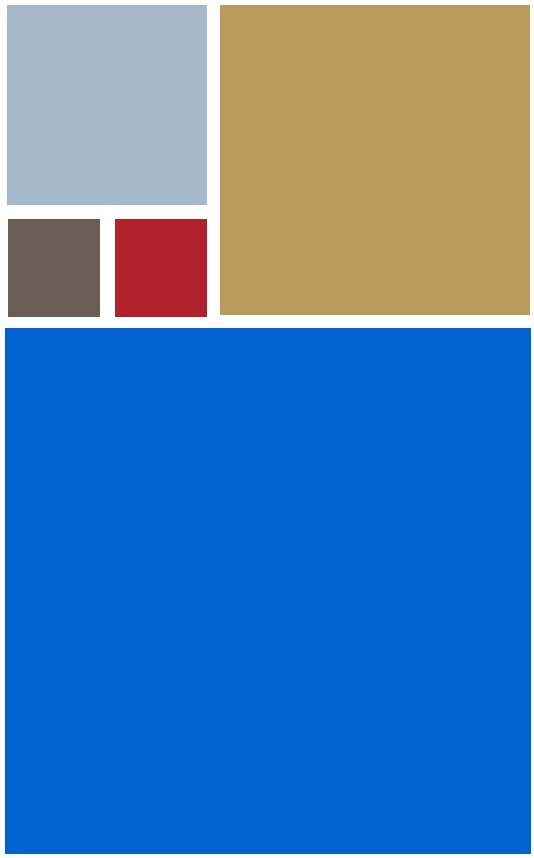


Home

Getting Started with PersonalJava™ Solution for OS-9® (MIPS32)

Version 3.1

RadiSys
THE POWER OF WE

www.radisys.com

Revision A • July 2006

Copyright and publication information

This manual reflects version 3.1 of PersonalJava™ Solution for OS-9.

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

July 2006

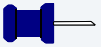
Table of Contents

Chapter 1: Introduction

This manual provides information for getting started with PersonalJava™ Solution for OS-9®. It is valid for the IDT 79S334A evaluation board.

For More Information

Refer to the current version of ***OS-9 Release Notes*** for possible last-minute updates to PersonalJava™ Solution for OS-9 or the IDT 79S334A board.



Note

Before proceeding, be certain you have installed either OS-9 for Embedded Systems or the OS-9 Board-Level Solution (BLS) for your processor, on your Windows-based host system. If you do not have either of these packages, contact your OS-9 supplier.

For More Information

Refer to the CD-ROM insert for information about installing PersonalJava™ Solution for OS-9 on your Windows-based host platform.

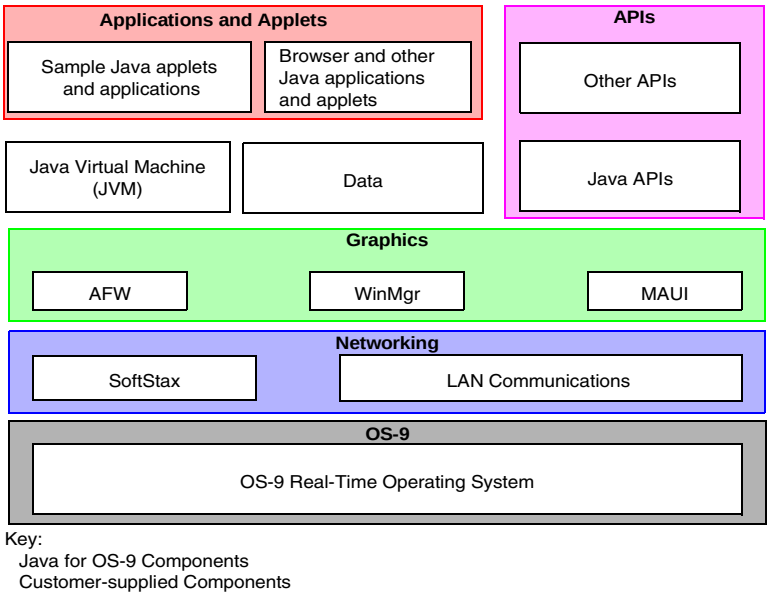
RadiSys.
MICROWARE SOFTWARE

PersonalJava™ Solution for OS-9 Runtime Components

PersonalJava™ Solution for OS-9 is a complete system software solution for developing Java-enabled devices. The PersonalJava™ Solution for OS-9 system consists of a scalable real-time operating system with specific software modules that help you create Java enabled devices without worrying about system software customization.

