Getting Started with Hawk

Version 2.3

www.radisys.com

World Headquarters
5445 NE Dawson Creek Drive • Hillsboro, OR
97124 USA
Phone: 503-615-1100 • Fax: 503-615-1121
Toll-Free: 800-950-0044

International Headquarters
Gebouw Flevopoort • Televisieweg 1A
NL-1322 AC • Almere, The Netherlands
Phone: 31 36 5365595 • Fax: 31 36 5365620

RadiSys Microware Communications Software Division, Inc.
1500 N.W. 118th Street
Des Moines, Iowa 50325
515-223-8000

Revision C
February 2002

Copyright and publication information

This manual reflects version 2.3 of Hawk
Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from RadiSys Microware Communications Software Division, Inc.

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.

February 2002

Table of Contents

Chapter 1: Welcome 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.

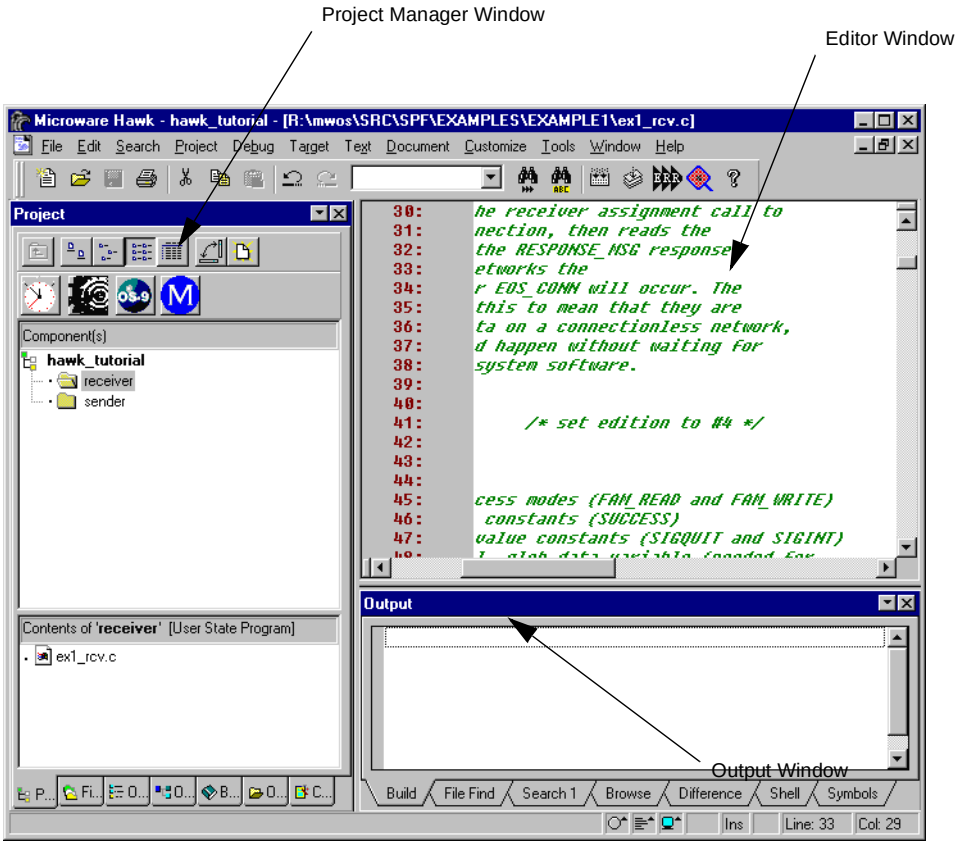
In addition, the Hawk development environment enables you to work in a seamless workspace integrating the following functions and tools:

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

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



Project Manager

The **Project Manager** window is located in the left portion of the Hawk IDE environment. The **Project Manage**r 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. Where other tools include various source files and require you to manually create a makefile, Hawk saves the 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 the following table.

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

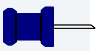
Components and units are the entities that the Project Manager uses to form logical associations between the various files it manages. A component is comparable to a module, descriptor, or driver, and a unit is an individual file.

For More Information

For more information on components and units, please refer to the ***Using Microware Hawk*** manual.

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.



Note

Only one project space can be open at 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.

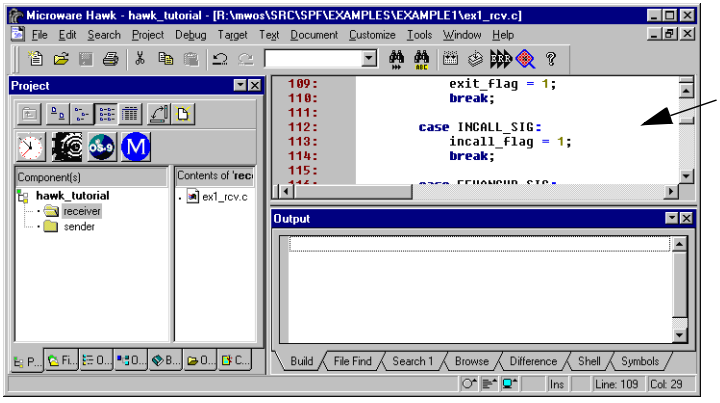
State files retain information about the windows and buffers opened during the last Hawk session and the position in which those windows and buffers existed. Workspaces are like mini-state files within projects. Each workspace retains a separate set of window information. Other state information such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

Editor

The **Editor** is located in the upper right portion of the Hawk IDE environment, as displayed in **Figure 1-2**. The **Editor** has the following features:

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

Figure 1-2 Hawk Source Code Editor



For More Information

To learn more about the source code editor, please refer to the ***Using Microware Hawk*** manual.

Debugger

Microware 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.

For More Information

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 Enhanced 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

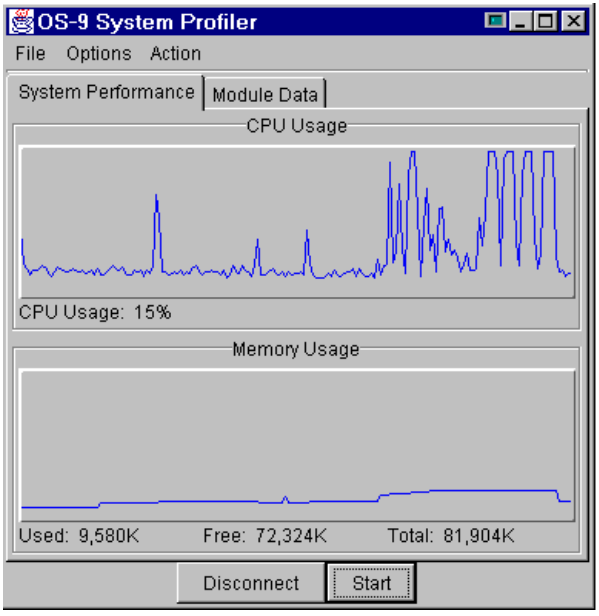
Note

A stand-alone version of the Debugger also exists, allowing for the debugging of 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 1-3**.

