Home

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

Version 3.1

RadiSys
THE POWER OF WE

www.radisys.com
Revision C • July 2006

Copyright and publication information

Disclaimer

Reproduction notice

Table of Contents

Chapter 1: Introduction

This manual provides you with the information you need to get started with PersonalJava™ Solution for OS-9®. The information in this manual is valid for the SH7709SE01 Solution Engine, SH7709ASE01 Solution Engine, EBX7709 Reference Platform, and SH7750SE01 Solution Engine. Information that is specific to a certain platform is appropriately labeled.

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 SuperH 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

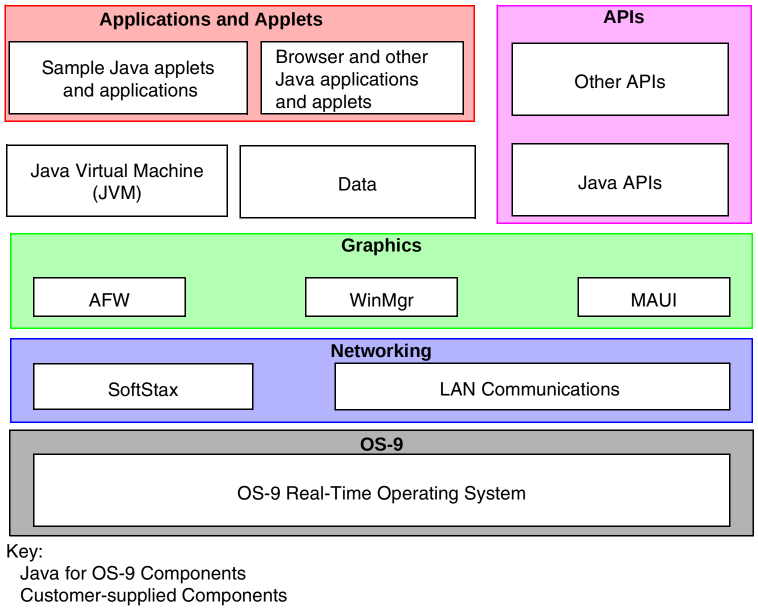
Refer to the CD-ROM insert for information about installing PersonalJava™ Solution for OS-9 on your Windows-based host platform.

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


Applications and Applets
Sample Java applets and applications
Browser and other Java applications and applets
APIs
Other APIs
Java APIs
Java Virtual Machine (JVM)
Data
Graphics
AFW
WinMgr
MAUI
Networking
SoftStax
LAN Communications
OS-9
OS-9 Real-Time Operating System
Key:
Java for OS-9 Components
Customer-supplied Components

Note

Many of these components were installed with your **OS-9 for Embedded Systems** or **OS-9 Board Level Solution** package. You must have installed one of these packages prior to installing PersonalJava™ Solution for OS-9.

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 support for 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 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 this tool in the OS-9 environment and 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 architecture neutral, users can develop their 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 to name a few. 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 more information about CodeWarrior, visit the Metrowerks website at http://www.metrowerks.com/.

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

For more information about Visual Cafe, visit the Symantec website 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 Demos

This chapter explains how to install and run Microware's PersonalJava demo application as well as your own Java applets and applications.

Microware's PersonalJava can run in a completely diskless environment. For your convenience, this section explains how to run PersonalJava using a PCMCIA IDE card as an IDE disk. In addition, this chapter includes information on loading modules and running scripts from the PCMCIA IDE card after the system has booted. In a truly embedded system, the modules that were loaded would have been placed in the boot and the script's commands would be accomplished programmatically.

Note

Before you begin, you must either have installed **Microware OS-9 for Embedded Systems** or the **Microware OS-9 Board Level Solution** for the target on your Windows-based host. If you do not have these packages, contact your OS-9 supplier.

Note

The procedures in this chapter use the E: drive on your host (this may vary depending on where you chose to install your PersonalJava™ Solution for Microware OS-9 package).