Figure 1-1 shows the PersonalJava™ Solution for OS-9 architecture. Each software subsystem found in PersonalJava™ Solution for OS-9 is defined in the following sections.

Figure 1-1 PersonalJava™ Solution for OS-9 Runtime Components



Note

Many of the following components were installed with your OS-9 for Embedded Systems or OS-9 Board Level Solution package.

OS-9

At the core of PersonalJava™ Solution for OS-9 is the OS-9 operating system and its support modules.

OS-9 Real-Time Operating System

OS-9 is an architecturally advanced, high performance real-time operating system available for several microprocessor families. At its core is the OS-9 stand-alone microkernel.

Coupled with the power of the microkernel, the unique modular architecture of OS-9 enables dynamic loading of any OS-9 system or user application module while the system is up and running.

Networking

The ability to communicate with other computers or devices is essential for a Java-enabled device. PersonalJava™ Solution for OS-9 uses the standard SoftStax® I/O implementation so a variety of transport layers can be used with Java.

SoftStax

SoftStax provides a consistent application-level interface using a variety of networking protocols. The protocols necessary for using PersonalJava™ Solution for OS-9 are included in LAN Communications.

LAN Communications

The Microware LAN Communications software consists of a TCP/IP protocol stack with User Datagram Protocol (UDP) support, Serial Line Internet Protocol/Compressed Serial Line Internet Protocol (SLIP/CSLIP) support, PPP support, and drivers for supported hardware.

Graphics

One of the strengths of Java as a programming language is its ability to support graphics. To handle graphics, PersonalJava™ Solution for OS-9 uses four components: Multimedia Application User Interface (MAUI®), Window Manager, the Application Framework (AFW), and the Java Abstract Windowing Toolkit (AWT).

Multimedia Application User Interface

MAUI is a high-level library that manages the display of graphics, text, and user input.

Window Manager

The PersonalJava™ Solution for OS-9 Window Manager is a MAUI application that manages windows. Three versions of the Window Manger are available, each with different levels of functionality.

Application Framework

The AFW is a class library that contains the code necessary to display Graphic User Interface (GUI) components and handle events for an interactive application.

Java Abstract Windowing Toolkit

The PersonalJava (PJAVA) environment includes an AWT package that allows Java applications to display GUI components, render images, draw graphics primitives, and respond to events. This package is standard across all PJAVA implementations, although some features are optional in PersonalJava™ Solution implementations. All optional AWT functionality is fully supported in Microware's PersonalJava™ Solution for OS-9.

Java Virtual Machine (JVM)

Consumer devices that interpret Java applications must contain the JVM. Java applications are comprised of Java classes consisting of byte codes.

Java byte codes are machine-independent and interpreted by the Java Virtual Machine. The purpose of the JVM is to interpret these Java byte codes and initiate appropriate actions on the host platform. In addition to executing byte codes in all classes within the system, the Java Virtual Machine also handles signals and Java exceptions, manages RAM, and is responsible for the simultaneous execution of multiple threads within the context of the JVM process.

Applications and Applets

Along with the basic system components, Microware has included several sample applications and applets on the PersonalJava™ Solution for OS-9 CD.

Sample Applications

Several sample applications have been included in this package. They are located in `MWOS\SRC\PJAVA\EXAMPLES`. Additional sample applets from Sun are located in `MWOS\DOS\jdk1.1.8\demo`.

Additional Java Tools

Running Java On a Diskless System

PersonalJava™ Solution for OS-9 includes a tool called JavaCodeCompact (JCC) that enables sets of Java class files to be pre-loaded in RAM or placed in ROM. This is accomplished by pre-processing the class files into an assembly language file that is eventually turned into a module. The module can then be loaded at run-time at a pre-determined address or loaded into the ROM of the device. This process eliminates the need to have the class files themselves, often called `classes.zip`, resident on the device.

