJ file maintains the relationship between components and units and the settings for the components and units. It also replaces the makefile task so that when a build is requested, Hawk automatically performs the old make utility task for you. The types of project files are summarized in Table 1-1.

Table 1-1. Project File Types

.pjt file	Maintains a list of all the files in a project and some project settings for the Hawk environment
.mpj file	Retains the structure and setting of the properties in a Hawk project
.mpjBackup file	Is a copy of the current .mpj file being saved

Components and units are the entities that the Project Manager uses to form logical associations between the various files it manages. A component is comparable to a module, descriptor, or driver, and a unit is an individual file.

Project Spaces

Project spaces store sets of projects and allow multiple projects to be open at one time. Before a project can be made, a project space has to be set up. Once one is created, it will appear in the Project Manager as a file cabinet icon and will be given a .psp extension.

Only one project space can be open at a time.

Workspaces

A workspace maintains state information about a project. It differs from a project in that it does not store the system-wide options normally stored in a configuration file. In a sense, the workspace is like a Hawk state file that can be “swapped” in and out as needed.

State files retain information about the windows and buffers opened during the last Hawk session and the position in which those windows and buffers existed. Workspaces are like mini-state files within projects. Each workspace retains a separate set of window information. Other state information such as search options, response histories, and bookmarks are stored as part of the project.

Editor

The Hawk Editor Window is located in the upper right portion of the Hawk IDE environment, as displayed in Figure 1-1. It has the following features:

- HTML viewer
- merge and difference
- Help manager
- API assistance
- syntax highlighting (ChromaCode)
- DLL extensibility
- elided text (selective display)
- IDE integration
- button links
- build file support

Debugger

Hawk can be used for debugging both applications and OS-9 system components. Application processes typically run in user-state and OS-9 system components run in system-state.

Debugging in User- and System-State

User-state application processes are not allowed by the kernel to interfere with the operating system; thus, errant pointers and bad logic do not cause system crashes or process failures. System-state is used by the components of OS-9, although if necessary processes can be designed to run in system-state. The operating system, drivers, device descriptors, and file managers operate in system-state. In this environment, the code associated with the operating system and its subsystems has complete access to the system.

Every OS-9 kernel has built-in debugging support, allowing the kernel to control the process that is being debugged. As a result, the debugging process for user-state applications is simplified. For both user- and system-state debugging, a client-server model is used, in which the Windows host machine acts as the client and the OS-9 target machine acts as the server. To successfully perform debugging, the following conditions must be met:

- You must have a stable TCP/IP connection between the Windows host machine and the OS-9 target machine.
- OS-9 must have low-level network I/O or SoftStax® installed and properly functioning.
- The debugging daemons (undpd or spfndpd) must be running.

If you do not have a fully functional TCP/IP connection established between the Windows Host machine and the OS-9 target machine, please refer to the board guide for your Microware OS-9 product.

To assist in the debugging of your user and system-state code, the Hawk Debugger contains a number of powerful features:

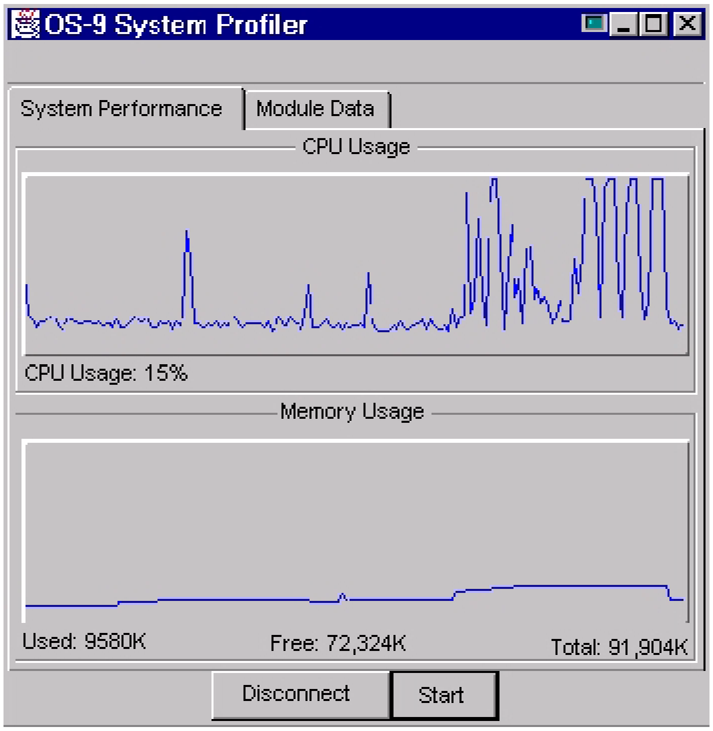
- source and assembly-level breakpoints
- display and change registers
- view locals
- watchpoints
- directly view and change memory
- stack back-tracing
- easy to use interface
- system and process level debugging

You can also use the stand-alone version of the Debugger. This version allows you to debug multiple processes or threads.

Profiler

The Hawk Profiler is used to examine the memory and CPU usage of processes running on an OS-9 target. It can show overall system statistics or module specific statistics, as shown in Figure 1-2.

Figure 1-2. The Profiler Main Window



CPU Usage: 15%

Used: 9580K Free: 72,324K Total: 91,904K

Ultra C/C++ Compiler

The Ultra C/C++ compiler uses state-of-the-art optimization techniques to obtain the maximum performance from your applications. While most compilers optimize your application on a file by file basis, Ultra C/C++ can see and optimize your application, along with its libraries.

Also available in Hawk is the Tools.h++ class library from Rogue Wave. This internationalized C++ foundation class library provides you with 120 reusable classes, including sets, bags, sorted collections, strings, linked lists, dates and times, and extensible virtual streams for persistence.

2 Creating Hawk Projects

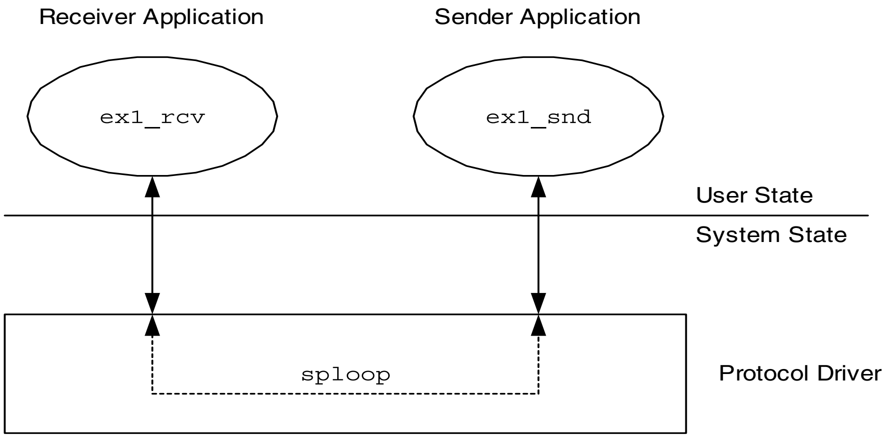
This chapter will teach you to create and modify a Hawk project. Before proceeding, be sure you meet the following requirements:

- You have installed Microware OS-9 software onto your host system
- You have connected your target system to your host system
- You have created an OS-9 ROM image and transferred it to the target system
- You have booted your system to the mshell prompt ($)
- Your target hardware has networking capabilities

The Example Applications

This chapter uses the sploop example to illustrate the process of creating and modifying a Hawk project. The sploop example consists of a sending application and a receiving application that uses the SoftStax network emulation driver (sploop) to send and receive a hello world message. Figure 2-1 outlines the sploop example.

Figure 2-1. Modules Used in the Tutorial



The sploop example consists of the following modules:

- Two applications (ex1_snd, ex1_rcv) that will run as separate processes
- One device driver (sploop)
- Two descriptor modules (loopc0, loopc1)

The required modules are included in the Enhanced OS-9 software package.

Refer to *Using SoftStax* for more information about sploop.

Create and Modify a Hawk Project

This section describes creating and modifying a Hawk project. During this process, you will complete the following tasks:

- Create a Project Space and Project
- Configure the Hawk Project
- Build the Module
- Add a Dependency

Create a Project Space and Project

Before a project can be built, a project space must be created to hold it. This section describes how to create the project space and add a project to it.

Step 1. From the Hawk window, select `Project -> Project Space -> New`. The **Create a New Project Space** dialog box appears.

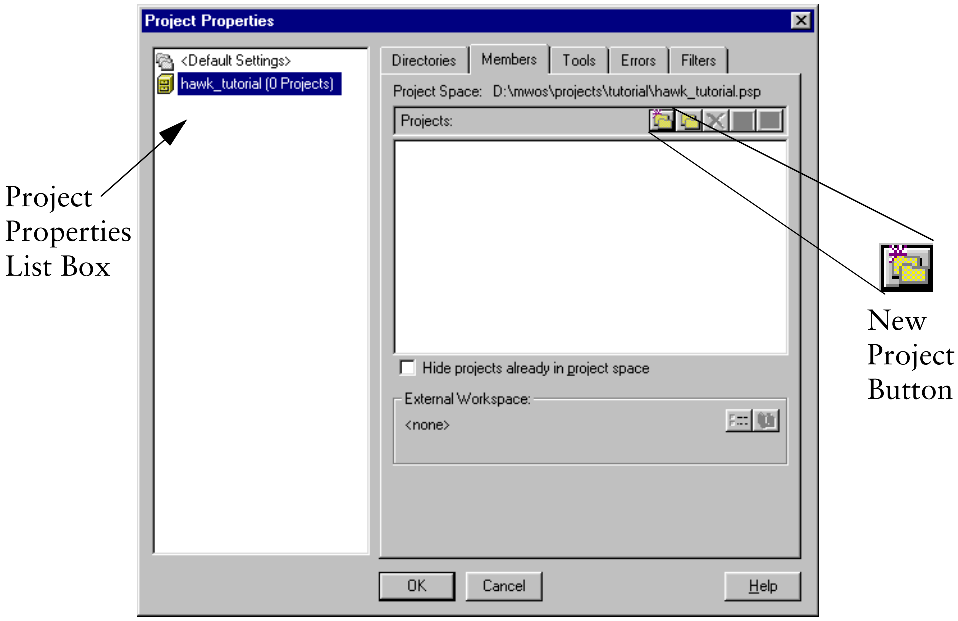
Step 2. From this dialog, enter the file name for your project space. For the purpose of this tutorial, enter the following path:

```
<drive>:\mwos\PROJECTS\HAWK_TUTORIAL\hawk_tutorial.psp
```

Click `OK`. Hawk creates the `PROJECTS` and `HAWK_TUTORIAL` folders automatically, along with a `hawk_tutorial.psp` file (the project space).

Step 3. Once you click the **OK** button, the **Project Properties** dialog box appears.

Figure 2-2. Project Properties dialog box



Project Space: D:\mwos\projects\tutorial\hawk_tutorial.psp

Step 4. Select the `Add New Project to project space` button (illustrated in the figure above). The **Add New Project to Project Space** dialog box appears.

Step 5. Enter `hawk_tutorial` (the name of the project) in the **Filename** field. It is not necessary to enter the full path because the current directory is correct.

Note that the name `hawk_tutorial` can be used for both the project and the project spaces because the extensions are different. (Projects end in .pjt and project spaces end in .psp).

Step 6. `Click OK`. The new project, `hawk_tutorial`, appears in the **Project Properties** list box as part of the `hawk_tutorial` project space.

Step 7. `Click OK` to dismiss the Project Properties dialog box.