System Requirements

Your SuperH reference board must have the following to run Java for OS-9:

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

Installing PersonalJava™ Solution for OS-9

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

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

Installation of Java on the target is initiated on the PC and finished on the target. Therefore, the following instructions are divided into these headings:

- **Installing the PersonalJava™ Solution for OS-9 Files**
- **Building the PersonalJava Demo Bootfile**
- **Running the PersonalJava Demo Bootfile**
- **Running Java Applets**
- **Considerations for Running Your Own PersonalJava Applications**

Installing the PersonalJava™ Solution for OS-9 Files

Before you can use PersonalJava (PJAVA) Solution for OS-9, you need to install it on your host system and on your SuperH reference board.

Note
You will be prompted for your password during the installation process, so have it ready before you start.

Installing Files onto the Host

Step 1. Insert the **OS-9 for SuperH CD-ROM** into your CD-ROM drive.

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

Step 3. Follow the directions in the installer windows.

Note

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

Installing Files onto the Target

The files that will go onto the target are found in the MWOS directory on the host machine. The path to the files is as follows: `MWOS\OS9000\<proc>\PORTS\<board>\PJAVA`.

Note

<proc> is either SH3 or SH4, depending on your target. <board> is either SH7709, SH7709ASE, EBX7709, or SH7750SE.

The `PJAVA` folder contains two relevant items, including pjava.mat and readme.txt. pjava.mat is an archive (in the Microware Archive Tool (MAT) format) of the files to go on the target. readme.txt explains how to install the MAT archive onto the PCMCIA IDE PC Card either from OS-9 or your Windows host machine.

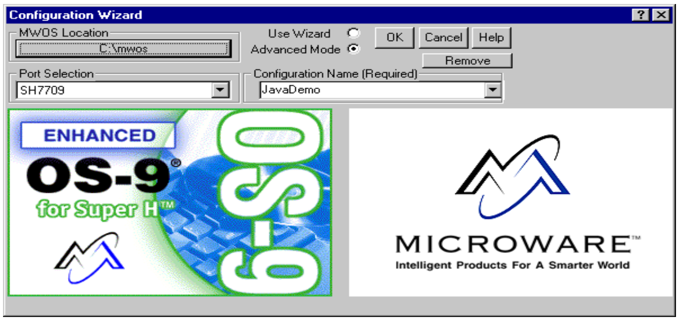
Building the PersonalJava Demo Bootfile

This section tells you how to run the Microware Java demo programs on your SuperH reference board. If you want to run other applets and applications instead of these demos, then you should skip to the following section: **Running Java Applets**.

Note

The Java Development Kit (JDK) as shipped from Sun Microsystems, Inc. is targeted strictly at desktop environments. PersonalJava™ Solution for OS-9 may be used on either disk-based or diskless systems. The examples you will be running use a diskless system.

Step 1. From the Windows Start Menu, select `Programs -> Microware OS-9 for <product> -> Configuration Wizard`. The following window appears:



Complete the following on the opening screen:

- Verify that the MWOS location is correct. If it not, click the `MWOS Location` field, browse to the MWOS directory tree that was installed when you installed your Microware OS-9 for Embedded Systems or Microware OS-9 Board Level Solution package, and select it.

  Example: `C:\MWOS`

- Select your target's board model in the `Port Selection` box.
- 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 into the MWOS tree on your PC at
  `C:\MWOS\OS9000\<proc>\PORTS\<portname>\BOOTS\INSTALL\INI\JavaDemo.ini`

Note

For SH-3 based reference platforms, <proc> is SH3. For SH-4 based reference platforms, <proc> is SH4.

For the SH7709SE01 Reference Board, <portname> is SH7709. For the SH7709ASE01 Reference Board, <portname> is SH7709ASE. For the EBX7709 Reference Platform, <portname> is EBX7709. For the SH7750SE01, <portname> is SH7750SE.