For More Information

Refer to ***Using JavaCodeCompact for OS-9*** for instructions on using the JCC tool in the OS-9 environment.

For More Information

Refer to ***Using PersonalJava™ Solution for OS-9*** for information about creating Java applications for a diskless OS-9 target.

Java Development Tools

Since Java is neutral architecture, you can develop Java applications using any of the GUI-based Java development packages on the market. Some of these include Metrowerks CodeWarrior, Sunsoft's Java Workshop, and Symantec's Visual Cafe. As long as the output of the development environment is standard Java class files containing standard byte codes, the code is compatible with PersonalJava™ Solution for OS-9.

Standard Java class files contain a great deal of information about the source code from which they were compiled, including symbol names. With the appropriate tools, it is possible to de-compile Java code into an almost exact replica of the source code. Some of these tools address this problem by munging or obfuscating the object code so de-compilation is not as easy. Refer to Java-related web sites and UseNet news groups for information on these tools.

For More Information

For information about CodeWarrior, visit the Metrowerks web site at http://www.metrowerks.com/.

For information about Java Workshop, visit the Sun web site at http://www.sun.com/.

For information about Visual Cafe, visit the Symantec web site at http://www.Symantec.com/.

Windows® Java Development Kit (JDK)

To make it easier for you to perform native method work, Microware has included the Windows JDK v.1.1.8 in the package for the host system.

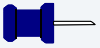
The `javah.exe` executable in this package has been modified to generate code that works with the Microware UltraC/C++ compiler.

The pre-loader classes are contained in the `jcc.zip` file. This file is on the Windows host machine in the `\MWOS\DOS\JDK1.1.8\lib` directory.

Chapter 2: Running PersonalJava™ Solution Demos

This chapter explains how to install and run Microware's PersonalJava™ Solution demo application and your Java applets and applications.

Microware's PersonalJava™ Solution can run in a diskless environment. For your convenience, this section explains how to run PersonalJava™ Solution using a NFS mounted file systems as a disk device. In addition, this chapter includes information on loading modules and running scripts from the NFS drive after the system has booted. In a truly embedded system, the loaded modules are placed in the boot and the script's commands are accomplished programmatically.



Note

Before proceeding, be certain you have installed either OS-9 for Embedded Systems or the OS-9 Board Level Solution (BLS) for your processor, on your Windows-based host system. If you do not have either of these packages, contact your OS-9 supplier.

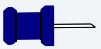
System Requirements

Your MIPS32 reference board must have the following hardware in order to run PersonalJava™ Solution for OS-9:

- 16MB of RAM
- 2MB of FLASH (Boot)
- VGA/NTSC display
- keyboard (optional)
- PS/2 or serial mouse

Installing PersonalJava™ Solution for OS-9

Before you begin, complete a test boot of your target and establish a virtual terminal connection between the PC and the target.



Note

For discussion on how to connect the target to your PC, refer to the OS-9 board guide for your reference board.

Assumptions

The information in the following sections makes the following assumptions of your site configuration:

- You are using a Linux machine (or equivalent) as a bootp server.
- Your MIPS32 board is configured to boot from the Linux bootp server. The boot filename should be `os9kboot`.
- Your Linux machine has a NFS exported directory suitable for use as a file system by OS-9000. Initially, this file system should be empty.
- Your Linux machine is configured to include Samba and is can be reached from your Windows NT machine.

Installing the PersonalJava™ Solution for OS-9 Files

Before you can use PersonalJava™ Solution for OS-9, you need to install it on your host system. To do this, complete the following steps:

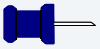
Note

Be certain to have your password on hand during this process; it is required for installation.

Step 1. Insert the Microware OS-9 for MIPS32 CD-ROM into your CD-ROM drive.

Step 2. Start the installer program and select the PersonalJava option.

Step 3. Follow the directions of the installer program.

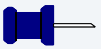


Note

Be certain you are installing PersonalJava™ Solution for OS-9 into your MWOS directory tree. If you do not install it in this directory, it may not work correctly.