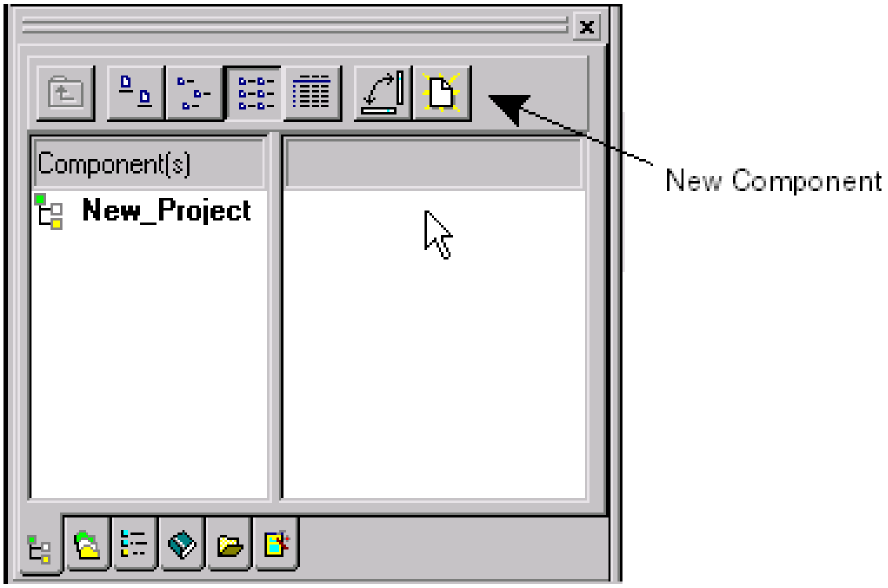
Creating a New Component for your Project

Once the project and project space have been created, you need to create a new component. A component is a grouping of files with unique settings. Although not all components create an output, most components build binary objects such as libraries, descriptors, or modules. The following table lists the valid component types:

Type	Output	Builder
User State Module	`Module`	Ultra C/C++
System State Module	`Module`	Ultra C/C++
Collection	`n/a`	n/a
Descriptor	`Module`	Ultra C/C++ or Editmod
Driver	`Module`	Ultra C/C++
File Manager	`Module`	Ultra C/C++
I-Code Library	`*.i or *.il`	Ultra C/C++
O-Code Library	`*.l`	`libgen`

Step 1. To create a new component, select the `New Component` button on the right side of the **Project Manager** window (illustrated in the figure below).

Figure 2-3. The New Component Button



Step 2. The **Create New Component** dialog box appears. Projects consists of multiple components. For this example, begin with one component, the `receiver` process. Type the following into the component dialog box:

Name: `receiver`

Description: `receiver process for the sploop (example 1) sender/receiver application`

Chip: <`Processor Name`>

Type: `User-State Program`

Psect File: This text box can be left blank. The Ultra C/C++ compiler will use the correct psect file for the executive option mode in use.

Step 3. Click Next>> to display the **Units** dialog box. Components consists of units, which can be library, header, or source files.

Step 4. At the Look in menu item, browse to the following location:

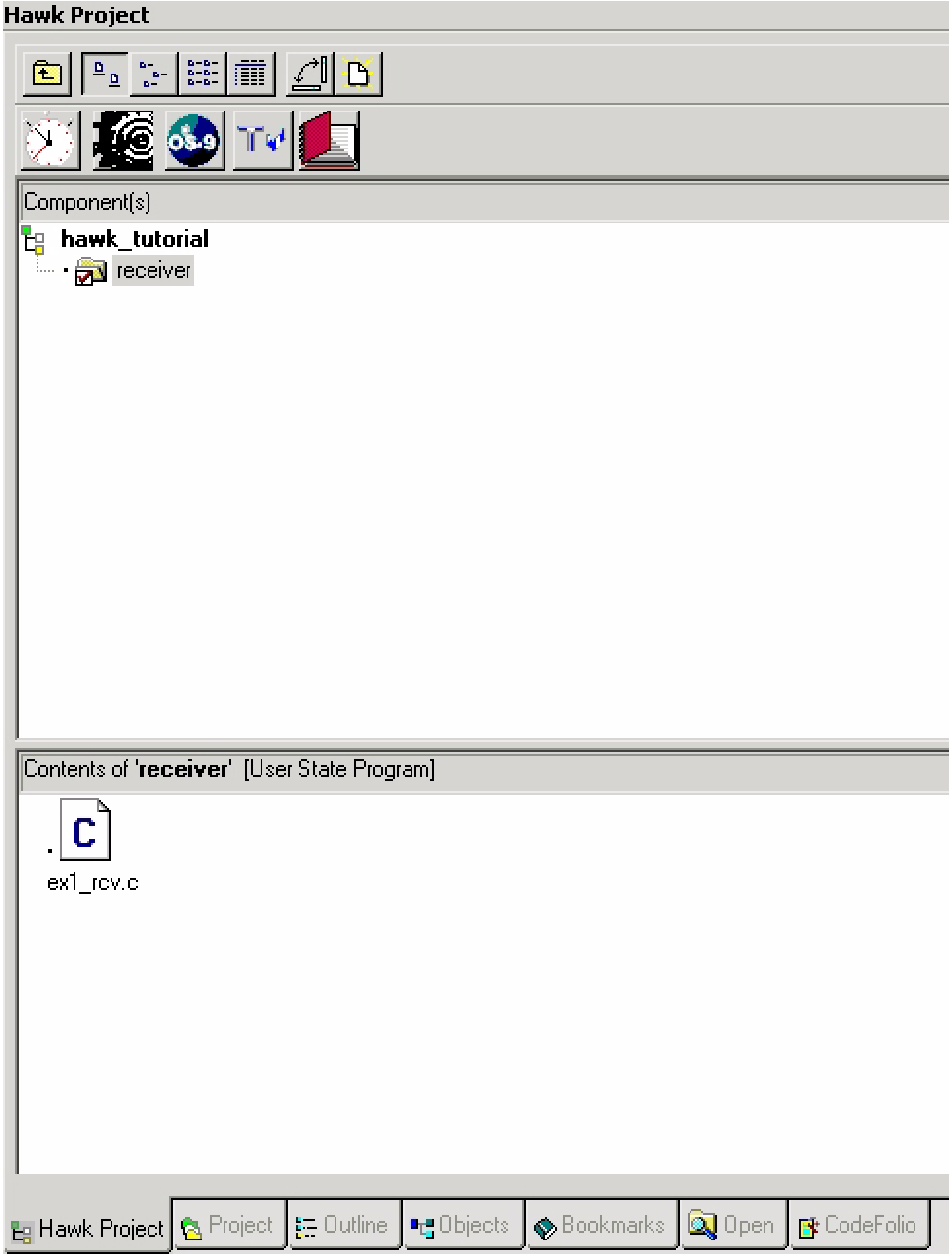
```
<mwos>\SRC\SPF\EXAMPLES\EXAMPLE1
```

Step 5. Select the ex1_rcv.c file and add by clicking the Add Selected Unit(s) button (the down arrow above the **Added Units** list box). The ex1_rcv.c full path list should now appear in the **Added Units** list box.

Step 6. Leave the **Generate Dependency Information** check box selected and click the Finish button. The Generating Dependencies dialog box appears while dependency information is created.

When complete, the Components frame should have a folder called receiver, and the **Contents** frame should contain a file called ex1_rcv.c (as illustrated in the figure below).

Figure 2-4. Results of Generated Receiver Component



Step 7. Save the project by selecting `Project -> Save`.

If you select a different processor for a component, the component settings override the project settings only for this component.

The component name is a module name by default. If the component name contains a space, Hawk inserts an underline for the module name. Also, if you have more than one component to enter, you must first create the project and add more components later.

More on Units

A unit is a single file which is added to a component. Since a unit is a file, it has an extension that identifies its type. Hawk recognizes the following types of file extensions:

Extension	Type	Builder	Viewer
`*.c, *.cpp, *.cxx`	Ultra C++ Source File	Ultra C++	text editor
`*.r`	Relocatable object file (ROF)	Ultra C++	`rdump`
`*.i, *.il`	I-Code file or library	Ultra C++	`idump`
`*.l`	O-Code library	Ultra C++	`libgen`
`*.a`	Assembler Source File	Ultra C++	text editor
`*.des`	Descriptor File	`editmod`	text editor
`*.mak`	Makefile (any type)	`OS9make`	text editor
* .	OS-9 Module	n/a	`ident`

Configure the Hawk Project

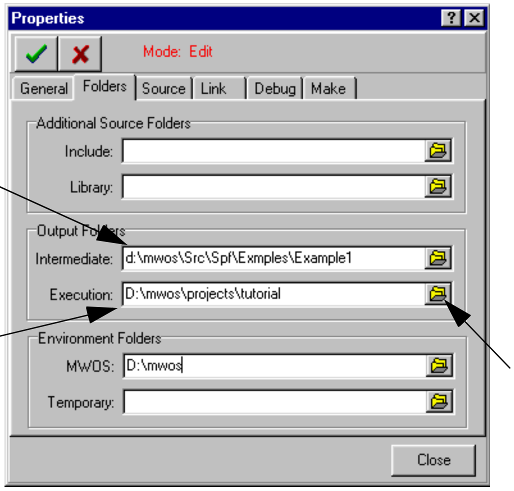
Search Paths

The next task is to configure your project to use correct search paths for libraries, header files, and storing intermediate and executable modules. Hawk knows the location of the default header files and libraries, but for the purpose of this tutorial, proceed through the following steps to learn how to find these paths manually.

Step 1. Select the `Project -> Properties` menu item. The **Properties** window for `Hawk_tutorial` is displayed. The `General` tab should be pre-configured with the default chip that was selected when the project was created. If it is not, scroll through the Chip drop-down menu and select the appropriate processor.

Step 2. Select the `Folders` tab. This tab enables you to add additional include files, libraries, and the destination for intermediate and executable modules.

Figure 2-5. Properties Dialog



Intermediate: d:\mwos\Src\Spf\Exmples\Example1

Execution: D:\mwos\projects\tutorial

MWOS: D:\mwos

Enter the folder location for the intermediate output for the project.

Enter the folder location for the execution output for the project.

Click on the folder icon to use the browser

Under the **Output Folders** section, add the following intermediate path by selecting the green plus sign, as illustrated in Figure 2-6. (Double or single-clicking will not add the path to the list.)

```
<drive>:\mwos\SRC\SPF\EXAMPLES\EXAMPLE1
```

Step 3. Save your change by selecting the green check mark in the upper left corner of the screen (illustrated in the Figure 2-5).

Execution Search Path

This section continues from the steps performed in the previous section.

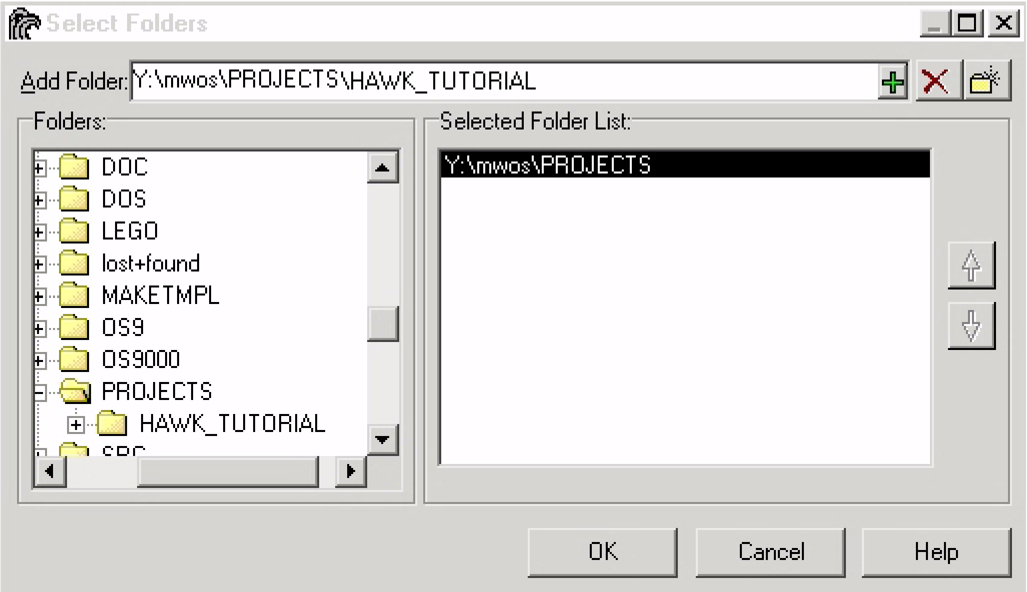
Complete the following steps in the **Properties** dialog to configure the **Execution** search path.

