



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

Version 3.1

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

Revision E
April 2001

Copyright and publication information

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

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.

April 2001
Copyright ©2001 by RadiSys Corporation.
All rights reserved.

EPC, INtime, iRMX, MultiPro, RadiSys, The Inside Advantage, and ValuPro are registered trademarks of RadiSys Corporation. ASM, Brahma, DAL, DAQ, MultiPro, SAIB, Spirit, and ValuePro are trademarks of RadiSys Corporation.

DAVID, MAUI, OS-9, and OS-9000, are registered trademarks of RadiSys Microware Communications Software Division, Inc. FasTrak, Hawk, SoftStax, and UpLink are trademarks of RadiSys Microware Communications Software Division, Inc.

† All other trademarks, registered trademarks, service marks, and trade names are the property of their respective owners.

Table of Contents

Chapter 1: Introduction **5**

- 6 PersonalJava Solution for OS-9 Runtime Components
- 7 OS-9
- 7 Networking
- 8 Graphics
- 9 Java Virtual Machine (JVM)
- 9 Applications and Applets
- 10 Additional Java Tools
- 10 Running Java On a Diskless System
- 11 Java Development Tools
- 12 Windows® Java Development Kit (JDK)

Chapter 2: Running PersonalJava Demos **13**

- 15 System Requirements
- 16 Installing PersonalJava Solution for OS-9 on Your Target
- 17 Stage 1: Building an OS-9 Bootfile Using the Configuration Wizard
- 24 Stage 2: On the PC
- 26 Stage 3: On the Target
- 27 Running Java Applets
- 33 Considerations for Running Your PersonalJava Applications

Appendix A: Java Load Script **35**

- 36 Example Java Load Script
- 36 StrongARM loadjava Script

Chapter 1: Introduction

This manual provides the information you need to get started with PersonalJava Solution for OS-9.



Note

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



Note

Before you 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.

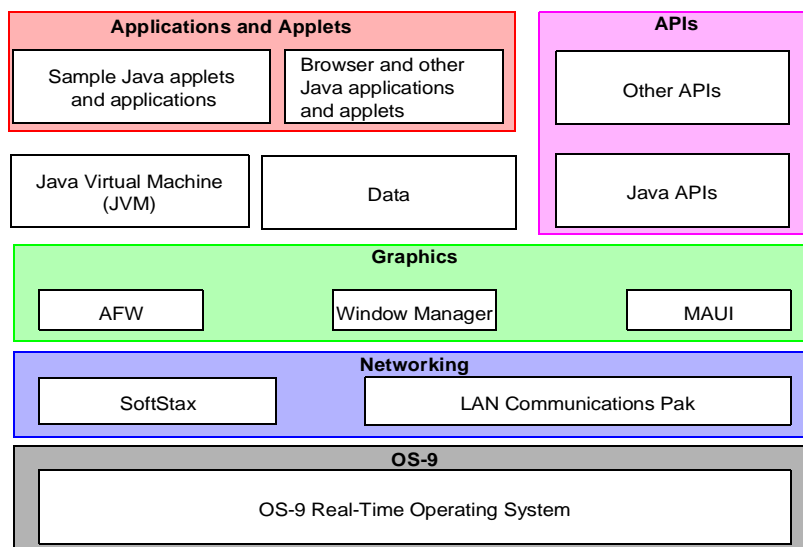


PersonalJava Solution for OS-9 Runtime Components

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



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 BLS package.

OS-9

At the core of PersonalJava Solution for OS-9 is OS-9 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 the LAN Communications Pak.

LAN Communications Pak

The Microware LAN Communications Pak software consists of a TCP/IP protocol stack with UDP support, 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: MAUI, Window Manager, the Application Framework (AFW), and the Java Abstract Windowing Toolkit (AWT).

Multimedia Application User Interface (MAUI)

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

Window Manager

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

Application Framework

The AFW is a class library that contains the code necessary to display Graphical 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 Java Virtual Machine (JVM). Java applications are comprised of Java classes consisting of byte codes.

Java byte codes are machine-independent and interpreted by the JVM. 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 JVM 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 Applets

Several sample applets 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 times 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.



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 architecturally neutral, you can develop your 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 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 on the host machine 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 and your Java applets and applications.

Microware's PersonalJava Solution can run in a diskless environment. In a truly embedded system, the modules that are loaded have been placed in the boot and the script's commands accomplished programmatically.