Building the PersonalJava™ Solution Demo Bootfile

This section explains how to run the Microware Java demo programs on your MIPS32 reference board. If you would like to run other applets and applications instead of these demos, skip this section and go directly to the **Running Java Applets** section.



Note

The Java Development Kit (JDK), as shipped from Sun Microsystems, Inc., is strictly targeted toward desktop environments. PersonalJava™ Solution for OS-9 may be used on either disk-based or diskless systems. Moreover, the examples in this section use a simulated diskless system.

To run a Java demo program on your MIPS32 board, complete the following steps:

Step 1. From the Windows Start Menu, select `RadiSys -> Microware OS-9 for <product> -> Configuration Wizard`.

Step 2. Complete the following steps in the Configuration Wizard opening screen:

1. Verify that the MWOS location is correct. If it is not, select the `MWOS Location` field, browse to the MWOS directory tree on which you installed your Microware OS-9 package, and select it.

   Example: `C:\MWOS`

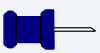
2. Select your target board's model in the **Port Selection** box.
3. Select the `Advanced Mode` option.

Fill in the **Configuration Name** field with the name of the Java demo configuration file, `JavaDemo`. This file is installed in the MWOS tree on your PC at the following location:
`C:\MWOS\OS9000\MIPS32\PORTS\<portname>\BOOTS\INSTALL\INI\JavaDemo.ini`

4. Click `OK`.

Note

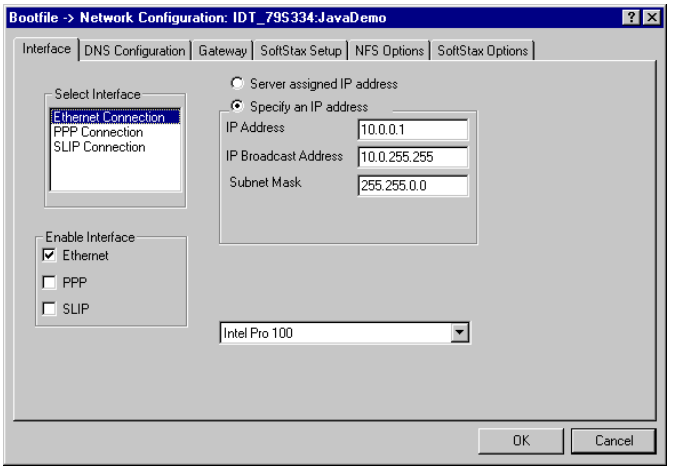
For the IDT 32334A reference board, `<portname>` is `IDT_79S334`.
For the IDT 32355 reference board, `<portname>` is `IDT_79EB355`.
For the MIPS Technologies Atlas 4KC reference platform, `<portname>` is `ATLAS_4KC`.



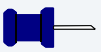
Note

For subsequent uses of a configuration, the Configuration Wizard automatically adds the board model to the beginning of the configuration name. Do not attempt to modify this portion of the name.

Step 3. From the Configuration Wizard menu options, select `Configure -> Bootfile -> Network Configuration`. Select the **Interface** tab at the top of the window. The following window appears:



Step 4. Select `Specify An IP Address` and fill in the Ethernet address text boxes with the Ethernet address information for your target.



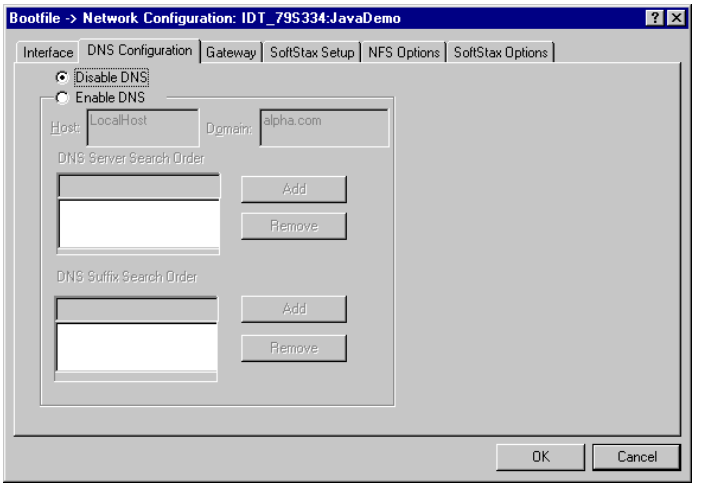
Note

If you do not know the Ethernet information for your target, contact your network administrator.

