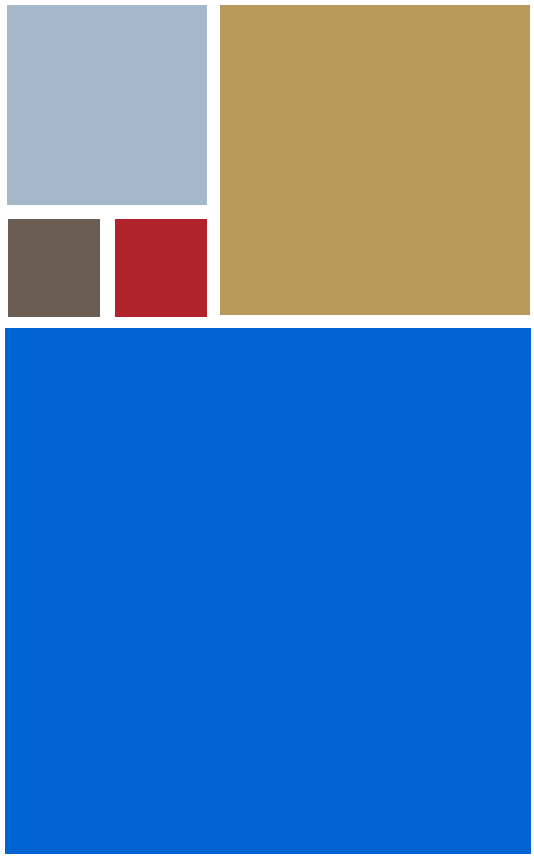


Home

Getting Started with PersonalJava™ Solution for OS-9® (X86)

Version 3.1

RadiSys
THE POWER OF WE

www.radisys.com

Revision C • 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

Contents

1 Introduction

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

- Refer to the current version of *OS-9 Release Notes* for possible last-minute updataes to PersonalJava Solution for OS-9 or the StrongARM board.
- Refer to the CD-ROM insert for information about installing PersonalJava Soluction for OS-9 on your Windows-based host platform.

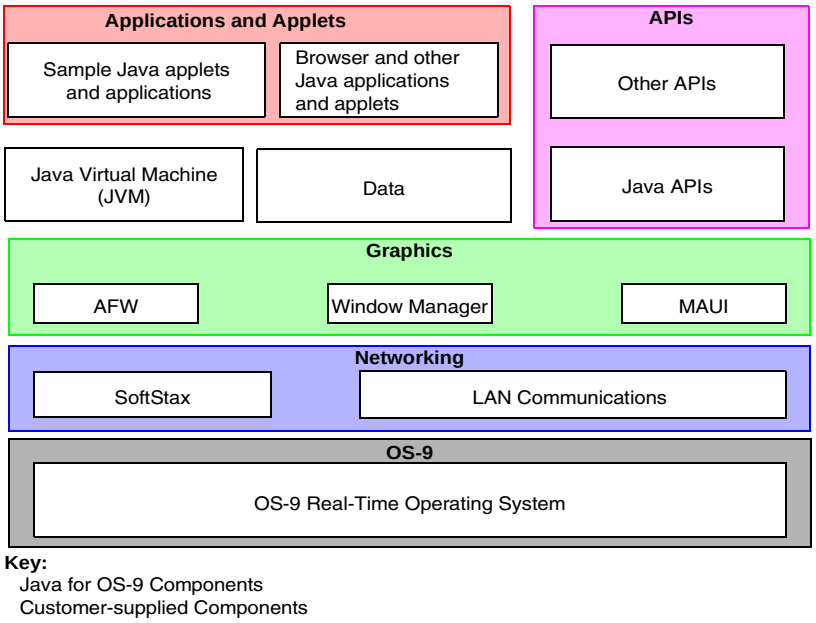
Before you proceeding, be certain you have installed either OS-9 for Embedded Systems or the OS-9 Board Level Solution (BLS) for your processor, on your Windows-based host system. If you do not have either of these packages, contact your OS-9 supplier.

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



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 LAN Communications.

LAN Communications

The Microware LAN Communications 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.