Step 1. Under the **Output Folders** section, click the `Browse` icon to the right of the **Execution** text box. The **Select Folders** dialog box appears.

Step 2. Select the `PROJECTS` folder on the left. The resulting path in the **Add Folder** text box should now appear as: `<drive>:\mwos\PROJECTS\HAWK_TUTORIAL`

Step 3. Select the green plus sign (illustrated in the figure below) to add this path to the **Selected Folders List**. (Double or single-clicking will not add the path to the list.)

Figure 2-6. Select Folders Dialog



Add Folder: Y:\mwos\PROJECTS\HAWK_TUTORIAL

Selected Folder List: Y:\mwos\PROJECTS

Step 4. Click `OK` to close the Select Folders dialog box.

Step 5. Save your change by selecting the green check mark in the upper left corner of the screen (illustrated in the Figure 2-5).

Source Options

This section continues from the steps performed in the previous section.

Complete the following steps in the **Properties** dialog to configure the source file options in the Properties dialog box:

Step 1. Select the `Source` tab.

Step 2. Select either the C or C++ language.

Step 3. Specify how closely the Compiler should follow ANSI standards.

Step 4. Save your change by selecting the green check mark in the upper left corner of the screen (illustrated in the Figure 2-5). Secect the `Close` button to dismiss the dialog.

Build the Module

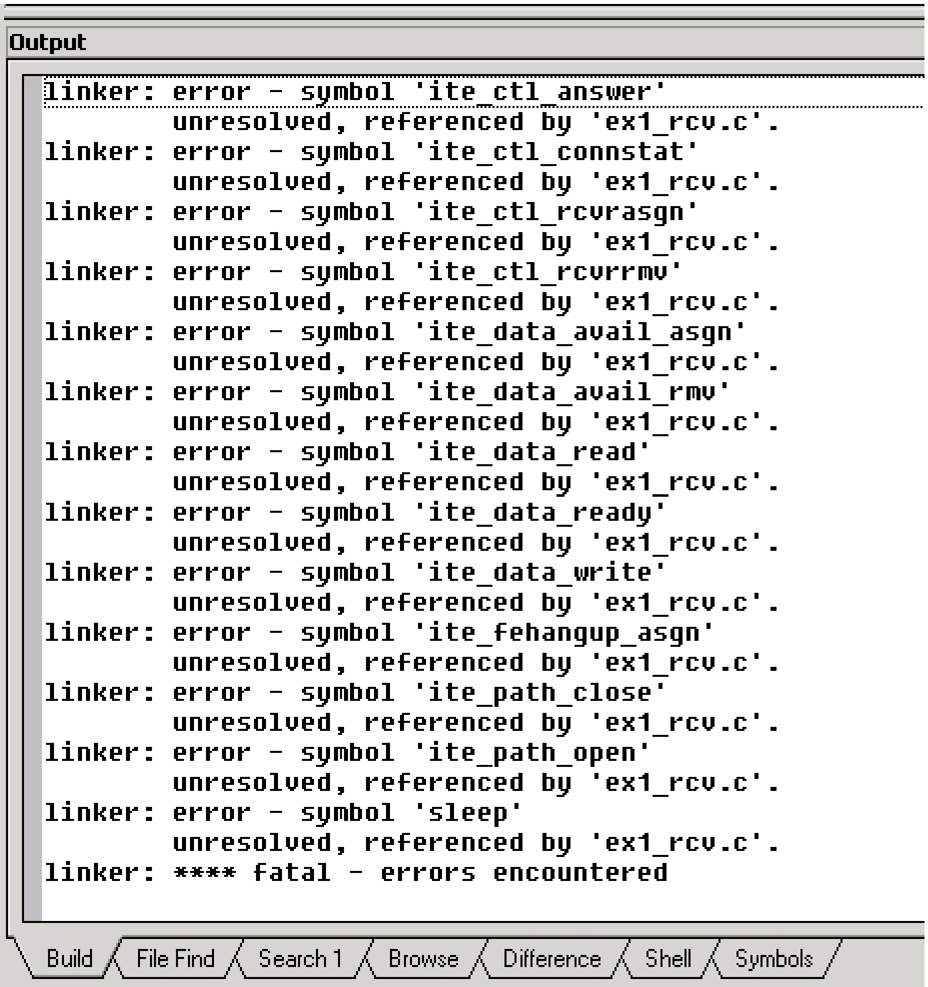
Complete the following steps to build your `Receiver` module. This section describes how to start the build and how to include a pre-compiled library.

Step 1. Under the `Project` menu, select `Build`.

A `Build Complete` dialog box appears and shows the progress of the build. It ends this build by stating that there were errors. These errors are shown in the `Build` tab window (refer to the figure below).

The linker states that there are many unresolved `ite_xxx` calls. The errors occur because the receiver process uses the `item.l` library, which was not included automatically. Hawk recognizes default OS-9 libraries and includes them automatically, but it does not automatically resolve any calls to the networking or graphics I/O system libraries.

Figure 2-7. Build Results with Errors

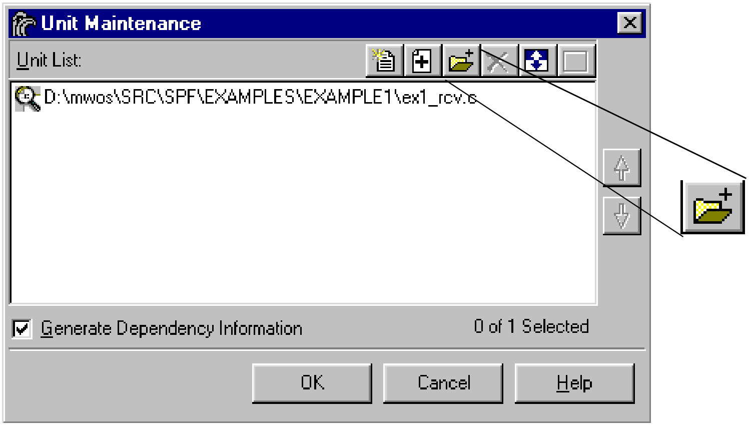


Output

```
linker: error - symbol 'ite_ctl_answer'
        unresolved, referenced by 'ex1_rcv.c'.
linker: error - symbol 'ite_ctl_connstat'
        unresolved, referenced by 'ex1_rcv.c'.
linker: error - symbol 'ite_ctl_rcvrasgn'
        unresolved, referenced by 'ex1_rcv.c'.
linker: error - symbol 'ite_ctl_rcvrrmv'
        unresolved, referenced by 'ex1_rcv.c'.
linker: error - symbol 'ite_data_avail_asgn'
        unresolved, referenced by 'ex1_rcv.c'.
linker: error - symbol 'ite_data_avail_rmv'
        unresolved, referenced by 'ex1_rcv.c'.
linker: error - symbol 'ite_data_read'
        unresolved, referenced by 'ex1_rcv.c'.
linker: error - symbol 'ite_data_ready'
        unresolved, referenced by 'ex1_rcv.c'.
linker: error - symbol 'ite_data_write'
        unresolved, referenced by 'ex1_rcv.c'.
linker: error - symbol 'ite_fehangup_asgn'
        unresolved, referenced by 'ex1_rcv.c'.
linker: error - symbol 'ite_path_close'
        unresolved, referenced by 'ex1_rcv.c'.
linker: error - symbol 'ite_path_open'
        unresolved, referenced by 'ex1_rcv.c'.
linker: error - symbol 'sleep'
        unresolved, referenced by 'ex1_rcv.c'.
linker: **** fatal - errors encountered
```

Step 2. Click OK to dismiss the Build Complete dialog box.

Step 3. Right click on the Receiver folder in the Components frame and select Unit Maintenance. The Unit Maintenance dialog appears.



Unit Maintenance

Unit List:

D:\mwos\SRC\SPF\EXAMPLES\EXAMPLE1\ex1_rcv.c

☑ Generate Dependency Information

0 of 1 Selected

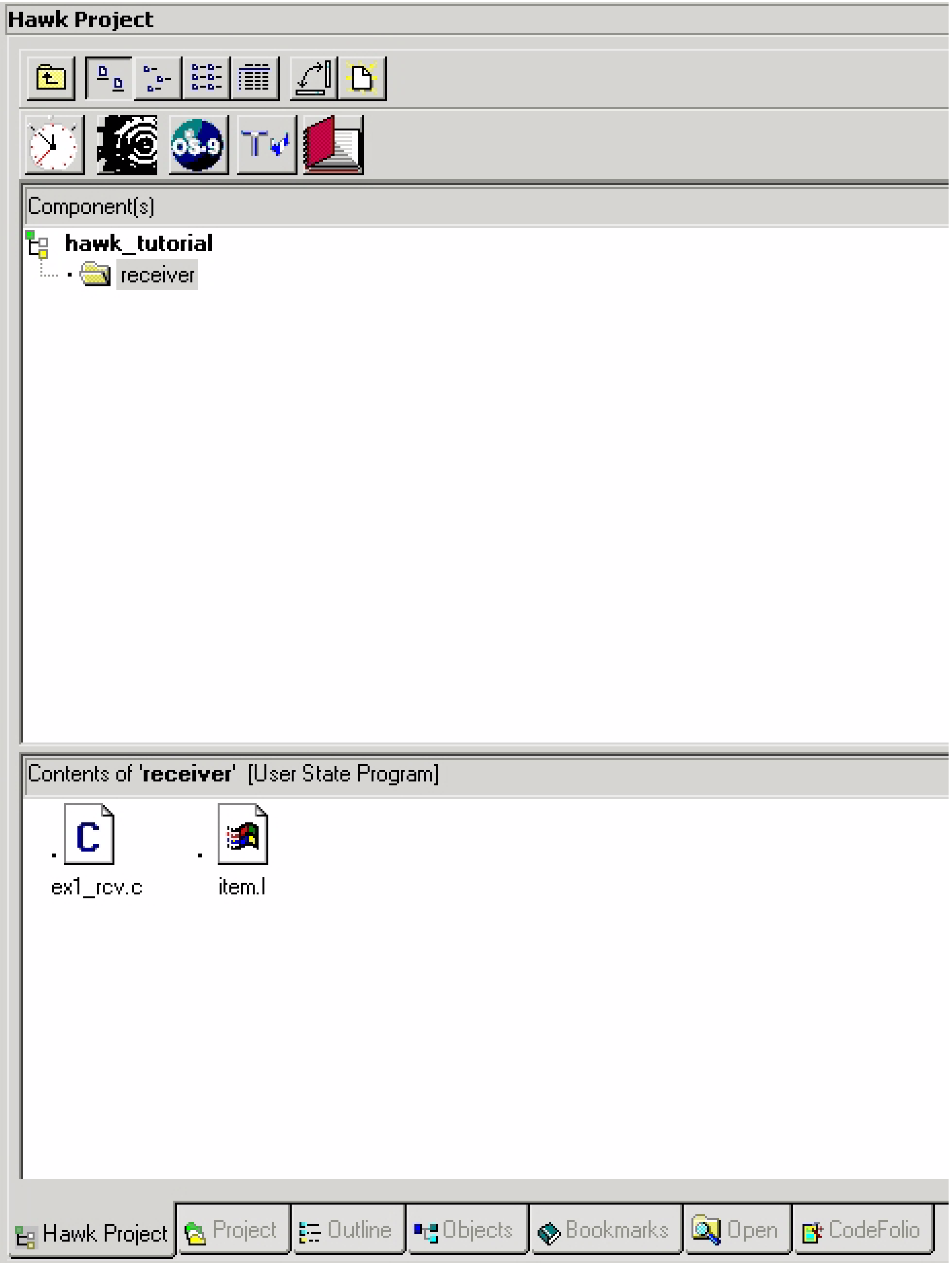
Add Existing Units button.