Figure 1-3 The Profiler Main Window



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.

Chapter 2: Creating Hawk Projects

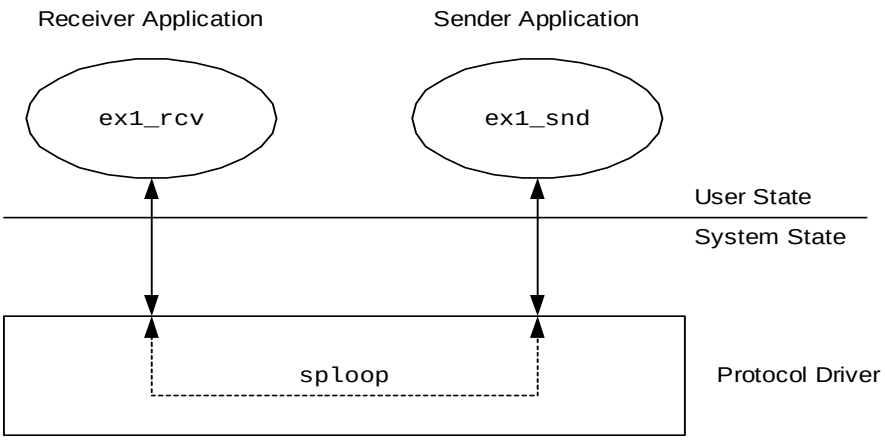
This chapter will teach you to create and modify a Hawk project and component. Before proceeding, be sure you have completed the following tasks:

- Install the Enhanced OS-9 software onto your host PC system.
- Connect your OS-9 target system to your host PC system.
- Create an OS-9 ROM Image and transferred it to the target OS-9 system.
- Boot your target OS-9 system to the shell prompt ($).

Example Application

This chapter uses the sploop example to illustrate the process of creating and modifying a Hawk project. The sploop example includes 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 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.

For More Information

Using SoftStax contains more information about sploop.

Create and Modify Projects and Components

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 the Sender Component**

Create a Project Space and Project

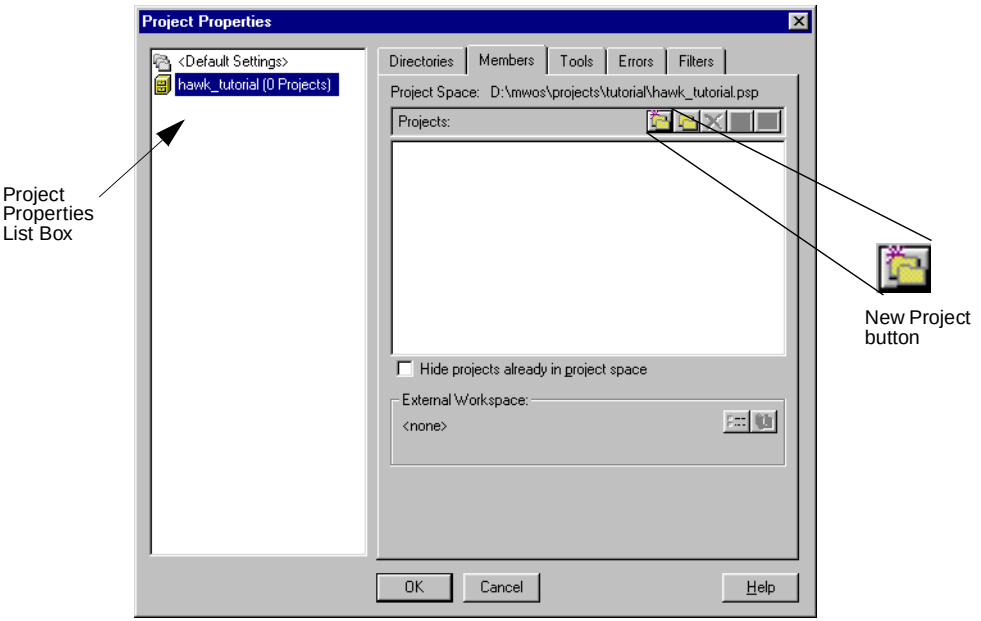
Before a project can be built, its project space must be created. Complete the following steps to create the project space and add a project to it.

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

Step 2. In the **Create a New Project Space** dialog, enter the file name for your project space. For this tutorial, enter the following path:
`<drive>:\mwos\projects\tutorial\hawk_tutorial`
The name of the project space is **hawk_tutorial**. `<drive>` is the hard drive or directory holding the `mwos` directory.

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

Figure 2-2 Project Properties dialog box



Step 4. Click the **New Project** button. 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. It is also permissible to use the same name as the project space because the project file will have a different extension. Project spaces use the `.psp` extension, and projects use the `.pjt` extension.

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 close the **Project Properties** dialog box.

Create a New Component for the Project

Once the project space and project 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 do build binary objects such as a library, a descriptor, or a module. The following table lists the valid component types:

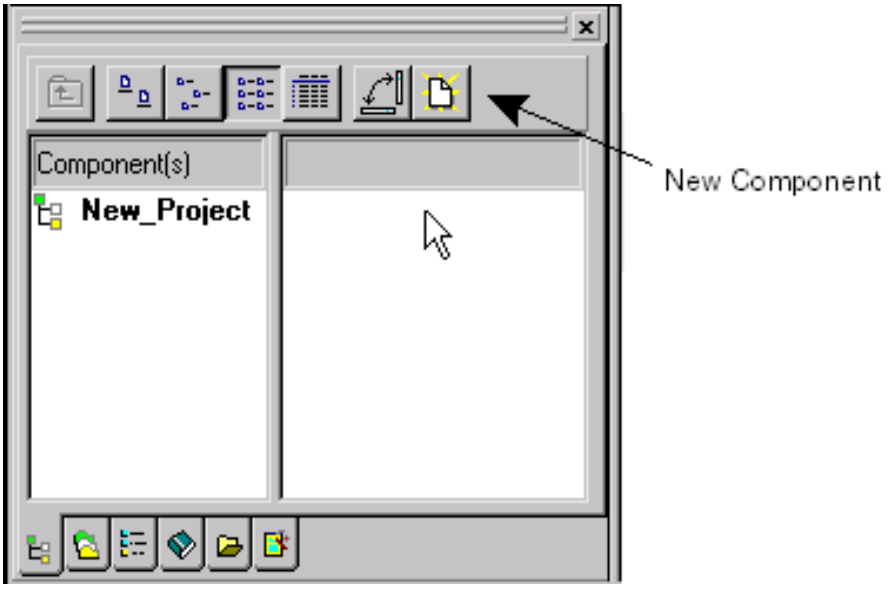
Table 2-1 Valid Component Types

Type	Output	Builder
user-state program or module	`Module`	Ultra C++
collection	`n/a`	n/a
descriptor	`Module`	Ultra C++ or Editmod
driver	`Module`	Ultra C++
file manager	`Module`	Ultra C++
I-Code Library	`*.i or *.il`	Ultra C++
O-Code Library	`*.l`	`libgen`
subroutine Module	`Module`	Ultra C++

To create a new component, complete the following steps:

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

Figure 2-3 The New Component Button



Step 2. The **Create New Component** dialog box appears. Projects consist of multiple components. For this example, start 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` or `Module`

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

Note

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

Note

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.

For More Information

Refer to the ***Using Ultra C/C++*** for additional information about default psect files.

Add a Unit to the Project Component

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. The Unit Maintenance dialog in Hawk allows you to add, delete, and reorder units within a component (accessed by right clicking on a project component). Hawk recognizes the following types of file extensions:

Table 2-2 File Extensions Recognized by Hawk

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`

Once you have added a component to your project, you can add a unit to that component. To do this, complete the following steps:

Step 1. From the **Create New Component** dialog, click `Next>>` to display the **Units** dialog box.

Step 2. At the **Look in** menu item, browse to the following location: `<mwos>\Src\Spf\Examples\Example1`

Step 3. Select the `ex1_rcv.c` file and click 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 4. Leave the **Generate Dependency Information** check box selected and click `Finish` at the bottom of the window. The **Generating Dependencies** progress box appears and the dependency information is created.

When complete, the **Components** frame should have a folder called `receiver`, and the **Contents** frame will contain a file called `ex1_rcv.c`.

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

Configure the Hawk Project

The following section discusses how to configure the Hawk project once you have added components and units to it.

Configure the Search Path

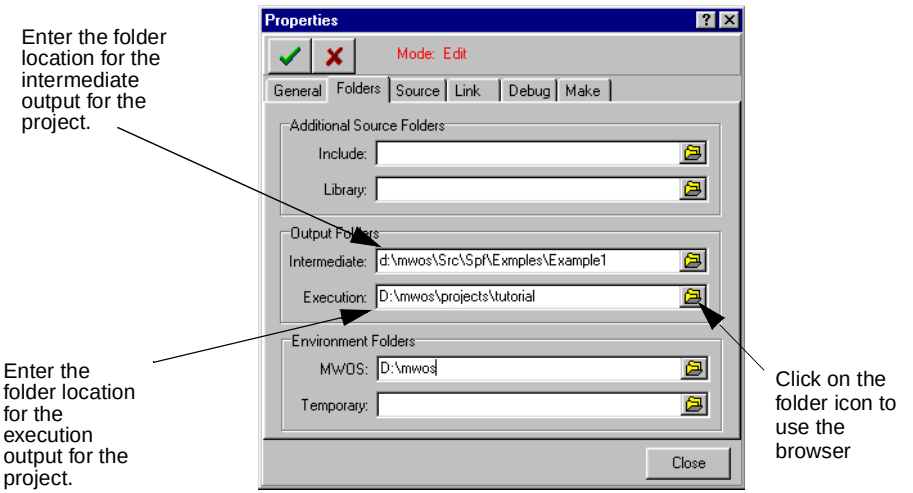
The next task is to configure your project to use correct search paths for libraries, header files, and store intermediate and executable modules.

Complete the following steps to set the location of the default header files and libraries:

Step 1. Select the `Project -> Properties` menu item. The **Properties** window for `Hawk_tutorial` is displayed.

The **General** tab contains the default processor. Use it to select the correct processor for your project.

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



Step 3. Under the **Output Folders** section, add the following intermediate path by typing the following command:
`<drive>:\mwos\SRC\SPF\EXAMPLES\EXAMPLE1`

Configure the Execution Search Path

Complete the following steps to configure the **Execution Search** path. This section shows how to add a path without directly typing it into the box.

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

Step 2. Select the tutorial folder. The resulting path in the **Add Folder** text box should be as follows:

```
<drive>:\mwos\projects\tutorial
```

Step 3. Click the green plus sign to add this path to the **Selected Folders List**.

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

Step 5. Click the green check mark button at the top of the **Properties** dialog box to save the new path.

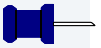
Configure Source Options

Complete the following steps 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.



Note

The **Link**, **Debug**, and **Make** tabs enable you to configure many options, but typically can be left with the default setup.

Step 4. Click the green check mark in the upper left hand corner of the **Properties** dialog to save changes to the project's properties.

Step 5. Click `Close` to close the **Properties** dialog box.

Build the Module

This section describes how to start the build and how to include a pre-compiled library.

Complete the following steps to build the `receiver` module.

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. The linker should state that there are many unresolved `ite_xxx` calls. These errors occur because the receiver process uses the Integrated Telephony Environment for Multimedia (ITEM) library, which has not been added to the component yet. The linker should also state that there is one unresolved `sleep()` call, which resides in the non-ANSI C library (`sys_clib.l`).

Hawk recognizes default OS-9 libraries and includes them automatically (such as `os_lib.l` and `clib.l`). Calls to `item.l`, `sys_clib.l`, and other libraries can be resolved by specifying the applicable library file(s) for searching.

Step 2. Click `OK` in the **Build Complete** dialog.

Step 3. Right click on the **Receiver** folder in the **Components** frame.

Step 4. Select `Properties`.