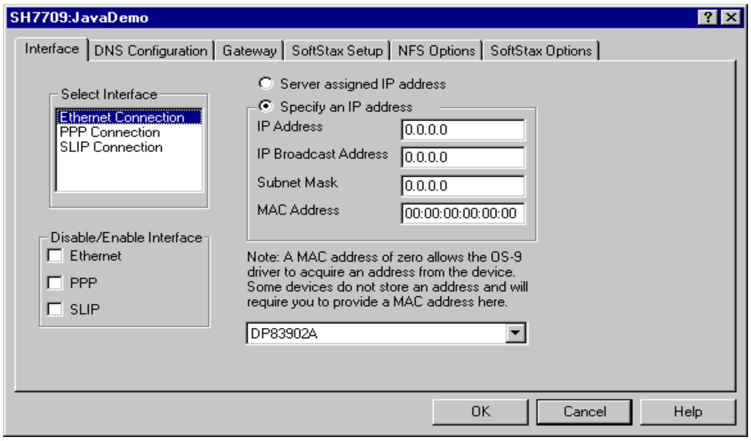
For subsequent uses of a configuration, Configuration Wizard automatically adds the board model to the beginning of the configuration name. Do not attempt to modify this portion of the name.

- Click `OK`

For More Information

Refer to the online help available with the Configuration Wizard for additional information about using the Configuration Wizard.

Step 2. 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:



SH7709:JavaDemo

Interface | DNS Configuration | Gateway | SoftStax Setup | NFS Options | SoftStax Options

Select Interface
- Ethernet Connection
- PPP Connection
- SLIP Connection

Disable/Enable Interface
- [ ] Ethernet
- [ ] PPP
- [ ] SLIP

( ) Server assigned IP address
(•) Specify an IP address

IP Address	0.0.0.0
IP Broadcast Address	0.0.0.0
Subnet Mask	0.0.0.0
MAC Address	00:00:00:00:00:00

Note: A MAC address of zero allows the OS-9 driver to acquire an address from the device. Some devices do not store an address and will require you to provide a MAC address here.

DP83902A

OK | Cancel | Help

Step 3. Select `Specify An IP Address` and fill in the Ethernet address text boxes with the Ethernet address information for your target:

- `IP Address` - your IP address in this field
- `IP Broadcast` - your IP Broadcast in this field
- `Subnet Mask` - your Subnet Mask in this field
- `MAC Address` - your MAC address in this field (may be optional, or may not appear, depending on your target)

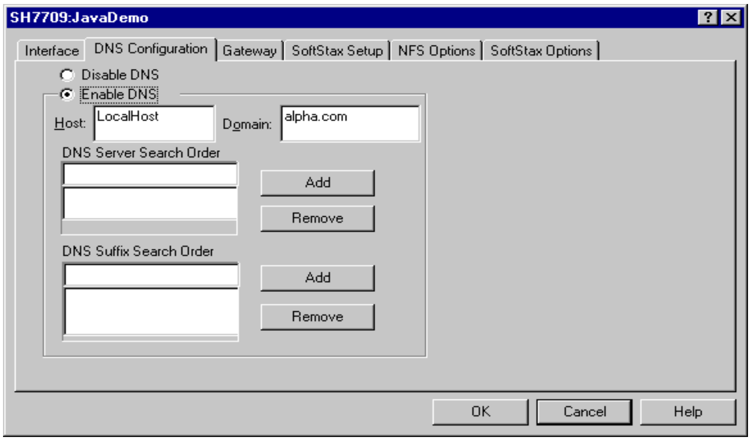
Note

You may need to contact your network administrator for the Ethernet addressing information.

Step 4. Click the `Ethernet` check box to select it. A check mark will appear in the box to show it is selected.

Step 5. Make sure the name of the Ethernet interface chip appears in the drop down combo box. For the SH7709SE01, SH7709ASE01, and SH7750SE01, be sure 'DP83902A' is selected. For the EBX7709 be sure '3COM PC CARD Socket #0' is selected.

Step 6. Select the `DNS Configuration` tab at the top of the window. The following window appears:



Disable DNS
[✓ Enable DNS]
Host: LocalHost
Domain: alpha.com
DNS Server Search Order
DNS Suffix Search Order