Step 4. Click the `Add Existing Units` button (illustrated in the figure above). The **Select One or More Files to Add to Project** dialog box appears.

Step 5. In the **Files of type** drop-down menu, select `O-Code Files (*.o, *.l)`. From the **Look in** box, browse to `<mwos>\OS9000\<processor>\LIB`

Step 6. Select the `item.l` file and click the **Open** button. You should now see the `item.l` file included in the **Unit Maintenance** dialog box.

Step 7. Leave the **Generate Dependency Information** check box selected and click `OK`. The `item.l` file should now be included in the **Contents of 'receiver'** pane next to the `ex1_rcv.c` file (illustrated in the figure below).



Step 8. Rebuild the project by selecting `Project -> Build`. There is now one unresolved error in your build. Proceed through the next section to resolve this error.

Add a Dependency

In the previous section, the build resulted in one unresolved error regarding the `sleep()` function. The `sleep()` function resides in the `sys_clib.l` library. In this case, `sys_clib.l` is included as a dependency instead of as a unit. All of the RadiSys-provided objects without associated sources must be included this way.

Step 1. Click `OK` in the Build Complete dialog box to dismiss it.

Step 2. Select `Project -> Properties`.

Step 3. Select the `Link` tab.

Step 4. In the **O-Code Libraries** box, select the `Browse` button. The **Library Selection** dialog appears. Select the `Add Existing Units` button as you did in the previous section, and navigate to the following location:

`mwos\OS9000\<processor>\LIB\sys_clib.l`.

Step 5. Click on `sys_clib.l` and click `Open` to include `sys_clib.l` as an O-Code library. Select `OK` to dismiss the **Library Selection** dialog.

Step 6. Click the green check mark button in the upper left hand corner of the Properties dialog to save the O-Code library changes.

Step 7. Click on the Click `Close` to dismiss the **Properties** dialog.

Step 8. Rebuild the project again using `Project -> Build`. The build should complete without errors.

Adding the Sender Component

Add the `sender` component to the Hawk project.

Step 1. Select the `New Component` button in the Hawk Project window (illustrated in Figure 2-3).

Step 2. Complete the fields in the **Create New Component** dialog for the sender process.

Name: `sender`

Description: `sender process for the sploop (example 1) sender/receiver application`

Chip: `<processor>`

Type: `User-State Program`

Step 3. `Click Next` to display the **Units** dialog and add the `ex1_snd.c` and `item.l` files in the same manner you added units to the `receiver` component in a previous section.

- The `ex1_snd.c` file resides in the following location:
  `<drive>:\mwos\SRC\SPF\EXAMPLES\EXAMPLE1\`
- The `item.l` file resides in the following location:
  `<drive>:\mwos\OS9000\<processor>\LIB\`

Step 4. Leave the **Generate Dependency Information** check box selected and select `Finish`.

Step 5. Rebuild the project by selecting `Project -> Build`.

You have now successfully built a project. The next step in this tutorial is to debug the project. Refer to the next chapter for the basic steps involved in user-state application debugging with Hawk.

3 Hawk Application Debugging

This chapter covers user-state debugging. The process model used by OS-9 consists of two environments: user-state and system-state.

User-state is the execution environment for application processes. Generally, user-state processes do not deal directly with the specific hardware configuration of the system.

System-state is the environment in which OS-9 system calls and interrupt service routines are executed. System-state routines often deal with the physical hardware present on a system.

This chapter will cover debugging user-state applications.

Preparing to do Application Debugging

Before you can do application debugging with Hawk. The following must be true:

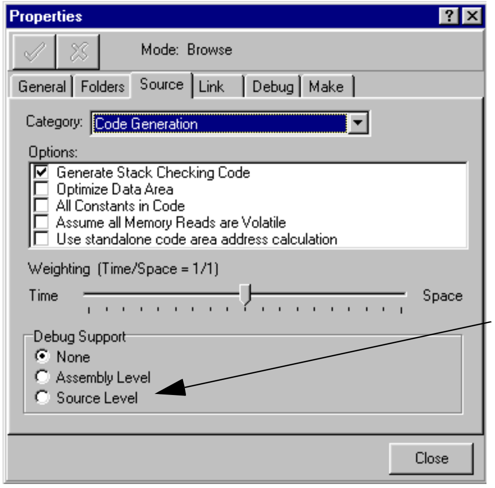
- SoftStax is included into the bootfile
- The sploop protocol driver and descriptors are loaded onto the target
- Timeouts must be increased for the receiver and sender processes
- Source level debugging must be enabled in Hawk

If you followed the board guide for your version of OS-9, you should already have a ROM image with Softstax enabled. It you do not have SoftStax enabled, you need to rebuild your OS-9 ROM image.

Configuring Debug Support for your Project

Step 1. Select `Project -> Properties` to bring up the Properties dialog box.

Step 2. In the **Source** tab, select the `Code Generation` category. In the **Debug Support** field, check the `Source Level` radio button to enable source level debugging.



Category: Code Generation

Options:
☑ Generate Stack Checking Code
□ Optimize Data Area
□ All Constants in Code
□ Assume all Memory Reads are Volatile
□ Use standalone code area address calculation

Weighting (Time/Space = 1/1)
Time — Space

Debug Support
● None
○ Assembly Level
○ Source Level

Check the Source Level button for the type of debug support you want.

Step 3. Click the green check mark button to save your changes, if necessary.

Step 4. Click the `Close` button to dismiss the **Properties** dialog.

Step 5. Save the project by selecting `Project -> Save`.

Increasing the Timeouts

For application debugging, you must increase the timeouts of `sender` and `receiver`, by completing the following steps:

Step 1. Open the `ex1_snd.c` file from the sender component. Perform a search for the following lines:

```
connect_npb.ntfy_timeout = 10;

fehangup_npb.ntfy_timeout = 10;

datavail_npb.ntfy_timeout = 10;
```

On each of these lines, change the timeouts to `100`.

Step 2. Now open the `ex1_rcv.c` file and perform a search for the following lines:

```
incall_npb.ntfy_timeout = 50;
fehangup_npb.ntfy_timeout = 10;
datavail_npb.ntfy_timeout = 10;
```

Change the timeouts in these lines to `100` as well.

Step 3. Save and close the files.

Step 4. Rebuild the project using `Project -> Build`.

Step 5. Save the project by selecting `Project -> Save`.

Application-Level Debugging Using Hawk

This section describes application-level debugging. Application-level debugging is also known as user-state debugging.

Setting Up the Debugger

The procedure in this section assumes that your reference board is not running.

Step 1. Apply power to your reference board.

Step 2. Click `Connect` in the serial window. Use the default Com Port Options. The command prompt displays once the board has booted.

If the Serial window is not visible, do the following steps to open it:

1. Select Tools->Customize->Toolbars
2. In the Toolbar Customization dialog box, select Serial.
3. Select the Visible check box and click Close.

Step 3. Type the following command in the serial window:

```
spfndpd <>>>/nil &
```

Step 4. The `sploop` example requires three modules loaded onto your system. The three modules are the `sploop` protocol driver module, the `loopc0` descriptor module, and the `loopc1` desciptor module.

You can load these modules in one of two ways: individually through Hawk, or by adding them to a bootfile. For the purposes of this tutorial, you should load them individually through Hawk.

If you want to learn how to add your own modules to a bootfile see the Configuration Wizard's help file.

Step 5. Select `Target -> Load` to open the **Load Module** dialog box.

Step 6. Click on the **Browse** button and navigate to the directory in which the driver and descriptor modules reside. The directory can be found in the following location:

```
mwos\OS9000\<processor>\CMDS\BOOTOBJS\SPF
```

Step 7. Select the `loopc0` file and click `Open`.

Step 8. Click `Load` in the **Load Module** dialog. The module is loaded onto your reference board.

Step 9. Repeat steps 5-8 for `loopc1` and `sploop`.

Step 10. Load your newly created sender program by selecting `Target -> Load.` Browse to the `<drive>:\mwos\PROJECTS\HAWK_TUTORIAL` folder. Click on the sender module in the list box and click `Open` to select the sender program.

Step 11. Click `Load` to load the sender module on the target.

Step 12. Select `Debug -> Connect` on the top menu bar. The Connect dialog box appears.

Step 13. Click the `Fork` tab. The **Connect** dialog displays the target machine name (or IP address) in the **Target** text box.

Step 14. In the **Program** box, browse to the `receiver` module in the `<MWOS>\PROJECTS` folder and click `Open`.

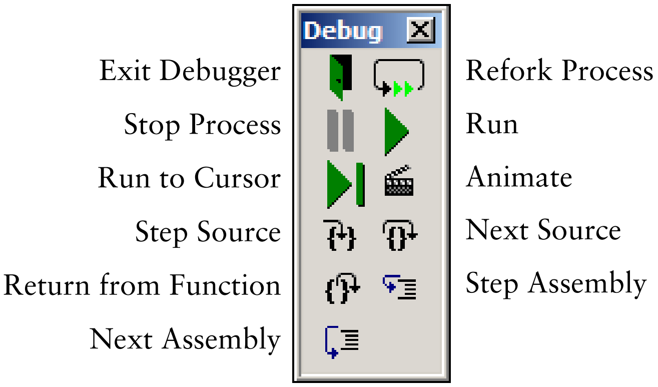
Step 15. Click `OK`. Follow the prompts to select `ex1_rcv.c`. `ex1_rcv.c` is at `mwos\SRC\SPF\EXAMPLES\EXAMPLE1`

Step 16. Click `Open` to select `ex1_rcv.c` source file.

Using the Debugger

The next task is to step through the code. At this point the `ex1_rcv.c` source code is displayed in the Source Code window with a yellow-outlined arrow pointing to the `main()` line. Also, the Debug toolbar is displayed. Complete the following steps to use the debugger.

Figure 3-1. Debug Toolbar



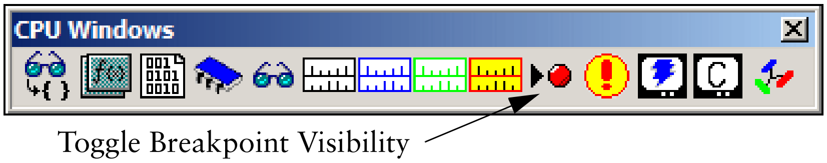
Step 1. Click the `Step source` button to scroll through the code. The output of `receiver` goes to the **Process I/O** window. The Process I/O window appears when the process creates output.

As you scroll through the code notice a large `while()` loop. The receiver sleeps until it gets a signal. It then wakes up and checks for incoming calls. If there is an incoming call, it reads the data, prints what it gets, then returns `Message Received` to the sender.

Step 2. Next you will need to set a breakpoint at the line that contains the `if (incall_flag)` statement. To do this, complete the following steps:

1. Open the **Breakpoint** window by selecting the `Toggle Breakpoints Visibility` button on the CPU Windows toolbar (illustrated in the figure below).

Figure 3-2. CPU Windows Toolbar



1. Right-click in the **Breakpoint** window and select `Insert` from the pop-up menu. The **Add Breakpoint** window appears.
2. From the **Add Breakpoint** window, set the breakpoint type to `Source`.
3. Select the file selector button the right of the File text box, and navigate to the source file you are debugging. In this case the source file is `ex1_rcv.c` and is located in `mwos\SRC\SPF\EXAMPLES\EXAMPLE1`.
4. Using your mouse, highlight the start of the line you want to set a breakpoint on, then right-click on the highlight and select `Toggle Breakpoint` from the menu.