Step 5. Select the **Link** tab in the **Properties** window.

Step 6. Click on the `Browse O-Code Libs` icon (to the right of the O-Code Libraries field).

Step 7. Click on the `Add Existing Units` icon (to the right of the Add Library field).

Step 8. In the **Select File(s) to Add to Library List** box, browse to the following location:

```
<MWOS>\<OS>\<CPU>\LIB
```

Step 9. Click on `item.l` and click `Open` to include `item.l` as an O-Code library.

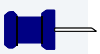
Step 10. Check the `non-ANSI C Library` box in the **Choose Libraries** list to include `sys_clib.l` as an O-Code library.

Step 11. Click `OK` to close the **Library Selection** box.

Step 12. Click the green check mark button in the upper left hand corner of the **Properties** dialog to save the O-Code library changes.

Step 13. Click `Close` to close the **Properties** dialog.

Step 14. Rebuild the project again using **Project** -> **Build**. The build should now complete without errors.



Note

OS-9 for 68K users will also see unresolved `_os_getstat()` and `_os_setstat()` calls, which can be resolved by including the `conv_lib.l` library. OS-9 users should not include this library.

Add the Sender Component

Add the sender component to the Hawk project by completing the following steps:

Step 1. Click the **New Component** button in the project dialog.

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

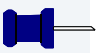
Name: `sender`

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

Chip: `<Processor>`

Type: `user-state Program` or `Module`

Step 3. `Click Next` to display the **Units** dialog and add `ex1_snd.c` and `item.l`.



Note

The `ex1_snd.c` file is in the same directory as `ex1_rcv.c`:
`<drive>:\mwos\SRC\SPF\EXAMPLES\EXAMPLE1\`

The `item.l` file is in the following directory:
`<drive>:\mwos\OS9000\<proc>\LIB\`

Enhanced OS-9 for 68K users should use the following path:
`<drive>:\mwos\OS9\68000\LIB`

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

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

Chapter 3: Application 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 with Hawk over an Ethernet connection, using SoftStax and the LAN Communications Pak to provide the Ethernet connectivity on an OS-9 target machine.

For More Information

For information on debugging system-state modules with Hawk, refer to Chapter 4: System-State Debugging. For information on performing user-state debugging with Hawk over a serial connection, refer to Appendix A: Application Debugging Using SLIP/PPP.

For More Information

This chapter will mention the Microware Configuration Wizard. This Wizard is not available for users of OS-9 for 68K. If you are an OS-9 for 68K user, refer to your board guide for information on adding modules to your bootfile.

RadiSys.

MICROWARE SOFTWARE

Prepare for Application Debugging

Before you can do application debugging with Hawk, the following must be true:

- SoftStax and Lan Comm (Ethernet) support are included in the bootfile and properly configured.
- The sploop protocol driver and descriptors are loaded onto the target.
- Time-outs must be increased for the receiver and sender processes
- Source level debugging must be enabled in Hawk.

The remainder of this section will discuss how to perform these tasks.

Note

If you do not have a ROM Image with SoftStax and Lan Comm enabled, rebuild your OS-9 ROM image to include them.

Load the Driver Modules

The sploop example requires that three modules be loaded onto your system: the `sploop` protocol driver module, the `loopc0` descriptor module, and the `loopc1` desciptor module. Because the modules are needed only for this example only, it is recommended that you load them individually using Hawk. The next section explains how to do this.

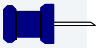
Load Modules Using Hawk

To load the modules using Hawk, navigate to the **Load** command under the **Target** menu and perform the following steps:

Step 1. Type the following command in the **Serial** window:

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

This starts user-state debugging on the target, puts the process in the background, and redirects the output to NIL. The target is now ready to communicate with the Hawk debugger on the host PC.



Note

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

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

Step 2. Select `Target -> Load` to open the **Load Module** dialog box.

Step 3. Type the target's DNS entry or IP address in the **Target:** box.

Step 4. Click on the **Browse** button and navigate to the directory holding the driver and descriptor modules. The directory is located in the following path: `mwos\<os>\<processor>\CMDS\BOOTOBJS\SPF`

Step 5. Select `loopc0` and click `Open`.

Step 6. Click `Load` in the **Load Module** dialog box. The module is loaded onto your reference board.

Step 7. Repeat steps two through five for `loopc1` and `sploop`.

Your reference board now has the drivers and descriptors needed to make this example work loaded onto it.

Increase the Time-outs

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

Step 1. Edit the `ex1_snd.c` file by clicking on the sender component and double-clicking the `ex1_snd.c` file. Follow the same procedure for `ex1_rcv.c`.

Change time-outs from 10 to 100 on the following lines in `ex1_snd.c`

```
155 connect_npb.ntfy_timeout = 10;
159 fehangup_npb.ntfy_timeout = 10;
163 datavail_npb.ntfy_timeout = 10;
```

Change time-outs to 100 on the following lines in `ex1_rcv.c`.

```
171 incall_npb.ntfy_timeout = 50;
175 fehangup_npb.ntfy_timeout = 10;
179datavail_npb.ntfy_timeout = 10;
```

Step 2. Save and close the files.

Step 3. Rebuild the project using `Project -> Build`.

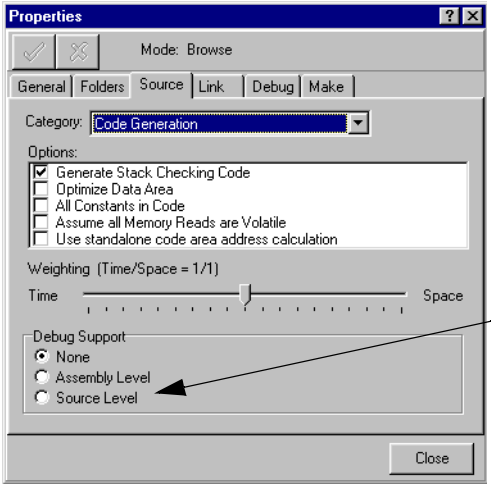
Step 4. Save the project by selecting `Project -> Save`.

Configure Debugging Support for the Project

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

Step 2. In the **Source** tab on the **Category:** pulldown menu, select `Code Generation`.

Step 3. In the **Debug Support** field, check `Source Level` to enable source level debugging.



Step 4. Click the green check mark button to save your changes.

Step 5. Click the **Close** button to dismiss the **Properties** dialog box.