Note

The following procedures assume that your target system is diskless. Refer to ***Using PersonalJava Solution for OS-9*** for information about using a disk-based target system.





Note

You must install Enhanced OS-9 for StrongARM before installing PersonalJava Solution for OS-9.

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

This chapter describes using PersonalJava Solution for OS-9 on the Intel SideARM, ADS Thin Client, ADS Graphics Client, ADS Graphics Master, or Intel Assabet evaluation system.

System Requirements



For More Information

The hardware and software requirements for using PersonalJava Solution for OS-9 on your host and target are listed in the appropriate ***Enhanced OS-9 Board Guide***.

Installing PersonalJava Solution for OS-9 on Your Target

Before you begin, set up your target with the correct boot ROM's and establish a virtual terminal connection between the PC and the target. This connection is described as the target console throughout the remainder of this document. If you are using a physical terminal as the target console instead of a virtual terminal, installation instructions remain identical.



For More Information

For more information about setting up your target board and which kind of PC Card devices are supported, see the appropriate ***Enhanced OS-9 Board Guide***.

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

- **Stage 1: Building an OS-9 Bootfile Using the Configuration Wizard**
- **Stage 2: On the PC**
- **Stage 3: On the Target**

Stage 1: Building an OS-9 Bootfile Using the Configuration Wizard



Note

The Java Development Kit (JDK) as shipped from Sun is targeted strictly at desktop environments. PersonalJava Solution for OS-9 may be used on either disk-based or diskless systems. The examples you are running require a system disk only for booting.

To build a bootfile using the Configuration Wizard, complete the following steps:

- Step 1. From the Windows **Start** menu, select **Programs** -> **Microware** -> **Enhanced OS-9 for StrongARM** -> **Configuration Wizard**.
- Step 2. Provide the following information on the opening screen:
- Fill in the `MWOS Location` field with the location of the MWOS tree installed when you installed your OS-9 for Embedded Systems or OS-9 Board Level Solution.
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 has been installed into the MWOS tree on your PC at `C:\MWOS\OS9000\ARMV4\PORTS\<portname>\BOOTS\INSTALL\INI\JavaDemo.ini`
 - Click **OK**.



Note

The Configuration Wizard automatically prepends the board model to the configuration name. Do not type this portion of the name.



For More Information

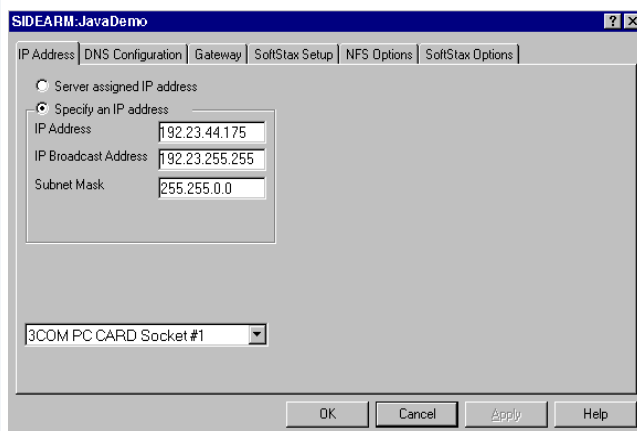
Refer to the online Help available in Configuration Wizard for additional information about using the Wizard.

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



Note

It is not necessary to use a network card to complete this demonstration. If you do not have a 3Com Etherlink III or equivalent PC card Ethernet adapter, you can skip to step six.



Fill in the Ethernet Setup fields for the high-level Ethernet connection with the following information for your target:

- **IP Address**—type your IP address in this field
- **IP Broadcast**—type your IP Broadcast in this field
- **Subnet Mask**—type your Subnet Mask in this field