Step 5. Select the **Ethernet** check box.

Step 6. Make sure the name of the Ethernet interface chip appears in the drop down combo box. For the SH7709SE01, SH7709ASE01, and SH7750SE01, select `DP83902A`. For the EBX7709, select `3COM PC CARD Socket #0`.

Step 7. Select the **DNS Configuration** tab at the top of the window. The following window appears:



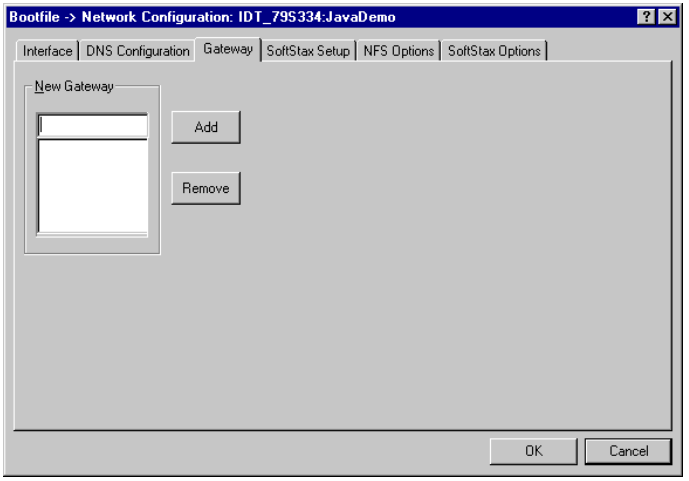
Complete the following fields:

- Select `Enable DNS`.
- Enter your target's host name in the **Host** field.
- Enter your target's domain name in the **Domain** field.
- Enter your target's DNS server search order addresses. Add as many addresses as are valid for your location.
- Enter your target's DNS suffix search order. Add as many suffixes as are valid for your location.

Note

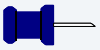
If you do not know the DNS information for your target, contact your network administrator.

Step 8. Select the `Gateway` tab at the top of the window. The following window appears:



From here, complete the following steps:

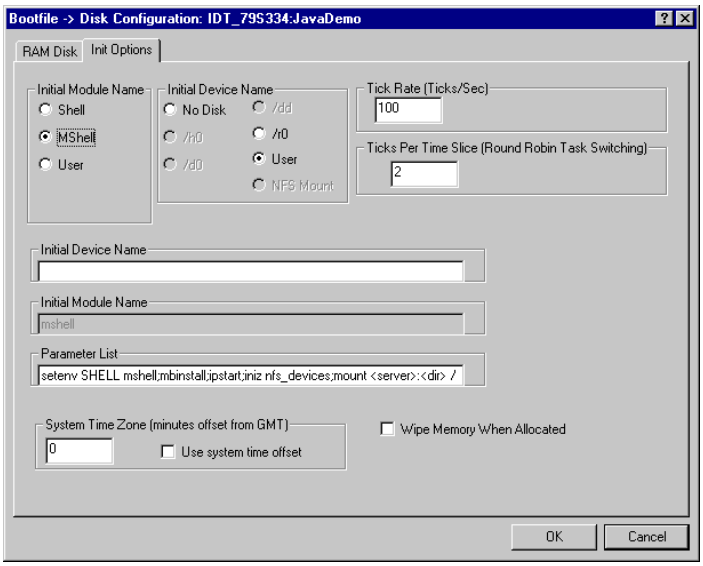
1. Enter your new gateway network address. Add as many addresses as are valid for your location.
2. Click `OK` to close the window.



Note

If you do not know the gateway information for your target, contact your network administrator.