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

2 Running PersonalJava Demos

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

Microware's PersonalJava Solution can run in a disk based system or in a completely diskless environment. The examples used in this chapter assume you are installing PersonalJava Solution on a system that includes a standard IDE disk; however, the disk can be a PCMCIA ATA flash device, a SCSI disk, or a RAM disk loaded through an Ethernet connection using FTP.

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.

You must install Microware OS-9 for x86 before you can install 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).

System Requirements

The hardware and software requirements for using PersonalJava Solution for OS-9 on your host and target are listed in the appropriate OS-9 board guide.

Installing PersonalJava Solution for OS-9

Installing the PersonalJava Solution for OS-9 files

PersonalJava Solution for OS-9 is first installed on the host system and then on the target X86/Pentium system.

Installing PersonalJava Solution onto the Host System

Step 1. Insert the CD-ROM containing PersonalJava Solution for OS-9 (X86) into your CD-ROM drive.

Step 2. The installation menu should come up automatically. If the installation menu fails to appear, navigate to the Autorun directory on the CD-ROM and double click on Autrorun.exe.

Step 3. Select `PersonalJava Solution for OS-9 (X86)` from the setup menu.

Step 4. Follow the directions in the installer windows. Enter the PersonalJava Solution for OS-9 password, when prompted. The password is located on the password card included with the PersonalJava Solution for OS-9 package.

Step 5. Enter the destination MWOS folder, when prompted. Microware OS-9 for X86 must have been previously installed in this folder.

Step 6. Select the components to install, either PersonalJava Solution, PersonalJava Documentation or both.

Step 7. Select the program folder. By default, the package installs into Microware OS-9 for X86, some Microware OS-9 packages created an OS-9 for X86 program folder. Verify that you install Personal Java in the same folder as the previously installed OS-9 for X86 product.

Step 8. Click `Next` to complete the install.

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 go onto the target are found in the MWOS directory on the host machine. The path to the files is as follows: `MWOS\OS9000\80386\PORTS\<port name>\PJAVA`.

The `PJAVA` folder contains two relevant items: `pjava.mat` and `readme.txt`. `pjava.mat` is a Microware Archive Tool (MAT) archive of the files to go on the target and `readme.txt` explains how to install the MAT archive onto the your disk.

Building the PersonalJava Demo Bootfile

This section discusses creating a disk-based boot for your X86 target hardware. It assumes that your target system is configured with a serial console, both floppy and IDE disk drives, VGA graphics and keyboard, a PS/2 style bus mouse, a supported network card and at least 16 MB of RAM. Refer to the OS-9 for PCAT Board Guide and the Wizard on-line help system for instructions on building a boot for other hardware options.

You should create a new disk based boot, even if you already have a target system running OS-9 for X86.

The OS-9 console must be moved to a serial port so that it does not conflict with the Java window manager running on the VGA/SVGA graphics hardware. This example assumes that the OS-9 console appears on COM1, and is connected to a host computer running a terminal emulation program such as Hyperterminal.

Follow the instructions in the OS-9 for PCAT Board Guide for detailed instructions on installing OS-9 on the target system and configuring a disk based boot image.

Once configured, your OS-9 target system should perform the following tasks:

- boot OS-9 from a hard disk
- bring the OS-9 system console up on a serial port, using an emulator running on the Windows host computer, such as hyperterminal
- have a working ethernet connection, tested by establishing a Telnet or FTP connection from the Windows host computer to the OS-9 target system
- have MAUI graphics configured on the target OS-9 system, tested by running the MAUI `fcopy` or `fdraw` demo programs on the target

Installing PersonalJava Solution on the Target Hardware

The procedures in this stage assume the following things:

- your target system boots from the hard disk without error
- you have an OS-9 system prompt, using the terminal emulation program running on your Windows host computer
- you have an Ethernet connection between the host and target machines, you can telnet and FTP to the target, from the Windows host computer
- you have a supported graphics card and have installed the MAUI graphics system

Complete the following steps on the target machine:

Step 1. Use FTP to download the pjava tar file (`x86_pjava.tar`) from the `MWOS\OS9000\80386\PJAVA` directory on the host computer. Perform the following steps:

```
cd MWOS\OS9000\80386\PORTS\PCAT\PJAVA
```

```
ftp <target>
User: super
Password: user
ftp> bin
ftp> cd /h0
ftp> send pjava.mat
ftp> quit
```

Step 2. Untar the pjava image by typing the following commands on the OS-9 system console.

```
$ chd /h0/MWOS
$ load -d /h0/CMDS/mat
$ tmode nopause
$ mat -x -v ../pjava.mat
```

Running Java Applets

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

Run the loadjava script

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

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

You can set up the loadjava script to run every time by defining it as a system startup script.