If you cannot see line numbers in your code, select `Customize -> Views` from the Hawk menu. On the **General** tab, select the `View Line Numbers` check box.

5. Click `OK`. The dialog box closes and the breakpoint is set on the line specified as indicated with a red dot.

To add breakpoints to your code more quickly, simply highlight the line on which you want to set a breakpoint, then right-click on the highlight and select `Toggle Breakpoint` from the pop-up menu.

Step 3. Select the `Run` button, indicated by a green arrow on the **Debug** toolbar, to allow the `Receiver` to free-run. In looking through the source code, you will notice that the `receiver` application sleeps indefinitely until a signal wakes the process.

Step 4. Type `Sender` in the serial window. A yellow arrow should appear to the right of the red ball where you set the breakpoint.

Step 5. Exit the Debugger by selecting `Debug -> Exit Debugger` from the main window.

If this option is not available or the Hawk interface appears locked, perform a `telnet` command to the target and terminate the `receiver` process by typing `procs`. From here, determine the process ID of the `receiver` and type `kill <number>`.

You have now completed the basics of application debugging using Hawk.

4 Hawk System-State Debugging

This section describes debugging system-state modules. As mentioned earlier, system-state debugging is needed only for device driver or other hardware dependent code. The example we will use in this section will revolve around debugging a RAM disk driver. During this debugging process, you will complete the following tasks:

- Create the Driver Project
- Prepare the Target for Debugging
- Attach to the System
- Attach to the Module

Create the Driver Project

The first task is to create a new project space and project to hold the driver. The process is very similar to building an application module. It is also possible to add driver projects to application projects, but it is not necessary for the current example.

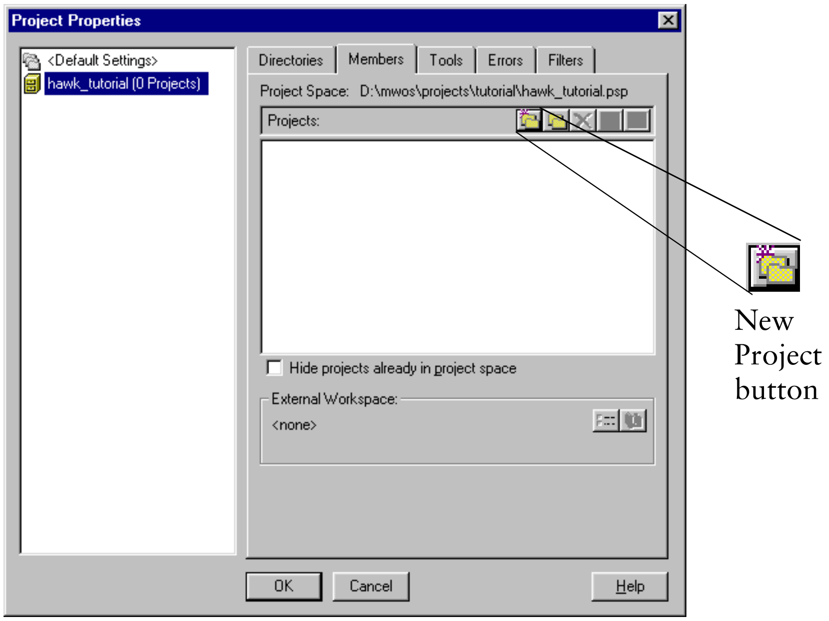
Step 1. From the Hawk window, select `Project -> Project Space -> New`. The **Create a New Project Space** dialog box appears.

Step 2. In the **Create a New Project Space** dialog box, enter the file name for your project space. For this tutorial, enter the following path:

```
<drive>:\mwos\PROJECTS\HAWK_TUTORIAL\sys_state_tutorial
```

Step 3. Click `OK.` Hawk creates a project space file called `sys_state_tutorial.psp`, and the **Create a New Project Space** dialog box is replaced by the **Project Properties** dialog box.

Figure 4-1. Project Properties dialog box



Project Space: D:\mwos\projects\tutorial\hawk_tutorial.psp

☐ Hide projects already in project space

<none>

New Project button

Step 4. Click on the `New Project button.` The **Add New Project to Project Space** dialog box appears.

Step 5. Enter `sys_state_tutorial` (the name of the project) in the **Filename** field.

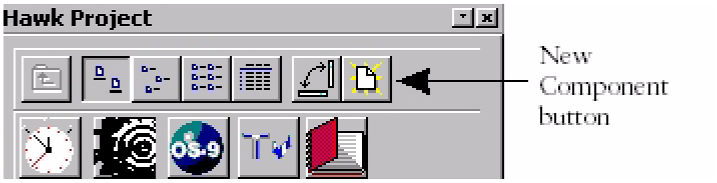
Step 6. Click `OK`. The new project, `sys_state_tutorial` appears in the **Project Properties** list box as part of the `hawk_tutorial` project space.

Add the Driver Components to the Project

Once the project space and project have been created, the driver components need to be added to the project.

Step 1. Click on the **New Component** button on the right side of the **Project Manager** window.

Figure 4-2. New Component Button



Step 2. In the component window, enter the following information:

Name: `ram`

Description: `RAM disk device driver`

Chip: `<Processor Name>`

Type: `Driver` (the `psect` will automatically change to `drvstart.r`.)

Step 3. Click `Next>>` . The **Units** window appears.

Step 4. At the **Look in** menu item, browse to the following location:

`<mwos>\OS9000\SRC\IO\RBF\DRVR\RAMDRVR`

Step 5. Select the files below and add them to the project by clicking the `Add Selected Unit(s)` button (the down arrow above the **Added Units** list box).

`drvrstat.c`

`init.c`

`main.c`

`misc.c`

`move.c`

`parity.c`

`read.c`

`stat.c`

`term.c`

`write.c`

Step 6. Next, change the **Files of Type** box to read **Header Files** (***.h, *.hpp**) and add the following files:

`prototyp.h`

`ram.h`

Step 7. Change the **Files of Type** box again to read **All Files** and add the following file:

`makefile`

Step 8. De-select the **Generate Dependency Information** check box. Hawk should not generate dependencies at this point because the existing makefiles are going to be used to generate the driver modules.

Step 9. Click `Finish`. A new `ram` component appears in the project window.

Step 10. Save the project by selecting `Project -> Save`.

Configure the Driver Makefiles

The next task is to configure the makefile to properly build the debugging version of the RAM disk driver. Complete the following steps:

Step 1. Right click on the `makefile` file and select `Properties` from the pop-up menu.

Step 2. Select the `Make` tab and configure the menu as follows:

Make: `os9make -f%b%e`

Forced Make: `os9make -f%b%e clean all`

Step 3. Click on the green check mark button in the upper left corner to save the changes.

Step 4. Click `Close` to dismiss the **Properties** window.

Edit the makefile File

The next task is to edit the `makefile` file. Complete the following steps:

Step 1. Right-click on the `makefile` file in the **Component** window and select `Open` in the pop-up window. The file appears in the **Document** window.

Step 2. At the line `DEBUG   = #-g`, remove the comment character “#” in front of the `-g` option. The line should now appear as follows:

```
DEBUG   = -g
```

The `-g` option causes the Compiler to create a `ram.dbg` file that provides the symbol information map for source level debugging of device drivers.

Step 3. Save the file.

Step 4. Right click on the `makefile` file and select `Rebuild` from the pop-up menu. The debugging version of the `ram` driver is placed in the following directory:

```
mwos\OS9000\<PROCESSOR>\CMDS\BOOTOBJS\ram
```

Do not choose the component build by right clicking on the component to get the contextual menu. Be sure to build by clicking on the file. This will invoke the non-Hawk makefile.

Prepare the Target for Debugging

Before this example can be performed, the OS-9 bootfile image must have certain services added and other services disabled.

Step 1. Open the Configuration Wizard and perform the following tasks:

- Enable the RAM Disk
  (**Configure** -> **Bootfile** -> **Disk Configuration** -> **Ram Disk** tab)
- Disable SoftStax
  (**Configure** -> **Bootfile** -> **Network Configuration** -> **SoftStax Setup** tab)

- Enable Remote Ethernet Debugging
  (**Configure** -> **Coreboot** -> **Main Configuration** -> **Debugger** tab)
- Set your Ethernet IP address
  (**Configure** -> **Coreboot** -> **Main Configuration** -> **Ethernet** tab)
- Make sure that the **User-State Debugging** option is checked in the **Master Builder** window (to load `undpd`).

Step 2. Build the bootfile. For more information on bootfile image customization or saving your image, refer to your *OS-9 for <target> Board Guide*.

Step 3. Reboot your target. If you were debugging an application using the low-level debugger, you would run the `undpd` daemon at this point with the command line "`undpd <>>>/nil &`". Since we are debugging system-state code, we do not need to run `undpd`.

Prepare Hawk for Debugging

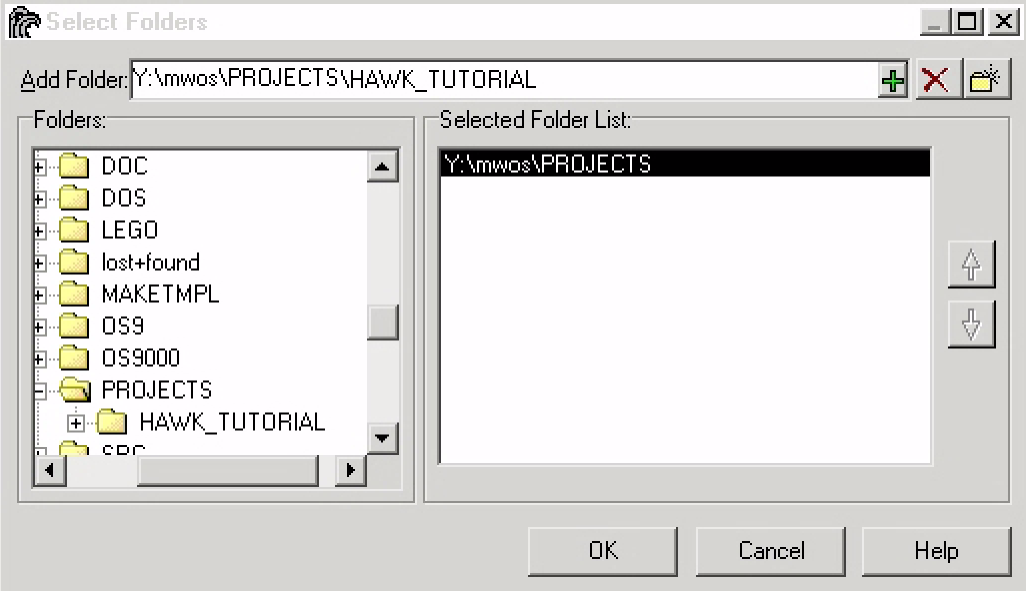
Perform the following tasks to set up the Hawk IDE for debugging.

Step 1. Select `Debug -> Options`.

Step 2. In the **Options** window, select the `Folders` tab. Boxes for inputting the source and object code search folder locations are displayed.

Step 3. Select **Browse** button next to the **Source Code** area. This displays the **Select Folders** window (shown in the figure below).

Figure 4-3. Select Folders Dialog



Add Folder: Y:\mwos\PROJECTS\HAWK_TUTORIAL

Selected Folder List: Y:\mwos\PROJECTS

Step 4. From this window, delete all of the currently selected folders, then browse to the following path and add it to the list by clicking on the green plus sign in the upper right corner).

```
mwos\OS9000\SRC\IO\RBF\DRVR\RAMDRVR
```

Step 5. Click `OK`.

Step 6. Now select the **Browse** button for the **Object Code** area. The Select Folders dialog appears again.

Step 7. From this window, delete all of the currently selected folders, then browse to the following path and add it to the list by clicking on the green plus sign in the upper right corner).

```
mwos\OS9000\<PROCESSOR>\CMDS\BOOTOBJS
```

Step 8. Select `OK`.