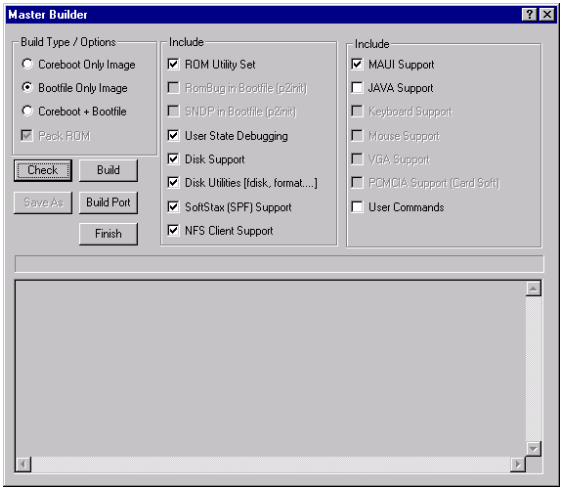
Step 9. From the Configuration Wizard menu options, select `Configure -> Bootfile -> Disk Configuration`. Select the **Init Options** tab at the top of the window. The following window appears:



Step 10. Edit the **Parameter List** string, replacing `<server>` with the name of your Linux machine and `<dir>` with the NFS exported directory you have dedicated to be your MIPS32 board's file system.

Step 11. Click the `OK` button to close the window.

Step 12. Build the JavaDemo bootfile image. From the Configuration Wizard **Master Builder** window, select `Configure -> Build Image`. The following window appears:



Step 13. Make sure the **Bootfile Only Image** build type is selected.

Step 14. In the **Master Builder** window, click `Build`. The bootfile is created in the following location: `C:\MWOS\OS9000\<portproc>\PORTS\<portname>\BOOTS\INSTALL\PORTBOOT` directory.

Step 15. Click `Save As`. A **Save As** dialog box appears.

Step 16. Navigate to the BOOTP/TFTP boot directory on you Linux machine.

Step 17. Type `os9kboot` into the filename text box.

Step 18. Click `Save`. The bootfile is saved to the root directory.

Step 19. Click `Finish`.

Step 20. Exit the Configuration Wizard.

Running the PersonalJava™ Solution Demo Bootfile

Complete the following steps on your PC to run the PersonalJava™ Solution demo bootfile:

Step 1. Create a PJAVA directory in the dedicated NFS directory.

Using the Windows Explorer, navigate to your Linux machine and create a directory in the dedicated directory called `PJAVA`.

Step 2. Navigate to the following directory:

`C:\MWOS\OS9000\MIPS32\PORTS\<portname>\CMDS\BOOTOBJS\PJAVA`

Copy `go.demo`, `pjruntime`, and each `.mod` file to the `PJAVA` directory in your dedicated directory.

Step 3. Start your virtual terminal program. (HyperTerminal is an example of a virtual terminal program.)

Step 4. Apply power to the board. Depending on how your system is configured, either the boot menu will appear or there will be an automatic boot from a BOOTP server. If yours is a system in which a boot menu should appear, proceed to step 5. If your system is one that boots automatically, go directly to the next section, **Running Java Applets**.

Step 5. If a boot menu appears: Type `eb`. The `os9kboot` built from the `JavaDemo` configuration file boots the reference board from the BOOTP server.

If you have a virtual console connected to the OS-9 target machine, you should see something similar to the following displayed:

```
OS-9000 kernel was found.
+3
$
```

At the mshell prompt type `chd PJAVA ; profile go.demo`. On the display device for your MIPS32 board the PersonalJava™ Solution LaunchPad application should display. You can select the demo you want to run from the LaunchPad application.

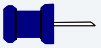
Running Java Applets

This section describes what you need to do to prepare your MIPS32 reference board to run the Sun Demo applets or your applets. The JDK v1.1 demo applets are contained in `MWOS/DOS/jdk1.1.8/demo` on your host computer.

Profile the pjava_setup Script

To profile the `pjava_setup` script, complete the following steps:

Step 1. Boot the board in the same manner you did for running the demos. However, this time, instead of profiling `go.demo`, profile `pjava_setup`. This will set the proper environment variables and start the Window Manager in preparation for running graphics-based Java applets.