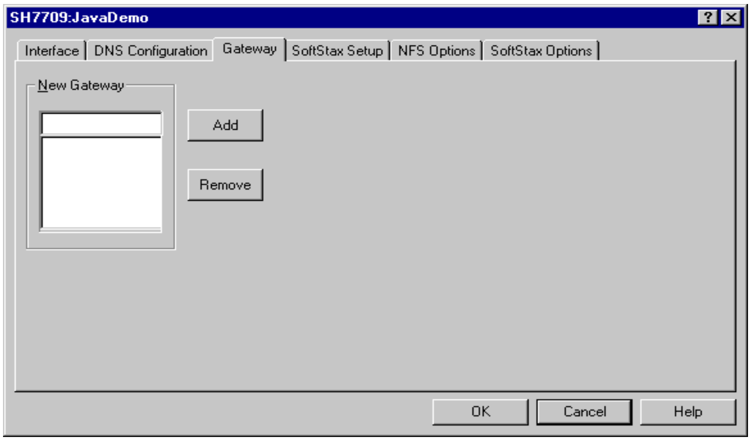
Complete the following fields:

- 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

You may need to contact your network administrator for the DNS information.

Step 7. Select the `Gateway` tab at the top of the window. The following window appears:



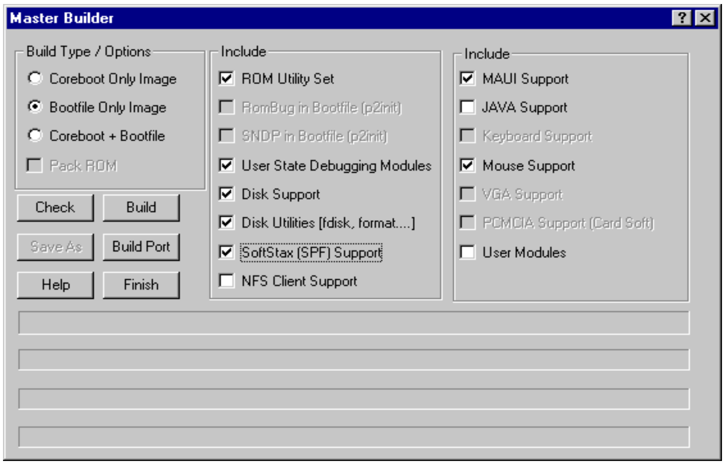
Complete the following:

- Enter your New Gateway network address. Add as many addresses as are valid for your location.
- Click `OK` to close the window.

Note

You may need to contact your network administrator for the Gateway information.

Step 8. Build the JavaDemo bootfile image. From the Configuration Wizard configuration window, select `Configure -> Build Image`. The following window appears:



Build Type / Options
- ○ Coreboot Only Image
- ● Bootfile Only Image
- ○ Coreboot + Bootfile
- ☐ Pack ROM

Include
- ☑ ROM Utility Set
- ☐ RomBug in Bootfile (p2init)
- ☐ SNDP in Bootfile (p2init)
- ☑ User State Debugging Modules
- ☑ Disk Support
- ☑ Disk Utilities [fdisk, format....]
- ☑ SoftStax (SPF) Support
- ☐ NFS Client Support

Include
- ☑ MAUI Support
- ☐ JAVA Support
- ☐ Keyboard Support
- ☑ Mouse Support
- ☐ VGA Support
- ☐ PCMCIA Support (Card Soft)
- ☐ User Modules

Step 9. Make sure the **Bootfile Only Image** build type is selected.

Step 10. In the Master Builder window, click `Build`. The bootfile is created in `C:\MWOS\OS9000\<portproc>\PORTS\<portname>\BOOTS\INSTALL\PORTBOOT` directory.

Step 11. Click `Save As`. A **Save As** dialog box appears.

Step 12. Navigate to the root directory of the PCMCIA IDE card. The `c:\` directory is an example of a root directory.

Step 13. Type `os9kboot` into the File name text box.

Step 14. Click `Save`. The bootfile is saved to the root directory.

Step 15. Click `Finish`.

Step 16. Exit the Configuration Wizard.

Running the PersonalJava Demo Bootfile

Note
The procedures in this stage use the G: drive on your Windows host to designate on which drive the PCMCIA IDE card is mapped.

Complete the following steps on your PC:

Step 1. Create the PJAVA directory on your PCMCIA disk.

Using the Windows Explorer, navigate to your PCMCIA disk and create a directory at the root called PJAVA.

Step 2. Navigate to the following directory:

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

Copy all `.mod` files, `go.demo`, and `pjruntime` to the G:\PJAVA directory.