Step 9. Select `OK` once again to dismiss the **Options** dialog.

Attach to the System

The next task is to attach the debugger to the system. Complete the following steps:

Step 1. In the serial window, type `break`. This stops the target and allows you to set breakpoints in the system code.

Step 2. Return to the Hawk interface and select `Debug -> Connect` from the main menu.

Step 3. At the **Connect** window, select the `Attach` tab. Make sure the **System** type is highlighted.

Step 4. If it is not already there, enter your target's name in the **Target** field.

Step 5. Click `OK`. After a few minutes, the debugging environment appears.

Attach to the Module

The next task is to attach to the module you will test. Complete the following steps:

Step 1. Select `Debug -> Process -> Attach Module to Current` and highlight `Module` in the `Type` box.

Step 2. Type `ram` in the Module box.

Step 3. Click `OK`.

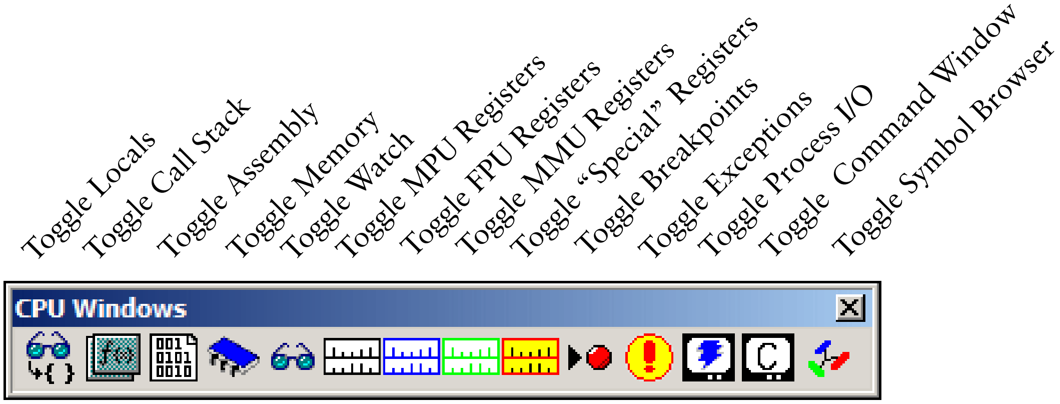
Step 4. Select `Debug -> View -> Browse Symbol`. This selection displays the symbol browser window. It should have the ram module icon and name in the window.

Step 5. Expand the `ram` icon and select the `read.c` file.

Step 6. Add a breakpoint in the `read.c` file. Breakpoints are set in the Breakpoints window.

Step 7. Click on the `Toggle Breakpoints Visibility` button on the Datawindows Toolbar to open the Breakpoint window.

Figure 4-4. CPU Windows Toolbar



Step 8. Right click in the Breakpoint window and select `Insert` from the contextual menu. The Add Breakpoint window appears.

Step 9. Leave the Breakpoint Type set to Source.

Step 10. Click on the file selector button to the right of the File text box and navigate to the source file being debugged. In this case, the source file is `read.c`, which is located in `mwos\OS9000\SRC\IO\RBF\DRVR\RAMDRVR`.

Step 11. Enter the line number where you want to place your breakpoint into the Line # spin box.

Step 12. Click `OK`. The dialog box closes and the breakpoint is set on the line specified as indicated with a red dot.

Step 13. Minimize the Symbol Browser window.

Step 14. Select the green arrow on the debugging bar to start the system. Once the green arrow is selected, the serial console should print out `***Warning*** - breakpoints halt timesharing.`

Step 15. Type `dir /r0` and you should see the breakpoint you set in the debugger window.

The system will halt and you can now step through the RAM driver source. As soon as you finish the step through process, the system will resume and you can either debug the driver further or stop debugging by selecting `Debug -> Stop`.

Once you disconnect from the Debugger, you will need to reset the system before you can reconnect.

5 Using Makefiles

This chapter describes building projects that run pre-existing makefiles. The following sections are included:

- Overview
- Running Makefiles in Hawk
- Makefile Example

For OS-9 SDK and Board-Level Solution Users:
The example used in this chapter creates a system-state module. If you are using the OS-9 SDK or Board Level Solution, you can follow these steps, but will have to use one of the user-state demo applications instead of the system-state example. For example, one of the MAUI® demos may be used for the project instead of the RAM disk driver.

Overview

Makefiles can be run very easily through Hawk. The RAM disk driver example in Chapter 4: System-State Debugging illustrates using a makefile in a Hawk project. This example is re-examined in this chapter with a different emphasis. Where the emphasis in Chapter 4 is on system-state debugging, this chapter will focus on the relationship between Hawk and the makefile.

OS-9 Makefiles

Many of the components of OS-9 are built using makefiles. OS-9 makefiles reside within the same directory as the component's source files and are either called `makefile` or identified with a `.mak` extension.

These makefiles set up a component's build by defining such items as the source files, the header files, and any library files. Usually, makefiles include other makefiles which control compiler and linker settings that are common to a number of components. Everything is defined in the current makefile or in one of the makefiles or templates referenced by the main makefiles.

Running Makefiles in Hawk

The following steps illustrate how to run makefile-built components within Hawk:

Step 16. Create a new project space and project.

Step 17. Add the source files, header files, and makefiles to the project in the **Units** dialog box.

Step 18. Deselect the `Generate Dependency` check box in the **Units** dialog box.

Step 19. Make sure Hawk is configured to run OS-9 make correctly. This is determined by right-clicking on the makefile icon, selecting properties, and selecting the make tab.

Step 20. Set any command line switch in the makefile as needed. For example, the RAM disk makefile has a macro called DEBUG, which is defined if the `-g` is not commented out. This creates a debug version of the driver.

Step 21. Invoke the makefile by right clicking on it and selecting `Build`. Selecting `Rebuild` performs a forced make.

Makefile Example

This example repeats the system-state debugging example from Chapter 4 with an emphasis on building the project and the relationship between Hawk and the makefile.

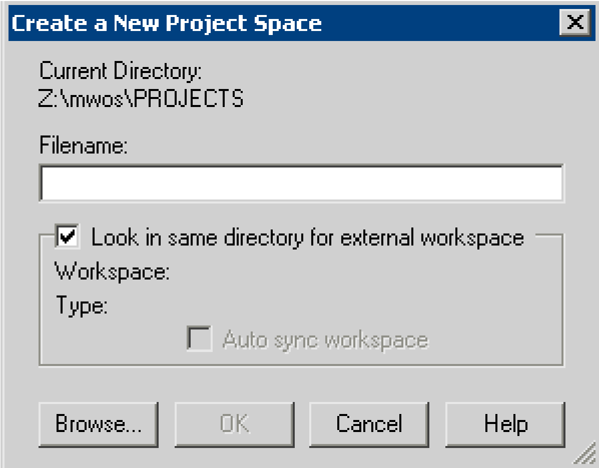
It is important to remember that the makefile will control the build process and information entered into Hawk may not always be used. In this case, the Hawk project is mainly a mechanism for organizing the component's files. The instances when the information entered in Hawk dialog boxes are not used will be noted at relevant points during the following procedures.

Creating the Project

The process of creating the project space and project is not affected by the makefile. Therefore, none of the information in the dialog boxes in this section is overridden by the makefile.

Step 1. From the **Hawk** window, select `Project -> Project Space -> New`. The **Create a New Project Space** dialog box appears.

Figure 5-1. Create a New Project Space



Current Directory:
Z:\mwos\PROJECTS

Filename:

☑ Look in same directory for external workspace

Workspace:

Type:

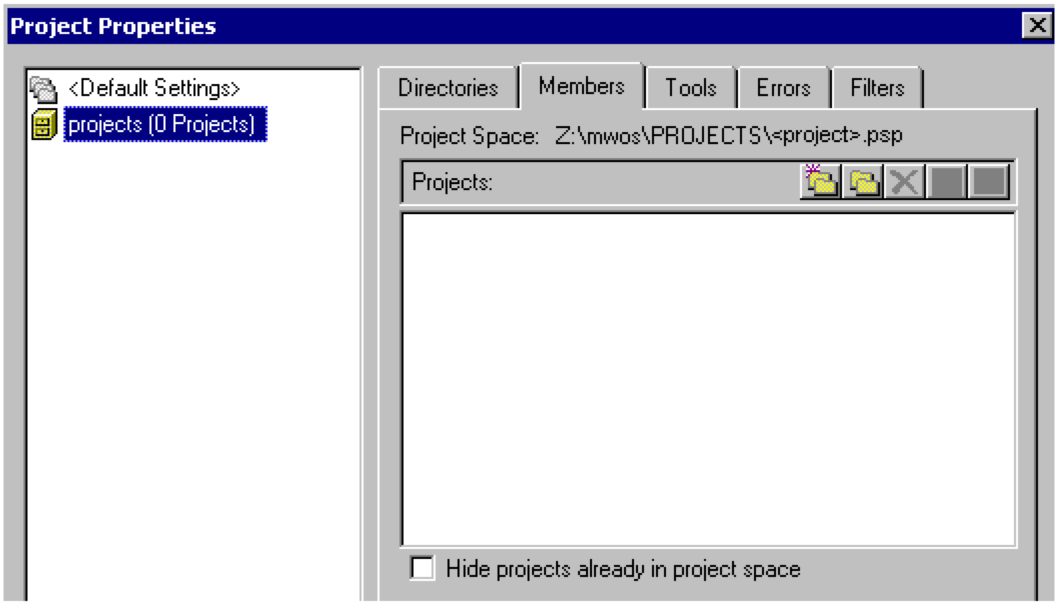
□ Auto sync workspace

Step 2. In the Create a New Project Space dialog, enter the file name for your project space. For this example, enter the following path:

```
MWOS\PROJECTS\RAM_disk_project_space\RAM_Project.psp
```

Step 3. Click `OK`. Hawk creates a project space file called `RAM_Project.psp`, and the **Create a New Project Space** dialog box is replaced by the **Project Properties** dialog.

Figure 5-2. Project Properties dialog



Project Space: Z:\mwos\PROJECTS\<project>.psp

Projects:

□ Hide projects already in project space

Step 4. Click the `New Project` button. The **Add New Project to Project Space** dialog appears.

Step 5. Enter `RAM_Driver` (the name of the project) in the **Filename** field.

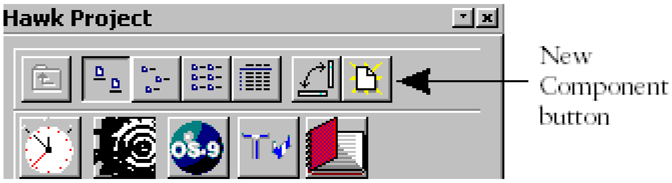
Step 6. Click `OK`. The new project, **RAM_Driver** appears in the **Project Properties** list box as part of the **RAM_Project** project space. Click `OK` to close the **Project Properties** window.

Add Driver Components to the Project

Once the project space and project have been created, the driver components need to be added to the project. The makefile will override some of Hawk settings in this section. The overridden settings are identified in the appropriate steps. The following procedure describes how to add the driver components to the project.

Step 1. Select the `New Component` button on the right side of the **Project Manager** window (as illustrated in Figure 5-3):

Figure 5-3. New Component button



Step 2. In the **Create New Component** window, enter the following information.

Name: `ram_disk`
Description: `RAM disk device driver`
Chip: `<Processor Name>`
Type: `Driver` (the `psect` will automatically change to `drvstart.r`)

The **Chip** and **Type** values are overridden by the makefile.

Step 3. Click `Next`. The **Units** window appears. At the **Look in** menu item, browse to the following location:

```
MWOS\OS9000\SRC\IO\RBF\DRVR\RAMDRVR
```

Do not move the source files, header files, or makefiles from their directory. These makefiles contain path information that would need to be updated if any related files are moved.