Note

The `pjava_setup` script must be re-profiled each time the board is booted.

Step 2. Apply power to your MIPS32 board. The target should boot to a console prompt.

Step 3. Move to the `PJAVA` directory by entering the following command:

```
chd /h0/PJAVA
```

Step 4. Enter the following command to run the `pjava_setup` script:

```
profile pjava_setup
```

Step 5. Type the following command to display the environment variables:

```
printenv
```

Step 6. Compare the listing on your screen with the following listing. Make sure that the listed environment variables are set correctly.

```
JAVA_HOME=/mm
CLASSPATH=.
```

Step 7. Make sure the `maui_inp` and Window Manager processes are running by typing the following command:

```
procs
```

You will see a listing of the processes that are currently running on your reference board. It will look similar to the following illustration.

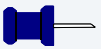
```
Id PId Grp.Usr  Prior  MemSiz Sig S    CPU Time   Age Module & I/O
  2   5   0.0    256   24.00k   0 s        0.03  0:02 maui_inp <>>>nil
  3   0   0.0    128   96.00k   0 s        0.08   ??? inetd <>>>nil
  4   0   0.0    128   20.00k   0 e        0.00   ??? spf_rx
  5   0   0.0    128   52.00k   0 w        0.48  0:04 mshell <>>>term
  6   5   0.0    250  168.00k   0 s        1.55  0:02 winmgr <>>>nil
  7   6   0.0    256  180.00k   0 s        0.58  0:02 maui_win <>>>nil
  8   5   0.0    128   48.00k   0 *        0.04  0:00 procs <>>>term
```

Step 8. Check that `maui_inp` and `winmgr` are listed under the Module heading.

You are now ready to run your applet.

Running an Applet

Once the board is set up by the `pjava_setup` script, you can run any desired example applet. To do this, complete the following steps:



Note

If your applet requires resources that are not present on the MIPS32 board (such as sound), it may not work correctly.

Step 1. Change to the directory containing the applet.
Example: `chd /h0/MWOS/DOS/jdk1.1.8/demo/TicTacToe/1.1`

Step 2. Run the applet by typing one of the following commands:

```
pappletviewer <htmlfile>
```

or

```
pjava sun.applet.AppletViewer <htmlfile>
```

Considerations for Running Your Own PersonalJava™ Solution Applications

The following section is a complete list of details that need to be addressed when you decide to run your applications. Many of these are solved with the `go.demo` script.

For More Information

See **Appendix A: Java Load Script** for an example `loadjava` script.

If your ultimate target is a diskless system, then the steps taken in `loadjava` must be accomplished by setting environment variables in the init module, and including the loaded modules in the bootfile or ROM image.

1. The environment variables need to be set correctly for the target system. Omit environment variables that are not applicable. (For those with a diskless environment, variables set to disk paths need not be set.) The following environment variables are included:
   - MWOS – location of your MWOS directory
   - JAVA_HOME – location of your Java properties files
   - CLASSPATH – list of directories and zip files to search for class files
   - LD_LIBRARY_PATH – list of directories to search for native method libraries
   - PATH – list of directories to search for executable files
   - PORT – device used to communicate with the user
   - USER – name used to refer to the user
   - HOME – “home” directory for the user
   - TZ – time zone setting for the system

2. The modules (executable code and configuration data) for PersonalJava™ Solution need to be in memory or at their appropriate location on the disk, if applicable. These include the following modules:

   - `winmgr` – PersonalJava™ Solution window manager (alternatively, `winmgrs` or `winmgrg` could be used)
   - `winmgr.dat` – window manager settings
   - `stock_8.res` – 8-bit image resources for the window manager and application framework
   - `stock_9.res` – 16-bit image resources for the window manager and application framework (available only on the SH7709SE01, SH7709ASE01, and SH7750SE01 Solution Engines)
   - `pjava` – PersonalJava™ Solution starter application
   - `libjavai.so` - PersonalJava™ Solution virtual machine
   - `libmawt.so` – AWT shared library module
   - `libmawt_0.dat` – pre-computed color palette
   - `libjavafile.so` - file handling native methods
   - `libnet.so` - TCP/IP network handling native methods
   - `libzip.so` - ZIP file handling native methods
   - font modules – various modules related to font rendering
   - MAUI modules – various modules related to graphics support
   - `*.properties` – property files needed by PersonalJava™ Solution (such as from a modman archive)