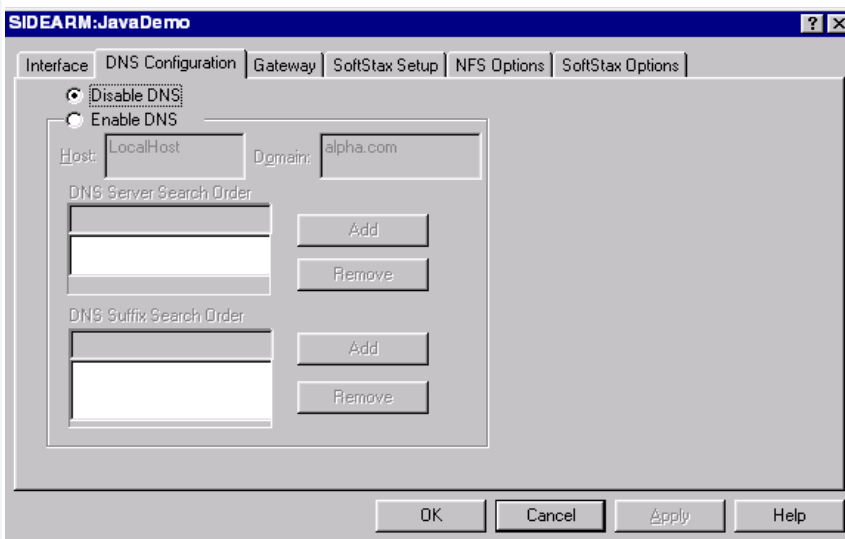


Note

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

If you are using a SideARM system, select 3COM PC CARD Socket #1 in the option box at the bottom of the dialog. If you are using a ThinClient, GraphicsClient, or GraphicsMaster system, select **Onboard (LAN91C94)**.

- Step 4. Select the **DNS Configuration** tab at the top of the window. The following window appears:



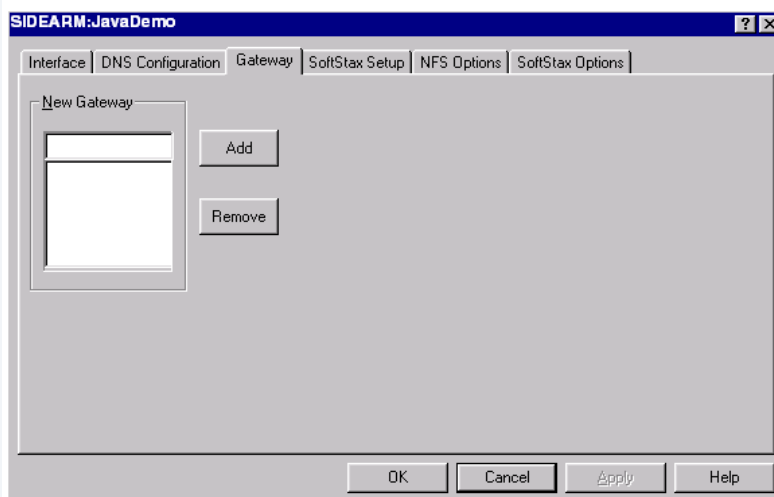
The screenshot shows the "SIDEARM:JavaDemo" window with the "DNS Configuration" tab selected. The "Interface" tab is also visible. The "DNS Configuration" tab contains the following elements:

- Radio buttons for "Disable DNS" (selected) and "Enable DNS".
- Text fields for "Host" (containing "LocalHost") and "Domain" (containing "alpha.com").
- A section titled "DNS Server Search Order" with a list box and "Add" and "Remove" buttons.
- A section titled "DNS Suffix Search Order" with a list box and "Add" and "Remove" buttons.
- Buttons for "OK", "Cancel", "Apply", and "Help" at the bottom.

Fill in the following fields with the information for your target:

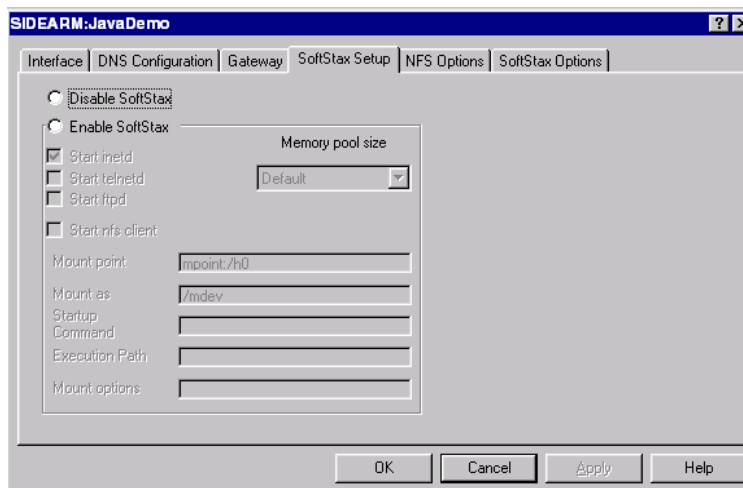
- Select **Enable DNS**.
- Enter your target's host name in the `Host` field.
- Enter your target's domain name in the `Domain` field.
- Enter your target's DNS Server Search Order addresses. Add as many addresses as are valid for your location.
- Enter your target's DNS Suffix Search Order. Add as many suffixes as are valid for your location.