Step 3. Turn off the power to the SuperH reference board.

Note
Inserting and removing a PCMCIA card with the power on is not supported. Damage may occur to the PCMCIA card if it is inserted or removed while power is applied to the board.

Step 4. Remove the PCMCIA IDE card from the computer.

Step 5. **For the SH7709SE01, SH7709ASE01, and SH7750SE01:** Turn the card upside down, and slide the card into the PCMCIA slot (located on the underside of the reference board) until the card snaps onto the pins. **For the EBX7709:** Slide the card into the bottom slot of the EBX7709 Reference Platform until the card snaps onto the pins.

Step 6. Start your virtual terminal program. HyperTerminal is an example of a virtual terminal program.

Step 7. Apply power to the board. Depending on how your system is configured, either the boot menu appears or you automatically boot from the PCMCIA card.

Step 8. If the boot menu appears, type `pcm_pc`. The `os9kboot` built from the `JavaDemo` configuration file boots the reference board using the PCMCIA IDE card.

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.
A valid OS-9000 bootfile was found.
Loading PersonalJava run-time components...
Loading PersonalJava for OS-9 examples...
+3
+4
Starting PersonalJava...
+6
$
```

On the SuperH board's monitor, the PersonalJava LaunchPad application displays after a few moments. You can select the demo you want to run from the LaunchPad application.

Running Java Applets

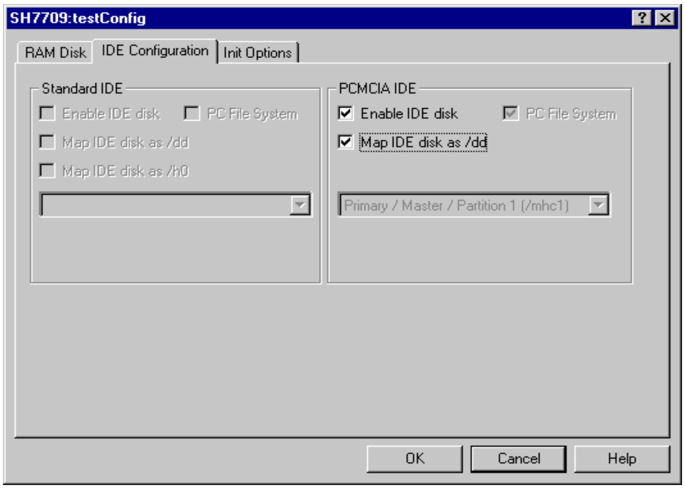
This section describes what you need to do to prepare your SuperH reference board to run the Sun Demo applets or your own applets. The JDK v1.1 demo applets are contained in `MWOS/DOS/jdk1.1.8/demo` on your host computer.

Create a Java ready bootfile

For your system to run PersonalJava, you need to have a bootfile that contains networking modules and Multimedia Application User Interface (MAUI) modules. If you followed the directions in the OS-9 board guide for your reference board, you should have the networking modules already loaded. Since including the MAUI support modules into the bootfile was not part of the directions in those manuals, you need to rebuild the bootfile to include the MAUI support modules.

Step 1. Follow the directions in your board guide for building a bootfile. Stop before you make a build and come back to step 2 in this section.

Step 2. Click on the System Disk Configuration button in the bootfile button group, then click on the IDE Configuration tab. You should see the contents of the IDE configuration tab. Click on `Map IDE disk as /dd`.