3. Before running a graphical PJAVA application, you must get the Window Manager started. Moreover, the Window Manager will not work until the MAUI input process has begun. The following command lines show an example startup sequence:

```
maui_inp ^255 <>>>/nil &
winmgr ^250 <>>>/nil &
pjava <application class>
```

Examine the system running the demos and the `loadjava` script for more information on configuring a system to run your own PersonalJava™ Solution applications.

Appendix A: Java Load Script

This appendix explains the Java Load Script required to start Java on a target platform.

Example Java Load Script

This example load script was used to set up the OS-9 init module to run Java. Use this script as a basis for your own scripts when configuring your system to run Java applications.

MIPS32 loadjava Script

The following example loadjava script is used on a SuperH target. This script will vary slightly depending on your target platform. Refer to your target's version for a more applicable example.

```
-tnpna
* * * * * * * * * * * * * * * * * * * *
* Load script for PersonalJava v3.x
*

*
* Set environment variables
*

let drive = "/dd"

setenv MWOS /%drive/MWOS
setenv JAVA_HOME $MWOS/SRC/PJAVA
setenv CLASSPATH $JAVA_HOME/LIB/classes.zip:\
$JAVA_HOME/LIB/javamath.zip:\
$JAVA_HOME/LIB/javasql.zip:\
$JAVA_HOME/LIB/javarmi.zip:\
$JAVA_HOME/LIB/sunrmi.zip:.
if (len(env("PATH")) == 0)
    setenv PATH /%drive/CMDS
endif
setenv PATH $MWOS/OS9000/MIPS32/CMDS:$PATH
setenv LD_LIBRARY_PATH $MWOS/OS9000/MIPS32/LIB/SHARED
setenv PORT /term
setenv HOME /%drive
setenv USER java_user
setenv TZ CST

let graphical_apps = TRUE
if (%graphical_apps == TRUE)
*
* All lines below this point are only needed if graphical Personal
* Java applications are being used. Assign FALSE to graphical_apps
* if you don't need this support.
*

*
* Load the Java Window Manager application, settings file,  and a
* file containing the resource module 'stock_x.res'
*
*    winmgrs  - Simplest/smallest Window Manager - No window frames
*    winmgr   - Standard Window Manager (default)
*    winmgrg  - Debug version - "Send Shutdown Message" &
```

```
*                "Dump Window Tree" fuctionality
*
*   winmgr.dat- Window Manager settings file
*
*    stock_8.res- 8-bit bitmap and cursor support (default)
*    stock_9.res- 16-bit bitmap and cursor support
*
let winmgr = "winmgr";* winmgr, winmgrg, or winmgrs
let resfile ="stock_8.res";* stock_8.res or stock_9.res

load -d $MWOS/OS9000/MIPS32/CMDS/%winmgr
load -d $MWOS/OS9000/MIPS32/CMDS/winmgr.dat
load -d $MWOS/OS9000/MIPS32/PORTS/IDT_79S334/CMDS/%resfile

*
* Load the pre-generated libmawt Color Cube module
*
load -d $MWOS/OS9000/MIPS32/PORTS/IDT_79S334/LIB/SHARED/libmawt_0.dat

*
* Load fonts for "font.properties"
*
load -d $MWOS/OS9000/MIPS32/ASSETS/FONTS/AGFA/MT/*
load -d $MWOS/OS9000/MIPS32/ASSETS/FONTS/AGFA/TT/utt.ss

*
* Initialize the keyboard and mouse devices, if needed
*
iniz m0

*
* Launch the MAUI input process
*
maui_inp ^256 <>>>/nil &

*
* Start window manager loaded above
*
%winmgr ^250 <>>>/nil &
-nt
unlet winmgr resfile
endif

-nt
unlet graphical_apps drive
```