Step 6. 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. During this process, you will complete the following tasks:

- **Set Up the Debugger**
- **Using the Debugger**

Set Up the Debugger

Complete the following steps to set up the Debugger:

Step 1. Apply power to your reference board.

Step 2. Click `Connect`—the left-most button in the Serial window. Use the default **Com Port Options**. The command prompt displays once the board has booted.

Step 3. Type the following command in the **Serial** window:

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

This starts user-state debugging on the target, puts the process in the background, and redirects the output to NIL. The target is now ready to communicate with the Hawk debugger on the host PC.

Step 4. Load your newly created sender program by selecting `Target -> Load.`

Step 5. Browse to the `mwos\Projects\tutorial` folder.

Step 6. Click on the sender module in the list box and click `Open` to select the sender program.

Step 7. Click `Load` to load the sender module on the target.

Step 8. Select `Debug -> Connect` on the top menu bar. The **Connect** dialog box appears.

Step 9. Click the Fork tab. The **Connect** window displays the target machine name (or IP address) in the **Target** text box. If the name is incorrect, enter the correct DNS name or IP Address.

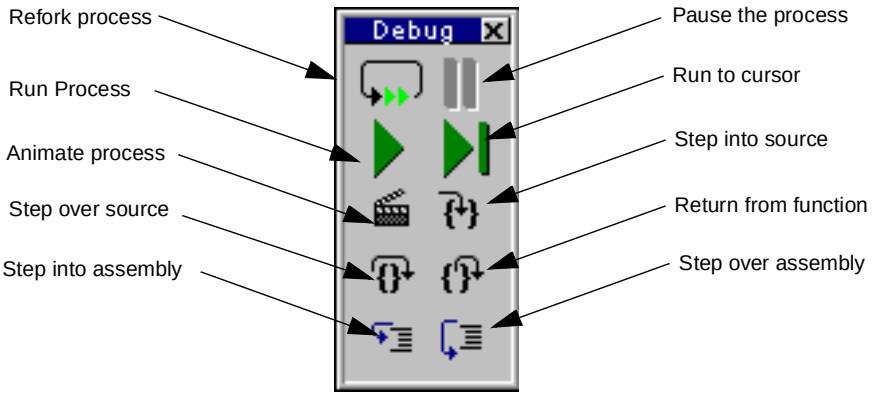
Step 10. In the **Program** box, browse to the `receiver` module in the `<MWOS>\PROJECTS\tutorial` folder and click Open.

Step 11. Click OK.

Using the Debugger

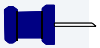
The next task is to step through the code. At this point the `ex1_rcv.c` source code is displayed in the **Source Code** window with a yellow-outlined arrow pointing to the `main()` line. In addition, the **Debug** toolbar is displayed.

Figure 3-1 The Debug toolbar



Complete the steps below to use the debugger.

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



Note

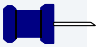
As you scroll through the code notice a large `while()` loop. The receiver sleeps until it gets a signal. It then wakes up and checks for incoming calls. If there is an incoming call, it reads the data, prints what it gets, then returns `Message Received` to the sender.

Step 2. Set a breakpoint at line 233 (`if (incall_flag)`) by clicking in the gray margin directly to the left of line 233. The breakpoint is indicated by a red dot in the gray margin.

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

Step 4. Type `Sender` in the serial window. A yellow outlined 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. You have now completed the basics of application debugging using Hawk.



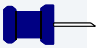
Note

If **Exit Debugger** is not available or the Hawk user interface appears to be locked up, telnet to the target and terminate the `receiver` process by typing `procs`. From this, determine the process ID of the `receiver`, and type `kill` <`number`>.

For More Information

For more information on the user-state debugging mechanism, refer to the ***Using Hawk*** manual.

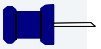
Chapter 4: System-State Debugging



Note

This chapter is only for customers who own an OS-9 for Embedded Systems package (OEM package). The OS-9 board level solution and OS-9 SDK packages do not contain the driver source code that is necessary for working the example in this chapter.

This chapter describes debugging system-state modules with Hawk over an Ethernet connection, using low-level network I/O (also known as LLROM) to provide the ethernet connectivity on an OS-9 target machine. As mentioned earlier, system-state debugging is typically needed only for device driver or other hardware dependent code. The example used in this section will revolve around debugging a RAM disk drive.



Note

In order to use Hawk for system-state debugging, an appropriate low-level network driver must be available. For OS-9, example low-level Ethernet drivers are typically provided for each supported target, along with low-level SLIP drivers for use with available serial ports. System-state debugging with Hawk is not supported on OS-9 for 68K, with the exception of on the MC68328ADS (low-level SLIP only).

If Hawk system-state debugging is not an option, consider using the ROMBug symbolic debugging facility. Information on ROMBug and symbolic debugging can be found in the ***Using ROMBug*** manual.

RadiSys.

MICROWARE SOFTWARE

Create the Driver Project

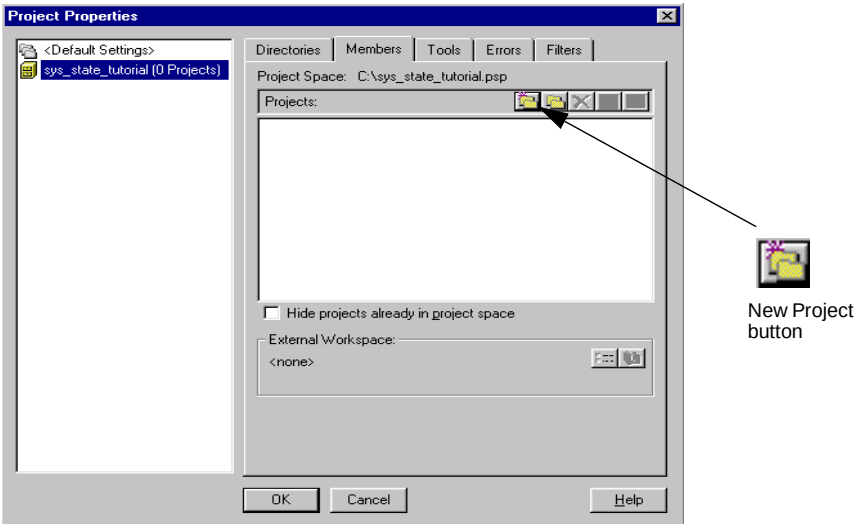
The first task is to create a new project space and project to hold the driver. The process is very similar to building an application module. Complete the following steps to create the driver project:

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\tutorial\sys_state_tutorial` The name of the project space is **sys_state_tutorial**. `<drive>` is the hard drive or directory holding the `mwos` directory.