Step 4. Select the following files and add them to the project by clicking the down arrow above the **Added Units** block.

```
drvrstat.c   init.c
main.c       misc.c
move.c       parity.c
read.c       stat.c
term.c       write.c
```

Step 5. Change the **Files of Type** box to read `Header Files` (`*.h`, `*.hpp`) and add the following files:

```
prototyp.h
ram.h
```

Step 6. Change the **Files of Type** box to `All` and add the following file:

```
makefile
```

All of the files needed to build the driver are now included in the project.

Step 7. De-select the **Generate Dependency Information** check box. Hawk should not generate dependencies at this time because the makefile contains dependency information.

Step 8. Click `Finish`. A new component named `ram_disk` appears in the project window. Save the project by selecting `Project -> Save`.

Configure the Driver Makefiles

Complete the following steps to verify that Hawk is set to run `os9make` properly:

Step 1. Right-click on `makefile` and select `Properties`.

Step 2. Select the `Make` tab and configure the menu as follows:

Make: `os9make -f%b%e`

Forced Make: `os9make -f%b%e clean all`

For more information about `os9make` and its options, refer to the *Utilities Reference* manual.

Step 3. Select the green check mark button to save the changes.

Step 4. Click `Close` to exit the **Properties** window.

Edit the Makefile File

You may find it necessary to apply options in a makefile before building the component. In this example, you will build the debugging version of the RAM disk driver. The `makefile` file is edited to apply the debug option. Complete the following steps to set up the makefile to build the debugging version of the driver:

Step 1. Double-click the makefile in the **Component** window or select `File -> Open` from the **File** menu.

Step 2. At the macro named `DEBUG=`, remove the comment character "`#`" in front of the `-g` option. The `-g` option causes the compiler to create a `ram.dbg` file that provides the symbol information map for source level debugging of device drivers.

Step 3. Right click on the `makefile` file and select `Rebuild` to do a forced make. The debugging version of the `ram` driver is built and placed in the following directory: `MWOS\OS9000\<processor>\CMDS\BOOTOBJS`.

⚠ Do not choose **Build** or **Rebuild** from the **Project** menu or from the `ram_disk` component's contextual menu. Choosing these items will cause Hawk to use the settings from the project, which are not set correctly, in building the driver. Instead, choose the **Build** or **Rebuild** command from the makefile contextual menu. This will invoke `os9make` and use the makefile to control the build instead of Hawk's project settings.

A Application Debugging Using SLIP/PPP

This appendix describes performing user-state debugging with Hawk over a Serial Line Internet Protocol (SLIP) connection. Windows NT is used as the host system and the LAN Communications SLIP device driver (spslip) is used to provide SLIP functionality in the SoftStax environment on an OS-9 target machine. This appendix uses the example application presented in Chapter 3, Hawk Application Debugging.

Supported Configurations application debugging using SLIP/PPP:

- Clients (host machines)
- Windows 95, 98, NT, ME, 2000
- Servers (target machines)
- Microware OS-9, Microware OS-9 for 68K
- Serial Interface
- SLIP - OS-9 target support via the LAN Communications or low-level network I/O
- PPP - OS-9 target support via the LAN Communications

Procedures may vary with your configuration. If you have questions, check Microware Software section of the RadiSys web site for the relevant procedures (listed below) or contact the Customer Support team at the following addresses:

`support@microware.com`

-or-

`www.radisys.com/service_support/microware/registered/appnotes/`

A password is required to access the above web address.

The following sections are included in this appendix:

- Debugging over a SLIP Connection
- Debugging over a SLIP Connection using Windows 2000

Debugging over a SLIP Connection

Windows 2000 Users:

The procedures in this section only apply to users whose host systems run Windows 95, 98, NT, or ME. If your host system runs Windows 2000, refer to the section Debugging over a SLIP Connection using Windows 2000.

Configuring the Host System

Complete the steps in the following sections to configure your host system.

Install Null Modem

The first step in configuring your host system is to install the Hawk Null Modem for the SLIP Connection. To do this, complete the following steps.

Step 1. Click `Start -> Settings -> Control Panel`.

Step 2. In the **Control Panel** window, double click on the `Modems` applet.

If the **Modems Properties** window displays, click the `Add` button.

Step 3. In the Install New Modem window:

- Check the `Don't detect` box and then click the `Next` button.
- In the **Manufacturers** area select `(Standard Modem Types)`.
- In the **Models** area select `Dial-Up Networking Serial Cable between 2 PCs`. Select the `Have Disk` button.
- In the **Install From Disk** window click the `Browse` button.
- In the **Locate File** window navigate to `$MWOS\DOS\BIN`, where `$MWOS` represents the directory on the Windows development host in which OS-9 is installed.

If you are using an NT host, open `mdmnull.nt40.inf`. Otherwise, open `mdmnull.inf`. Click `OK`.

Step 4. When you are returned to the **Install New Modem** window you should see the text `Hawk Null Modem SLIP Connection` in the **Models** area. Select `Next`.

Step 5. Select the port onto which you want to install the Hawk Null Modem. This example installs the Hawk Null Modem onto COM2. This allows the COM1 port to be used as the console. Select the `Next` button.

Install RAS Device

The second step is to add the Remote Access Service (RAS) to the list of network services. This service enables you to work offsite as though connected directly to a network. Adding this service can be accomplished by completing the following steps.

Step 1. Click `Start -> Settings -> Control Panel`.

Step 2. In the **Control Panel** window double click on the `Network` applet.

Step 3. In the **Network** window click the `Services` tab and then click the `Add` button.

Step 4. In the **Select Network Service** window select `Remote Access Service` from the list of network services. Click the `Have Disk` button.

Step 5. After you have inserted the appropriate Microsoft CD-ROM, click the `OK` button in the **Insert Disk** window.

Step 6. In the **Add RAS Device** window, select `Hawk Null Modem` and click `OK`. Be sure to install the RAS device onto the same port that you installed the HAWK modem in 1.

Dial-Up Networking

The third step is to create a Phonebook entry and to start the Dial-Up Networking. To do this, complete the following steps:

Step 1. Click `Start -> Programs -> Accessories -> Dial-Up Networking`.

Step 2. In the Dial-Up Networking window, click the `New` button to start the **New Phonebook Entry Wizard.**

1. Fill in the `Name the new phonebook entry` text field. Click `Next`.
2. For a SLIP connection, none of the three server options apply; therefore, do not check any of the option check boxes. Click the `Next` button.
3. Enter any `phone number` in the Phone Number text field (this is a required field). Click the `Next` button.

Step 3. In the **Dial-Up Networking** window, click the `More` button and select `Edit entry and modem properties` option.

Step 4. In the **Edit Phonebook Entry** window:

- Click the `Basic` tab. Ensure the **Dial using** field has `Hawk Null Modem` selected. Click the `Configure` button. Ensure the **Initial Speed** (bps) has the correct modem speed (`19200`).
- Click the `Server` tab.
- Make sure the **Dial-up server type** has `SLIP Internet` selected.
- Make sure the `TCP/IP` check box is checked.
- Click the `TCP/IP Settings` button.
- In the SLIP TCP/IP Settings window enter an appropriate `SLIP IP address`. For example, 192.168.1.1

- Click the `Script` tab. Ensure the After dialing (login) has chosen `None`.
- Click the `Security` tab. Ensure the `Accept any authentication` option is chosen.
- Click the `X25` tab. Ensure there is no X25 network chosen.

Step 5. In the **Dial-Up Networking** window, select the `Dial` button. In the **Connect to** window, click the `OK` button.

You only need to start the **Dial-Up Networking Monitor** once per login session. Windows NT takes approximately one to two minutes to complete the connection.

Refer to Chapter 3, Hawk Application Debugging for the generic application debugging procedures. Make sure the **Dial-Up Networking Monitor** is running and connected. It should appear in the Windows taskbar tool tray.

Debugging over a SLIP Connection using Windows 2000

Windows 2000 contains an option that is designed to allow you to change SLIP MTU and IP header compression parameters. However, this option does not work due to a bug in Windows 2000 (one that is not corrected in Service Packs 1 and 2). To properly connect over SLIP between Windows 2000 and OS-9, you must perform the procedures described in the following sections.

Stage One: Configuring the Target

Configure your target as described in the following steps:

Step 1. Open the `spf_desc.h` file in the following location:

```
/MWOS/OS9000/<processor>/PORTS/PROTOCOLS/SPF/SLIP/DEFS
```

Step 2. From this file, edit the line:

```
#define SLIPMTU     1006   /* IP level MTU */
```

to:

```
#define SLIPMTU     1500  /* IP level MTU */
#define COMPRESS_FLAG    0    /* compression off */
```

Step 3. Save and close the header file. Navigate up one directory to `SLIP` and execute `os9make` to rebuild the SLIP descriptors.

Step 4. Open the Configuration Wizard and enable SLIP support in your boot as follows:

1. From the **Configuration Wizard** main menu, select `Configure -> Bootfile -> Network Configuration`. Select the `Interface Configuration` tab.
2. Select the `SLIP Connection` box on the left side of the dialog to expand the **SLIP Connection** tree. Under this tree, select the `Use SLIP Connection` box.
3. In the **SLIP Configuration** area, enter the appropriate source and destination addresses.
4. Select the `Commit Change` button and click `OK` to exit the dialog.

Step 5. **Optional:** If you are building a coreboot image in addition to a bootfile image, you may want to perform the following task:

- Under the **Define ROM Ports** tab of the **Coreboot** -> **Main Configuration** dialog, select the applicable radio button (located in the **Define Communication Port** area). Select a baud rate of `19200` from the drop-down menu.

  The baud rate for the communication port can also be set with the following command: `xmode /t<n> baud=19200`, where `/t<n>` is the serial port descriptor.

Stage Two: Configuring the Host System

Once you have configured your target system, you complete the procedures below.

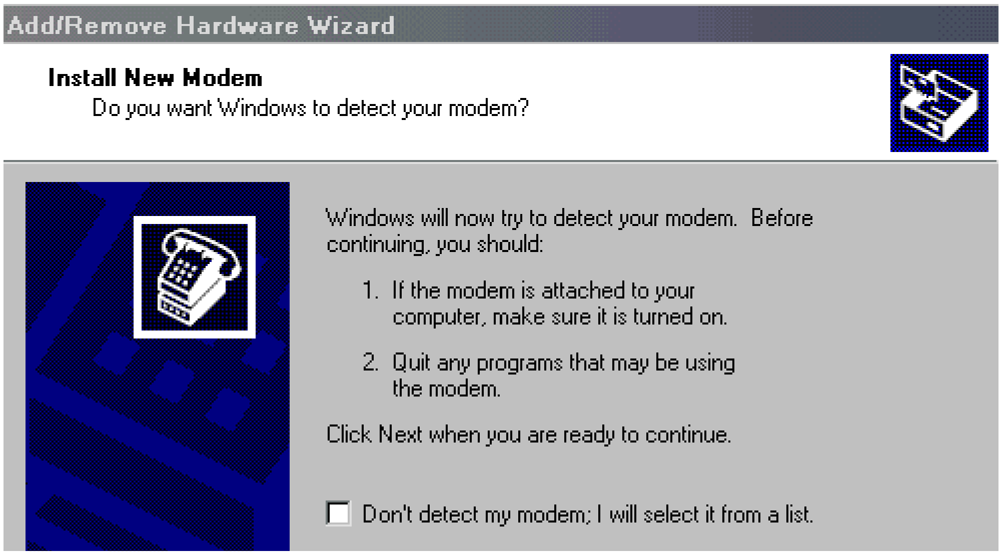
Install the Hawk Null Modem

The first step in configuring your host system is to install the Hawk Null Modem for the SLIP Connection:

Step 1. From the **Start** menu on your Windows desktop, select `Settings -> Control Panel`. In the Control Panel window, select `Phone and Modem Options`.

Step 2. From the **Phone And Modem Options** dialog, select the `Modems` tab. Click the `Add` button to display the **Add/Remove Hardware Wizard** dialog (shown in Figure 5-4).