Step 5. Select the **Gateway** tab at the top of the window. The following window appears:



Enter your new gateway network address. Add as many addresses as are valid for your location.

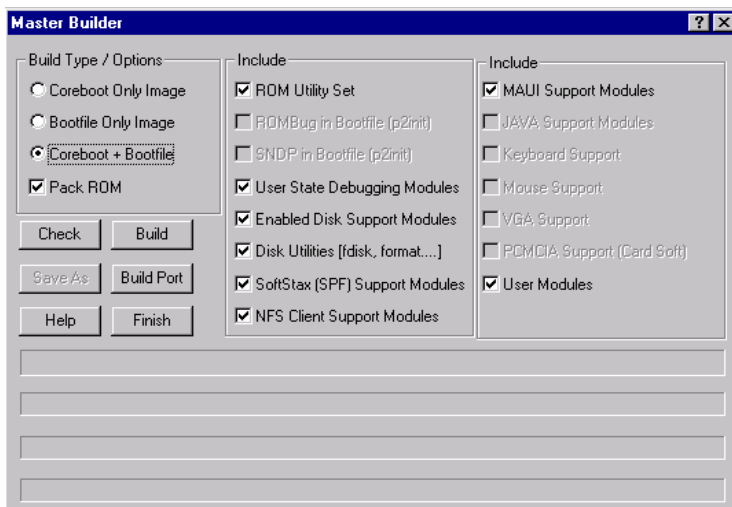
Step 6. Select the **SoftStax Setup** tab at the top of the window. The following window appears:



Fill in the following fields with your information:

- Select **Enable SoftStax**.
- Select the **Start inetd** check box.
- Click on the **OK** button at the bottom of the screen to close the window.

Step 7. Build the Java Demo bootfile image. From the Configuration Wizard menu options, select **Configure** -> **Build Image**. The following window appears:



Step 8. In the Master Builder window, click the **Build** button. The file `C:\MWOS\OS9000\ARMV4\PORTS\<portname>\BOOTS\INSTALL\PORTBOOT\os9kboot` is created. If the build does not complete successfully, refer to **Troubleshooting** in the Configuration Wizard helpfile for more information about errors you may receive.

Step 9. Click the **Finish** button.

Step 10. Exit the Configuration Wizard.

Stage 2: On the PC



Note

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

Complete the following steps on the host PC:

- Step 1. Install an empty PCMCIA ATA Flash card into the PCMCIA slot on your PC. This card should have a minimum capacity of 8MB.
- Step 2. Copy the file `os9kboot` to the root directory of the PCMCIA ATA card.

Using the Windows Explorer, navigate to the following directory:

```
E:\MWOS\OS9000\ARMV4\PORTS\<portname>\BOOTS\INSTALL\PORTBOOT
```

Right click on the file `os9kboot` and drag it to the drive letter G:\ to copy it to the PCMCIA card.

- Step 3. Create the PJAVA directory on the PCMCIA ATA card.

Using the Windows Explorer, navigate to the following directory:

```
G:\
```

Select **File** -> **New** -> **Folder**. Change the name to PJAVA.

- Step 4. Copy the PersonalJava modules and startup script to the PCMCIA ATA card.

Using the Windows Explorer, navigate to the following directory:

```
E:\MWOS\OS9000\ARMV4\PORTS\<portname>\CMDS\BOOTOBS\PJAVA
```

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

Step 5. Convert the line ending in go.demo from MS-DOS to OS-9.

Using a DOS shell, change directory to G:\PJAVA and execute this command line:

```
cudo -co go.demo
```

Step 6. Turn off the power to the target system.



WARNING

Inserting and removing a PCMCIA card with the power on is not supported in this version of Enhanced OS-9 for StrongARM. The PCMCIA card may be damaged if it is inserted or removed while power is applied to the system.

Step 7. Remove the PCMCIA card from the host computer.