Refer to *Using OS-9* manual or the board guide for your processor for more information on startup files.

Step 1. From the OS-9 console, change directories to the `SYS` directory by typing the following on the command line:

```
chd /h0/SYS
```

Step 2. Enter the following commands to run the loadjava script:

```
tmode nopause
profile loadjava
```

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

Step 3. Type the following command to display the environment variables:

```
printenv
```

Step 4. Compare the listing on your screen with the following listing. Make sure that the listed environment variables are set correctly.

```
MWOS=/h0/MWOS
JAVA_HOME=/h0/MWOS/SRC/PJAVA
CLASSPATH=/h0/MWOS/SRC/PJAVA/LIB/classe
s.zip:.
PATH=/h0/MWOS/OS9000/80386/CMDS:$PATH
PORT=/term
HOME=/h0
USER=java_user
```

Step 5. Make sure the `maui_inp` process is running by typing the following command:

```
procs
```

You should see a listing of the processes that are currently running on your target computer. It should look similar to the following illustration.

```
Id PId Grp.Usr  Prior  MemSiz Sig S    CPU Time
Age Module & I/O
  2   0   0.0     128   52.00k   0 w        0.28
??? mshell <>>>term
  3   2   0.0     128   68.00k   0 s        0.05
??? telnetd <>>>nil
```

Step 6. Check that `maui_inp` is listed under the module heading.

You are now ready to run your applet.

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

Running an Applet

The best way to test the PJava installation on the OS-9 target is to download and run an applet. Several example applets were installed in the `\MWOS\DOS\jdk1.1.8\demo` directory on the Windows host. This example downloads and runs the Jumping Box applet.

The PersonalJava Solution Window Manager must be started before running this example applet. At the OS-9 console, type: `winmgr ^250 <>>>/nil&`

Step 1. Change to the JumpingBox demo directory on the Windows computer.

```
cd \mwos\dos\jdk1.1.8\demo\Fractal
```

Step 2. Use FTP to download the Jumping box example files.

```
ftp <target>
User: super
Password: user
ftp> bin
ftp> cd /h0
ftp> send example1.html
ftp> prompt
```

```
ftp> mput *.class
ftp> quit
```

Step 3. Run the applet from the OS-9 console by typing the following:

```
chd /h0
pappletviewer example1.html
or
pjava sun.applet.AppletViewer example1.html
```

Considerations for Running Your Own PersonalJava Applications

The loadjava 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.

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/ROM image.

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 items:

- 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

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 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 Solution
`libmawt.so`	AWT shared library module

`libmawt_0.dat`	pre-computed color palette
font modules	various modules related to font rendering
MAUI modules	various modules related to graphics support
`*.properties`	properties files needed by PersonalJava Solution (e.g. from a modman archive)

Before running a graphical PersonalJava application, the Window Manager must be started. Before running the window manager, the MAUI input process must be started. The following command lines show an example startup sequence:

- `maui_inp ^255 <>>>/nil &`
- `winmgr ^200 <>>>/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.

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.

X86 loadjava Script

The following example loadjava script is used on a X86 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 $MWOS/OS9000/80386/CMDS:$PATH
setenv LD_LIBRARY_PATH $MWOS/OS9000/80386/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/OS9000/80386/CMDS/%winmgr
load -d $MWOS/OS9000/80386/CMDS/winmgr.dat
load -d $MWOS/OS9000/80386/PORTS/PCAT/CMDS/%resfile

*
* Load the pre-generated libmawt Color Cube module
*
load -d $MWOS/OS9000/80386/PORTS/PCAT/LIB/SHARED/libmawt_0.dat

*
```

```
* Load fonts for "font.properties"
*
load -d $MWOS/OS9000/80386/ASSETS/FONTS/AGFA/MT/*
load -d $MWOS/OS9000/80386/ASSETS/FONTS/AGFA/TT/utt.ss

*
* Initialize the keyboard and mouse devices, if needed
*
iniz m0 k0

*
* 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
```