SH7709:testConfig

RAM Disk | IDE Configuration | Init Options

Standard IDE
- [ ] Enable IDE disk
- [ ] PC File System
- [ ] Map IDE disk as /dd
- [ ] Map IDE disk as /h0

PCMCIA IDE
- [x] Enable IDE disk
- [x] PC File System
- [x] Map IDE disk as /dd

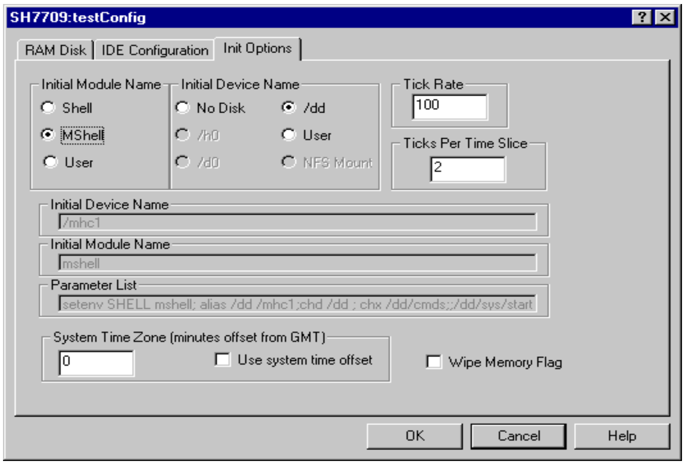
Primary / Master / Partition 1 (/mhc1)

Note

If the `Map IDE disk as /dd` checkbox is grayed out, then you need to deselect the `Map RAM disk as /dd` checkbox.

Step 3. Click on the `Init Options` tab.

Step 4. Select `MShell` as the initial module name and `/dd` as the initial device.



SH7709:testConfig

RAM Disk | IDE Configuration | Init Options

Initial Module Name: Shell [✓ MShell] User

Initial Device Name: No Disk /h0 /d0 [✓ /dd] User NFS Mount

Tick Rate: 100

Ticks Per Time Slice: 2

Initial Device Name: /mhc1

Initial Module Name: mshell

Parameter List: setenv SHELL mshell; alias /dd /mhc1;chd /dd ; chx /dd/cmds;;/dd/sys/start

System Time Zone (minutes offset from GMT): 0 ☐ Use system time offset

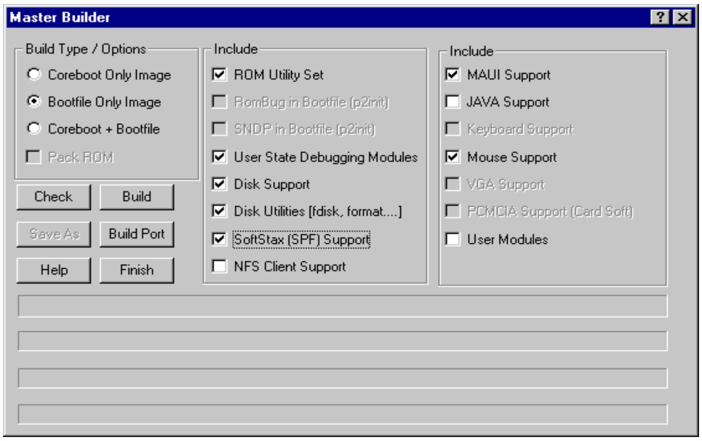
☐ Wipe Memory Flag

OK | Cancel | Help

Step 5. Click OK to close the dialog box.

Step 6. Click the `Build Image` button to display the **Master Builder** window.

Step 7. Make sure that the check boxes in the Master Builder window match the following illustration:



Master Builder

Build Type / Options: Coreboot Only Image [✓ Bootfile Only Image] Coreboot + Bootfile ☐ Pack ROM

Include:
- ☑ ROM Utility Set
- ☐ RomBug in Bootfile (p2init)
- ☐ SNDP in Bootfile (p2init)
- ☑ User State Debugging Modules
- ☑ Disk Support
- ☑ Disk Utilities [fdisk, format....]
- ☑ SoftStax (SPF) Support
- ☐ NFS Client Support

Include:
- ☑ MAUI Support
- ☐ JAVA Support
- ☐ Keyboard Support
- ☑ Mouse Support
- ☐ VGA Support
- ☐ PCMCIA Support (Card Soft)
- ☐ User Modules

Step 8. Build the bootfile by clicking `Build` in the Master Builder screen.

Step 9. Load the bootfile onto the SuperH board by following the directions in your reference board’s OS-9 for SuperH board guide.

Copy the PJava support files onto your target

Once you have created the bootfile, you need to copy the support files for Personal Java onto your PCMCIA card. Use the `mat` utility to extract the contents of the pjava.mat file onto the PCMCIA card. First create an MWOS directory on the root of your hardcard. Then cd to that directory, and use a command like the following to extract the archived files (this example assumes you installed the Personal Java for OS-9 package into MWOS on your C: drive):