Stage 3: On the Target

-
- Step 1. For a SideARM system insert the PCMCIA card into the top slot (furthest away from the printed circuit board) until the card is firmly seated. For a ThinClient system, insert the PCMCIA card into the bottom slot (closest to the circuit board). For GraphicsClient, GraphicsMaster, and Assabet, insert the card into the only PCMCIA slot.
- Step 2. Apply power to the system. If your ROMs require you to specify a boot device, choose the IDE PCMCIA car.

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

```
OS-9000 Bootstrap for the ARM
```

```
OS-9000 kernel was found
```

```
+3
```

```
+5
```

```
Loading PersonalJava run-time components
```

```
+6
```

```
+7
```

```
Starting PersonalJava...
```

```
+9
```

```
$
```

On the OS-9 target system's display, you should see the PersonalJava LaunchPad application's window.

Running Java Applets

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

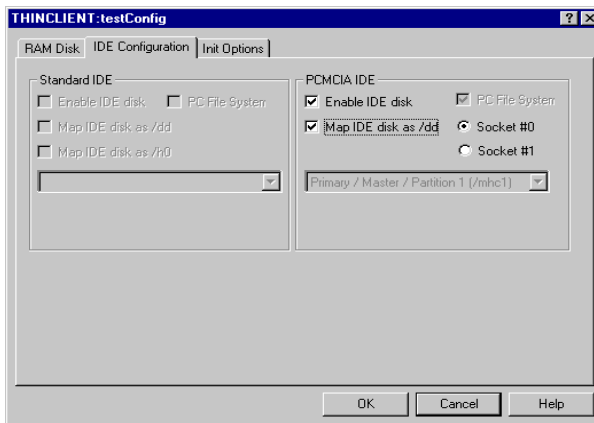
In the previous example the Java runtime objects were combined into an OS-9 module that was loaded into memory at boot-time. That approach is useful in diskless environments. In this example we show how to use Java on a target board that supports local disk. The two approaches differ in the way the Java objects are loaded into memory, but the behavior of the Java runtime environment is identical, regardless of which approach you use.

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 Enhanced 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.

To create a Java-ready bootfile, complete the following steps:

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

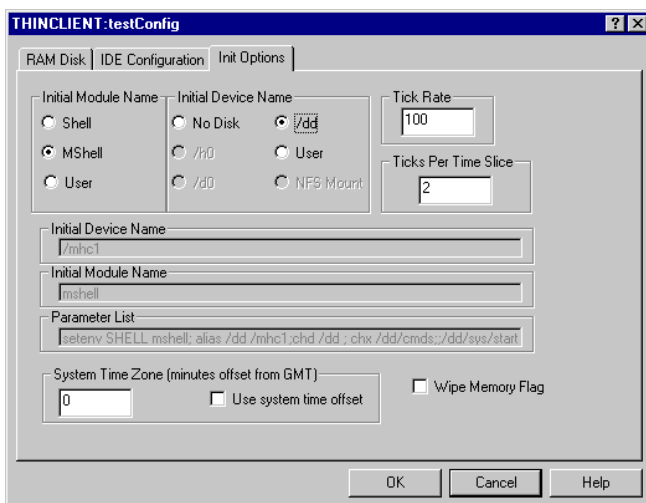


Note

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.

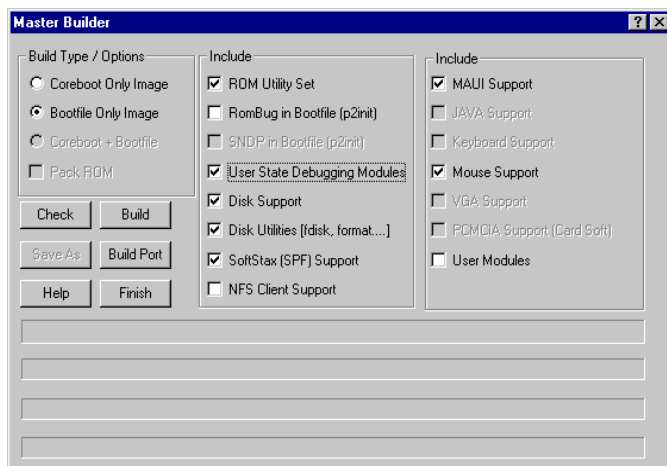
Step 4. Select **MShell** as the initial module name and **/dd** as the initial device.