Figure 5-4. Phone and Modem Options Dialog



Add/Remove Hardware Wizard

Install New Modem

Do you want Windows to detect your modem?

Windows will now try to detect your modem. Before continuing, you should:

1. If the modem is attached to your computer, make sure it is turned on.
2. Quit any programs that may be using the modem.

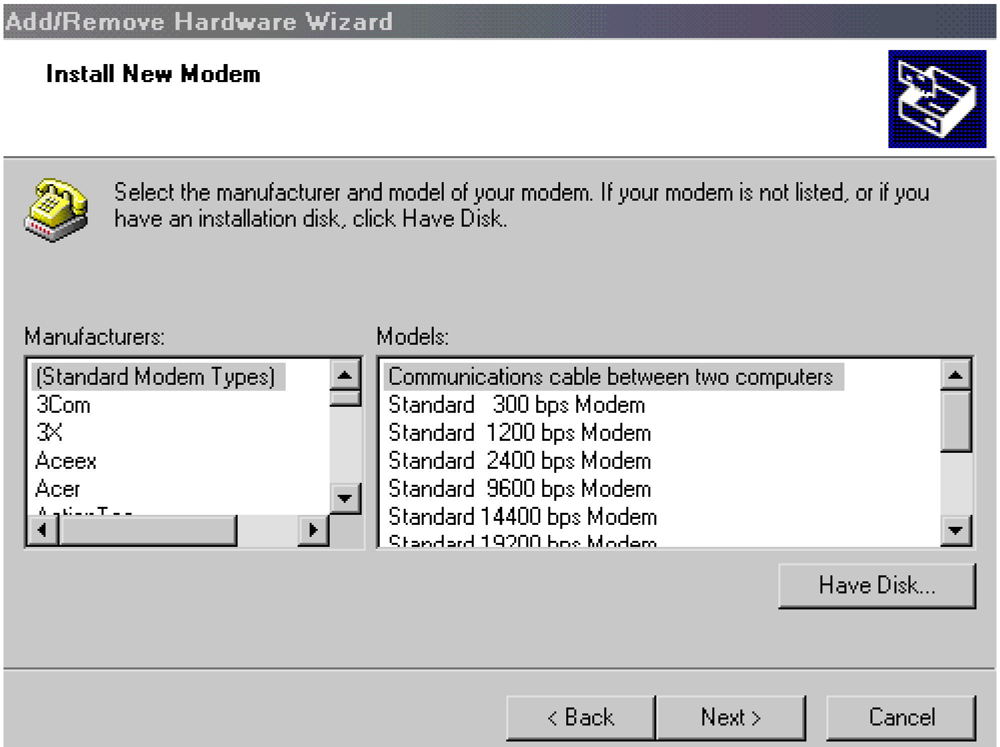
Click Next when you are ready to continue.

□ Don't detect my modem; I will select it from a list.

Step 3. In this dialog, check the `Don't detect my modem` box and click `Next`.

Step 4. A window displays (shown in Figure 5-5), allowing you to select from the following menus: **Manufacturers** and **Models**.

Figure 5-5. Manufacturers and Models Window



Select the manufacturer and model of your modem. If your modem is not listed, or if you have an installation disk, click Have Disk.

Manufacturers:

(Standard Modem Types)
3Com
3X
Aceex
Acer

Models:

Communications cable between two computers
Standard 300 bps Modem
Standard 1200 bps Modem
Standard 2400 bps Modem
Standard 9600 bps Modem
Standard 14400 bps Modem
Standard 19200 bps Modem

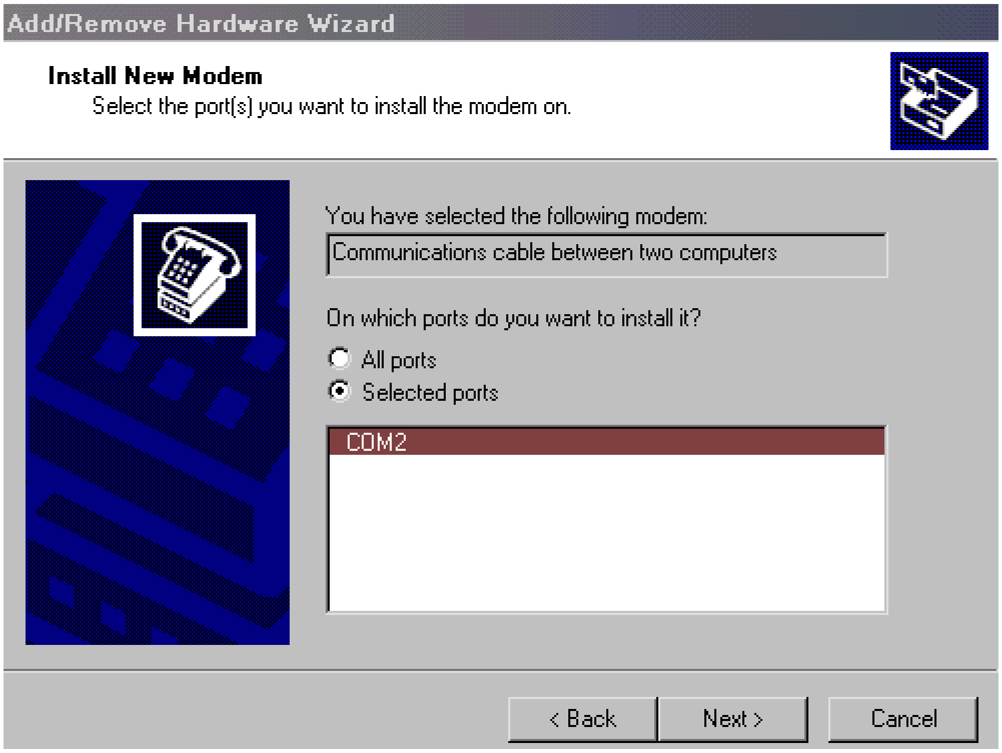
- From the **Manufacturers** list, select `(Standard Modem Types)`.
- From the **Models** list, select `Communications cable between two computers`.

Step 5. Select the `Have Disk` button. The **Install From Disk** window appears. From here, navigate to `<DIR>/MWOS/DOS/BIN` and select the file `mdmnull.nt40.inf`. Click `Open`.

Step 6. Select `OK` to exit the **Install From Disk** window. You should now see the text `Hawk Null Modem SLIP Connection` under the **Models** list. Click `Next` to proceed.

Step 7. A dialog displays that allows you to select the port onto which you want to install the modem. (This dialog is shown in Figure 5-6.) Make sure the **Selected ports** radio button is enabled and select an appropriate port from the list. Click `Next` to proceed.

Figure 5-6. Select Port Window



Add/Remove Hardware Wizard

Install New Modem

Select the port(s) you want to install the modem on.

You have selected the following modem:

Communications cable between two computers

On which ports do you want to install it?

○ All ports

● Selected ports

COM2

The example in this dialog installs the Hawk Null Modem onto COM2, which allows the COM1 port to be used as the console.

Step 8. The **Digital Signature Not Found** dialog appears, stating that the software you selected to install does not contain a Microsoft digital signature, and asking if you would like to proceed with the installation. Click `Yes`. Windows begins the modem installation. This may take a few minutes. When the installation is complete, select the `Finish` button. This will return you to the **Modems** tab of the **Phone and Modem Options** dialog.

Step 9. From the **Modems** tab, highlight the `Hawk Null Modem SLIP Connection`, then select the `Properties` button.

Step 10. The **Properties** dialog appears. In the **General** tab, change the **Max Port Speed** to `19200`. Click `OK`.

This is a Null Modem connection; querying the modem does not work.

Step 11. Click `OK` to close the **Phone and Modem Options** window.

Dial-Up Networking Setup

The second step for configuring your host system is to create a dial-up connection. To do this, complete the following tasks:

Step 1. From the **Start** menu on the Windows desktop, select `Settings -> Control Panel`. Open `Network and Dial-up Connections` window.

Step 2. From here, double-click to open the **Make a New Connection** item. The **Network Connection Wizard** appears. Click the `Next` button to proceed.

Step 3. Select the `Dial-up to private network` radio button. Click `Next`.

Step 4. Enter an appropriate phone number in the **Phone Number** text field. (This is a required field.) Click the `Next` button.

Step 5. Select the `For all users` or `Only for myself` radio button, as approrpiate, then click `Next`.

Step 6. In the next dialog, type the name of your connection in the applicable field and select `Finish`.

Step 7. The **Connect <Connection Name>** window should appear. If it does not, go back to **Network and Dial-up Connections window**, right-click on your connection and select `Properties` from the pull-down menu.

Step 8. The **Properties** dialog appears. On the **General** tab, select the `Configure` button and select `19200` from the **Maximum speed** drop-down menu. Click `OK`.

Step 9. Select the **Networking** tab. From the **Type of dial-up server** menu, select `SLIP: UNIX Connection`.

Step 10. From the same tab, select `Internet Protocol (TCP/IP)` from the **Components** list. Select the `Properties` button. The **Internet Protocol Properties** dialog appears. Select the `Use the following IP address` radio button, then enter an appropriate SLIP IP address (example: 10.0.0.2). Click `OK`.

Step 11. Click `OK` to close the **Properties** window, then click `Cancel` to close the **Connect <Connection Name>** window.

Dial-Up Connection

The third step for configuring your host system is to perform the dialo-up connection. To do this, complete the following tasks:

Step 1. Connect a serial cable between the communications port on your target machine (the one you configured for SLIP) and the port on the host system (the one that contains the Hawk Null Modem).

The target side fo the serial cable must have an R39F connector, and the host side of the cable must have a null modem connector.

Step 2. Apply power to the target machine.

Step 3. From the **Start** menu on the Windows desktop, navigate to `Settings -> Control Panel -> Network and Dial-up Connections`.

Step 4. Open the Dial-up connection you created in the Dial-Up Networking section.

Step 5. Select the `Dial` button.

Step 6. From the console, type `echo x >/t<n>` at the target prompt, where `/t<n>` is the descriptor of the serial port on which the target is establishing a SLIP connection. This allows Windows 2000 to complete the dialing process and connect to the target over SLIP.

Refer to Chapter 3, Hawk Application Debugging for the generic application debugging procedures.

B Subroutine Debugging Library

This appendix explains how Hawk can be used to debug user-state code located external to an application process, such as in a subroutine library.

The following information assumes that the appropriate underlying networking is in place. For subroutine library debugging, the underlying network includes SoftStax TCP/IP (SLIP, PPP or Ethernet).

If you are using a subroutine module that is built using a makefile, review Step 3 and apply these procedures to the source and build system for the module. Be sure to update the makefile for source level debugging and then rebuild the component as described.

Debugging a Subroutine Library

To debug a subroutine library, complete the following steps:

Step 1. Load the updated subroutine module into memory on the OS-9 target. If an older version already exists in memory be sure to take the appropriate action to ensure that the updated module is properly loaded. (Either remove the older version from memory first or make sure the new version has a higher revision.)

Step 2. Set up thse proper Source and Object Code search folders for the subroutine module in Hawk.

From here you have two ways in which to proceed:

- If you have access to the application code that uses the subroutine module, use Hawk to debug this application, proceed directly to Step 4.
- If you do not have access to the application code that uses the subroutine module, proceed to Step 3.

Step 3. If you do not have access to the application code that uses the subroutine module (in the case of `maui_inp`), run the application from the mshell prompt, attach to the application process using Hawk by specifying the process ID, and if necessary run any other process(es) necessary that will trigger/wakeup the application. (In the case of `maui_inp`, run `inp` with the necessary parameters.)

Step 4. Once you are in the Hawk Debugger context, attach to the subroutine module by specifying the module name. You should now be able to set breakpoints as desired in the subroutine module.

Information on installing and executing subroutine libraries can be found in the *OS-9 Technical Manual*.