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



Step 4. Click 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 **sys_state_tutorial** project space.

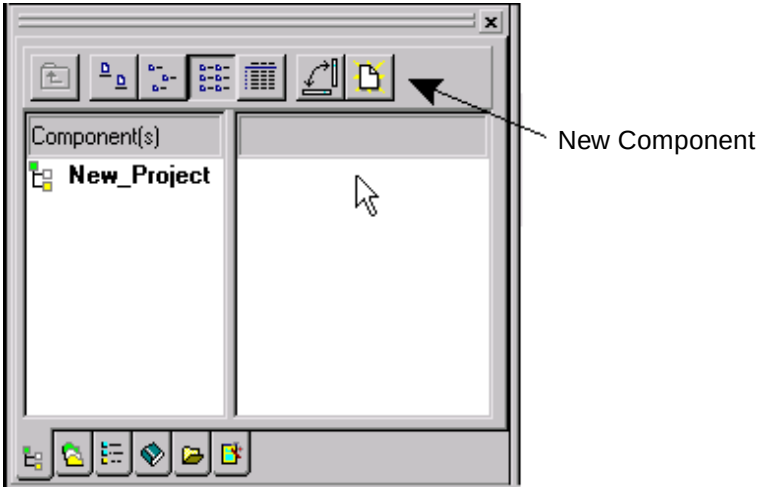
Step 7. Click OK to close the **Project Properties** window.

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. To do this, complete the following steps:

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

Figure 4-2 The 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`)

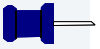
For More Information

Refer to ***Using Ultra C/C++*** for additional information about default psect files.

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`



Note

It is recommended that you make your own copy of the entire RAMDRVR directory and use that directory going forward. This will ensure that the original state of the RAMDRVR source, build system, and object code is preserved. Be sure to avoid using spaces anywhere in the source code directory path, and make sure the new directory is located at the same level as the original. For example:
`C:\MWOS\OS9000\SRC\IO\RBF\DRVR\RAMDRVR.DBG`

Step 5. 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 6. 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 to `all` and add the following file:

makefile

Step 8. De-select the **Generate Dependency Information** check box. For this example the existing makefile will be used to generate the driver module, as such Hawk generated dependencies will not be used in the build process.

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. To do this, complete the following steps:

Step 1. Right-click on the `makefile` file 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`

Step 3. Click on the green check mark button to save the changes.

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

Edit the Makefile File

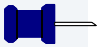
The next task is to edit the `makefile` file. To do this, complete the following steps:

Step 1. Double-click the makefile in the component window for `ram` or select the makefile and open it with `File -> Open` from the **File** menu.

Step 2. At the `DEBUG=` macro, 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.

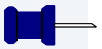
Step 3. At the `ODIR =` macro, replace `$(MWOS_ODIR)/BOOTOBJS` with your own object code directory. For example:
`ODIR = $(MWOS)/PROJECTS/TUTORIAL`



Note

Avoid using spaces anywhere in the object code directory path.

Step 4. Save the file and then right click on the makefile unit and select `Rebuild`. The debugging version of the ram driver will be placed wherever ODIR is defined.



Note

Do not choose the component build by right-clicking on the component to get the contextual menu. However, you should build by right-clicking on the makefile. This will invoke the makefile.

Prepare the Target for Debugging

Before this example can be performed, the OS-9 ROM Image must have certain services added and other services disabled. To do this, open the Configuration Wizard and complete the following steps:

Step 1. Enable the RAM disk.
(Configure->Bootfile->Disk Configuration, **Ram Disk** tab)

Step 2. Disable SoftStax.
(Configure->Bootfile->Network Configuration, **SoftStax Setup** tab)

Step 3. Enable Remote Ethernet Debugging.
(Configure->Coreboot->Main Configuration, **Debugger** tab)

Step 4. Set the Ethernet IP address.
(Configure->Coreboot->Main Configuration, **Ethernet** Tab)

Step 5. Check that the user-state Debugging Modules is selected in the **Master Builder** window (to load `undpd`).

Step 6. Load the bootfile onto your target. Use the directions found in the appropriate board guide for your target system.

Step 7. Reboot your target and check to see if `hlproto` has already been initialized. If not, type the following command into mshell.

```
p2init hlproto
```

Step 8. Type the following command at the `mshell` prompt to start the application debugger daemon.

```
undpd <>>>/nil&
```

Load the RAM Driver

To load the new RAM driver, select `Target -> Load`, and specify the target name (or IP address) and the ram driver module path.

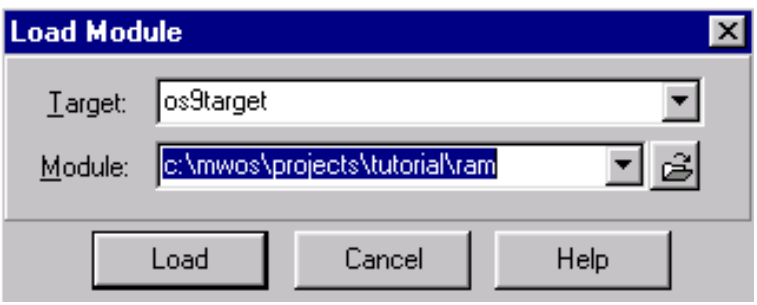
If a module is already in memory, attempting to load a new module of the same name, type, and revision number (or lower) will not replace the original. Instead, the original module will remain and its link count and will simply increment by one.

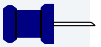
If the **Enable RAM disk** box is checked in the Wizard and/or if a ram driver module has been loaded into memory previously, then you will need to increment the revision number of the current debugging version of the ram driver module before loading it into memory. You can update the current debugging version of the ram driver module by updating `MWOS\OS9000\SRC\IO\RBF\DRVR\RAMDRVR.DBG\main.c` as follows:

```
_asm("_sysedit: equ 17");
_asm("_m_attrev: equ $A001");
/* Add the above line, making sure the module revision */
/* is one higher than that currently in memory. */
```

Be sure to save this file and then rebuild the ram driver as previously described before using Hawk to load the ram driver module into memory.