```
mat -to -xv c:/mwos/OS9000/SH4/PORTS/SH7750SE/PJAVA/pjava.mat
```

Copy the Applet to the Target

Once you have copied the Personal Java for OS-9 support files onto your hardcard, you need to copy the applet you want to run to the PCMCIA card.

WARNING

When removing and inserting the IDE PCMCIA card into your board, make sure the power to the board is off. Otherwise, you may damage the PCMCIA card and the board.

Run the loadjava script

The loadjava script sets up the OS-9 environment variables, loads the MAUI support modules into memory, initializes the modules, runs the MAUI input process, and starts the PersonalJava window manager.

Note

The loadjava script must be run every time the board is booted.

For More Information

You can set up the loadjava script to run every time by defining it as a system startup script. See “Making a Startup File” in Chapter 9 of ***Using OS-9***.

Step 1. Insert your PCMCIA card into your target, and turn on the power.

Step 2. The target should boot up to a console prompt.

Step 3. Change directories to the `SYS` directory.

```
chd /mhc1/SYS
```

Step 4. Enter the following commands to run the loadjava script:

```
tmode nopause
profile loadjava
```

As the loadjava script executes, you will see a series of messages scroll up the screen.

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.

```
MWOS=/dd/MWOS
JAVA_HOME=/dd/MWOS/SRC/PJAVA
CLASSPATH=/dd/MWOS/SRC/PJAVA/LIB/classes.zip:.
PATH=/dd/MWOS/OS9000/SH3/CMDS:$PATH
LD_LIBRARY_PATH=/dd/MWOS/OS9000/LIB/SHARED
PORT=/term
HOME=/dd
USER=java_user
TZ=CST
```

Note

If you are using the SH7750SE01 Solution Engine, the path environment variable is different from the above listing. For the SH7750SE01 solution engine, the definition should be as follows:

```
PATH=/dd/MWOS/OS9000/SH4/CMDS:$PATH
```

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 loadjava script, you can run any desired example applet.

Note

If your applet requires resources that are not present on the SuperH board (sound for example), then it may not work correctly.

Step 1. Change to the directory on your PCMCIA card containing the applet.
Example: `chd /mhc1/MWOS/DOS/jdk1.1.7B/demo/TicTacToe/1.1`

Step 2. Run the applet by typing the following:

```
pappletviewer <htmlfile>
```

or

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

Considerations for Running Your Own PersonalJava Applications

The JavaDemo Configuration Wizard configuration file and go.demo script took care of a number of details that you should be aware of when running your own applications. The following section is a complete list of details that need to be addressed. Many of them are taken care of in the loadjava 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 will have to be accomplished by setting the 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 (e.g. for a diskless environment, variables set to disk paths need not be set). These include the following:
   - 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 modules include the following:

- `winmgr` – PersonalJava 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 starter application
- `libjavai.so -` PersonalJava 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 (e.g. from a modman archive)

3. Before running a graphical PJAVA application, Window Manager must be started. Before running Window Manager, the MAUI input process must be started. 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 applications.

Appendix A: Java Load Script

This appendix lists the Java Load Script needed to start Java on your target platform.

Example Java Load Script

This 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.

SuperH 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 $MWOS/OS9000/SH4/CMDS:$PATH
setenv LD_LIBRARY_PATH $MWOS/OS9000/SH4/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/SH4/CMDS/%winmgr
load -d $MWOS/OS9000/SH4/CMDS/winmgr.dat
load -d $MWOS/OS9000/SH4/PORTS/SH7750SE/CMDS/%resfile

*
* Load the pre-generated libmawt Color Cube module
*
load -d $MWOS/OS9000/SH4/PORTS/SH7750SE/LIB/SHARED/libmawt_0.dat

*
* Load fonts for "font.properties"
*
load -d $MWOS/OS9000/SH4/ASSETS/FONTS/AGFA/MT/*
load -d $MWOS/OS9000/SH4/ASSETS/FONTS/AGFA/TT/utt.ss

*
* Initialize the keyboard and mouse devices, if needed
*
iniz m0 t3

*
* 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
```