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:



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

Step 9. Load the bootfile onto the StrongARM board by following the directions in your reference board's Enhanced OS-9 for StrongARM 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/ARMV4/PORTS/<port>/PJAVA/pjava.mat
```

Copy the Applet to the Target

Once you have created the bootfile you need to copy the applet you want to run to the IDE 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 10. Change directories to the `SYS` directory by typing the following on the command line:

```
chd /mhcl/SYS
```

- Step 11. Enter the following commands to run the `loadjava` script:

```
tmode nopause
profile loadjava
```

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

- Step 12. Type the following command to display the environment variables:

```
printenv
```

- Step 13. 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/ARMV4/CMD:$PATH
LD_LIBRARY_PATH=/dd/MWOS/OS9000/ARMV4/LIB/SHARED
PORT=/term
HOME=/dd
USER=java_user
TZ=CST
```

- Step 14. Make sure the `maui_inp` and Window Manager processes are running by typing the following command:

```
procs
```

You should see a listing of the processes that are currently running on your reference board. It looks 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 15. 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 StrongARM 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/jdk1.1.8/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 PersonalJava Applications

The JavaDemo Configuration Wizard configuration file and go.demo (in the JavaDemo example) and the structure of the pjava.mat file (in the “Running Java Applets” example) 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.



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 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
 - `pjava` – PersonalJava starter application
 - `libjavai.so` – PersonalJava virtual machine
 - `libmawt.so` – AWT shared library module
 - `libmawt_0.dat` – pre-computed color palette
 - `libjavafile.so` – I/O handling native methods
 - `libmath.so` – arbitrary precision math 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 an example script that can be used to start Java on your target platform. An implementation of this script called *loadjava* is placed in the SYS directory when you install the pjava.mat file onto your target system's local storage device.



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.

StrongARM loadjava Script

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

```
-tnp
* * * * *
* Load script for PersonalJava v3.1
*
*
* Set environment variables
*
setenv MWOS /mhcl/MWOS
setenv JAVA_HOME /mhcl/MWOS/SRC/PJAVA
setenv CLASSPATH /mhcl/MWOS/SRC/PJAVA/LIB/classes.zip:.
setenv PATH /mhcl/MWOS/OS9000/ARMV4/CMDS:$PATH
setenv LD_LIBRARY_PATH /mhcl/MWOS/OS9000/ARMV4/LIB/SHARED
setenv PORT /term
setenv HOME /mhcl
setenv USER java_user
setenv TZ CST

*
* Load the Java Window Manager application and a file containing
* the resource module 'Stock.res'
*
*   winmgrs - Simplest/smallest Window Manager - No window frames
*   winmgr  - Default Window Manager
*   winmrg  - Debug version - "Send Shutdown Message" & "Dump Window Tree" functionality
*
*   Stock_8.res - 8-bit bitmap and cursor support (default)
*   Stock_9.res - 16-bit bitmap and cursor support
*
let winmgr = "winmgr";* winmgr, winmrg, or winmgrs
load -d /mhcl/MWOS/OS9000/ARMV4/CMDS/%winmgr
*
load -ld /mhcl/MWOS/OS9000/ARMV4/PORTS/THINCLIENT/CMDS/stock_8.res
*load -ld /mhcl/MWOS/OS9000/ARMV4/PORTS/THINCLIENT/CMDS/stock_9.res

*
* Load the Window Manager Settings
load -ld /mhcl/MWOS/OS9000/ARMV4/CMDS/winmgr.dat
```

```
*
* Load the pre-generated libmawt Color Cube module
load -ld /mhc1/MWOS/OS9000/ARMV4/PORTS/THINCLIENT/LIB/SHARED/libmawt_0.dat

*
* Load fonts for "font.properties"
*
load -d /mhc1/MWOS/OS9000/ARMV4/ASSETS/FONTS/AGFA/MT/*
load -d /mhc1/MWOS/OS9000/ARMV4/ASSETS/FONTS/AGFA/TT/utt.ss

*
* Initialize the keyboard and mouse devices
*
iniz m0 kx0

*
* The two lines below can be removed if non-graphical Personal Java applications
* are to be used.

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

*
* Start window manager loaded above
*
%winmgr ^250 <>>>/nil &

-nt
```

Product Discrepancy Report

To: Microware Customer Support

FAX: 515-224-1352

From: _____

Company: _____

Phone: _____

Fax: _____ Email: _____

Product Name:

Description of Problem:

Host Platform _____

Target Platform _____



MICROWARE SOFTWARE