Figure 4-3 Load Module dialog





Note

If r0 (the device descriptor for the ram disk) is already initialized, be sure to deinitialize this device before proceeding. If r0 is configured as the Initial Device Name in the Wizard (refer to the text box, not the radio button), you will need to reconfigure your boot image since the original ram driver module was initialized at startup and cannot be deinitialized.

Prepare Hawk for Debugging

Complete the following 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. Click the `Source Code Browse` button. This displays the **Select Folders** window.

Step 4. From this window delete all current selected folders and add the path to your copy of the ram driver source directory. For example:

```
C:\MWOS\OS9000\SRC\IO\RBF\DRVR\RAMDRVR.DBG
```

Step 5. Click `OK`.

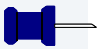
Step 6. Select the object code folders browse button and add the path to the debugging version of the ram driver object code. For example:
`C:\MWOS\PROJECTS\TUTORIAL`

You can do this by either manually typing in the path or by selecting the proper folder in the **Folders:** box.

Step 7. Close the **Options** window by clicking `OK`.

Attach Debugger to the System

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



Note

If sndp has not already been initialized, you will need to type the following command at the mshell prompt before proceeding:

```
p2init sndp
```

Step 1. Type the following command at the mshell prompt:

```
break
```

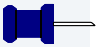
Step 2. Select `Debug -> Connect`.

Step 3. At the **Connect** window, select the **Attach** tab.

Step 4. Select `type: System`.

Step 5. Enter the OS-9 target's name (or IP address) in the **Target:** field

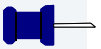
Step 6. Click `OK`. After a short time, the debugging environment appears.



Note

You must stop the target system (invoke the system-state debugger) by typing break from the mshell command line before trying to attach to the system. At any point after the attachment you can resume timesharing by selecting the green arrow on Hawk's debugging bar. You can also repeat the break/resume sequence as often as you require. If you detach, however, the target must be reset in order to establish reconnection.

Attach Debugger to the Module



Note

In order to attach to a system-state module the following must be true:

- The Hawk Debugger must already be attached to the OS-9 target system (see previous section).
- The OS-9 target system must be stopped (via break).

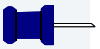
The next task is to attach to the module you will test. Complete the following steps:

Step 1. Select `Debug -> Connect -> Attach` tab and highlight `Module` in the **Type** box.

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

Step 3. Click `OK`.

Set Breakpoints in Module



Note

In order to set breakpoints in a system-state module, the following must be true:

- The Hawk Debugger must already be attached to the OS-9 target system and module.
- The OS-9 target system must be stopped (via break).

Step 1. 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 2. Expand the ram icon and select the file `init.c`.

Step 3. Expand the `init.c` icon, right click on the init function and select `Set Context`.

Set Context causes the Debugger's context to be set to the selected function or block. If there is an active stack frame for the selected function or block, the variables that are in scope can be evaluated. Breakpoints (in the current context) can then also be toggled from within the Editor Window.

Once you set context to the init function the `init.c` source file will be opened in the Editor Window (if it was not already), and a black arrow outlined in yellow will identify the new context location.

Step 4. Close the **Symbol Browser** window.

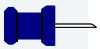
Step 5. In the Editor Window, set a breakpoint at the beginning of the init function. Double-click on the init function (highlighting any portion of this line will do), right click and select the **Toggle Breakpoint** option. A red dot will indicate the breakpoint position.

Step 6. Select the green arrow (Run) on the debugging bar to restart the system. Once the green arrow is selected, the OS-9 target will print out the following message:

"***Warning*** - breakpoints halt timesharing."

Step 7. Type `iniz /r0` at mshell prompt to have the `init()` function called in the ram driver module and therefore hit our breakpoint. The debugging bar will become active again in the Hawk Debugger, with the arrow indicating the current position.

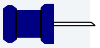
Step 8. Use the **Step into Source** button to step through the source code.



Note

Kernel-assisted debugging facilities are not available in system-state. Therefore, once the desired breakpoint(s) have been set, the OS-9 target system must be restarted (via the Hawk Debugger's Run option) in order for the applicable code to execute such that the breakpoint(s) will be hit.

In this case, use the iniz utility to call the ram driver's `init()` routine. As with the `break` utility, once a breakpoint is encountered in the instruction stream of the processor, the system-state debugger will step forth and once again take total control over the system. Tracing assembly instruction or stepping through source that can then be performed.



Note

The low and high level drivers for a particular device cannot be used or initialized at the same time. In situations where this conflict may exist (such as attempting to use the low-level ethernet driver to debug the Lan Comm ethernet driver), consider the following options that may allow the low-level network I/O and SoftStax/Lan Comm to coexist:

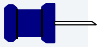
- Use low level SLIP for system-state debugging on one serial port, a different serial port for the system console and LAN Communications Pak Ethernet.
- Use low level Ethernet for system-state debugging and LAN Communications Pak SLIP (or PPP).

For More Information

Additional information on the system-state debugging mechanism is described in Chapter 8: Debugging OS-9 Projects of the ***Using Hawk*** manual.

Chapter 5: Using Makefiles with Hawk

This chapter describes building projects that run pre-existing makefiles.



Note

The example used in this chapter creates a system-state module. If you are using the OS-9 SDK and 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.

Using Makefiles in Hawk Projects

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 was on system-state debugging, in this chapter, the emphasis will be on the relationship between Hawk and the makefile.

Overview

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.

Building with Makefiles summary

The following steps reviews the basic tasks for running makefile built components within Hawk.

Step 1. Create a new project space and project.

Step 2. Add the source files, header files, and makefiles to the project in the **Units** dialog box.

Step 3. Deselect the `Generate Dependency` check box in the **Units** dialog box.

Step 4. 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 5. 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 6. 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.

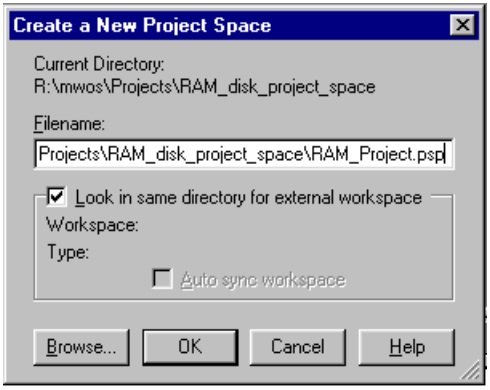
It is important to remember that the makefile will control the build process and information entered into Hawk may not always be used. In this case, the Hawk project is mainly a mechanism for organizing the component's files. The instances when the information entered in Hawk dialog boxes are not used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

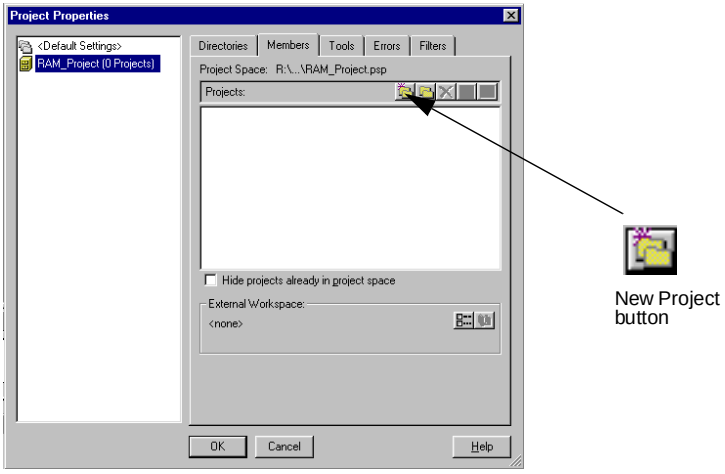
Figure 5-1 Create a New Project Space dialog box



Step 2. In the **Create a New Project Space** dialog box, enter the file name for your project space. For this example, enter the following path: `<drive>:\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 box.

Figure 5-2 Project Properties dialog box



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

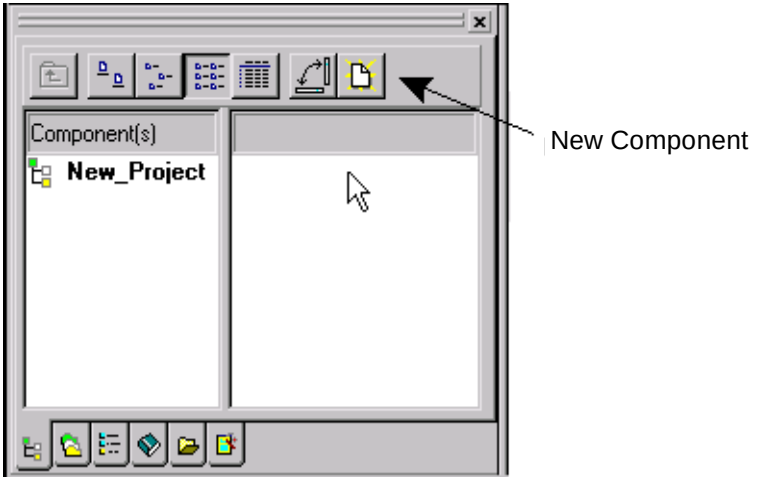
Step 7. Click OK to close the **Project Properties** window.

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

Figure 5-3



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`)

Note

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

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`

WARNING

Do not move the source files, header files, or makefiles from their directory. The makefiles have path information that would have to be updated if these files were moved to a different directory.

Step 5. 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 6. 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 to `all` and add the following file:

makefile

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

Step 8. De-select the **Generate Dependency Information** check box. Hawk should not generate dependencies at this time because the makefile contains dependency information.

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

Step 10. 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 the `makefile` file 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

More information about how `os9make` works and its command line options can be found in the ***Utilities Reference*** manual.

Step 3. Click on the green check mark button to save the changes.

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

Edit the Makefile File

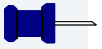
You may find it necessary to apply options in a makefile before building the component. In this example, you will build the debugging version of the RAM disk driver. The `makefile` file is edited to apply the debug option. Complete the following steps to set up the makefile to build the debugging version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



Note

Do not choose **Build** or **Rebuild** from the **Project** menu or the **Build** or **Rebuild** command 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.

Be sure to choose the **Build** or **Rebuild** command from the makefile's contextual menu. This will invoke `os9make` and use the makefile to control the build instead of Hawk's project settings.

Appendix 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 Pak 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: Application Debugging.

Note

Supported Configurations for Hawk Application Debugging using SLIP/PPP:

Clients (host machines)

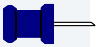
- Windows 95, 98, NT, ME, 2000

Servers (target machines)

- OS-9, OS-9 for 68K

Serial Interface

- SLIP - OS-9 target support via the LAN Communications Pak or low-level network I/O
- PPP - OS-9 target support via the LAN Communications Pak



Note

Procedures for using a different host/target/serial interface configuration will vary slightly. If you are using a different configuration and have questions, check the Application Notes section of the Microware Communications Software web site for the relevant procedures, or contact support@microware.com:

`http://www.microware.com/Support/Registered/AppNotes/appnotes.html`

Debugging with Hawk over a SLIP Connection

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`. Click 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 where Enhanced 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. Click `Next`.

Step 5. Select the port that you want to install the Hawk Null Modem onto. This example installs the Hawk Null Modem onto COM2. This allows the COM1 port to be used as the console. Click 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 step 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.

- Fill in the `Name the new phonebook entry` text field. Click `Next`.
- 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.
- Enter any `phone number` in the Phone Number text field (this is a required field). Click the `Next` button.

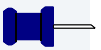
Step 3. In the Dial-Up Networking window click the `More` button and select `Edit entry and modem properties` option.

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.
- Ensure the Dial-up server type has `SLIP Internet` selected.
- Ensure 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

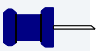
- Click the Script tab. Ensure the After dialing (login) has chosen None.
- Click the Security tab. Ensure the Accept any authentication option is chosen.
- Click the X25 tab. Ensure there is no X25 network chosen.

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



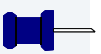
Note

You only need to start the Dial-Up Networking Monitor once per login session.



Note

Windows NT takes approximately one to two minutes to complete the connection.

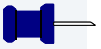


Note

Refer to Chapter 3: 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.

Appendix B: Subroutine Library Debugging

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.



Note

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).

For More Information

If you are using a subroutine module that is built using a makefile, review Chapter 5: Using Makefiles with Hawk 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 the proper Source and Object Code search folders for the subroutine module in Hawk.

If you have access to the application code that uses the subroutine module, use Hawk to debug this application, move directly to step four. If not, proceed to step three.

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.

For More Information

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