



RoboJDE™

Java™-enabled Robotics Software
Development Environment

User Guide

Version 1.5

www.ridgesoft.com

IMPORTANT NOTICE

Information in this document is furnished under license and may only be used in accordance with the terms of the license.

RIDGESOFT PRODUCTS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT.

RIDGESOFT PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS WHERE FAILURE COULD RESULT IN THE LOSS OF LIFE OR HAVE SERIOUS LIFE THREATENING OR ECONOMIC IMPACT.

Copyright © 2003-2006 by RidgeSoft, LLC. All rights reserved.

RidgeSoft™, RoboJDE™ and IntelliBrain™ are trademarks of RidgeSoft, LLC.

RidgeSoft, LLC
PO Box 482
Pleasanton, CA 94566
www.ridgesoft.com

Java™ and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All other brand or product names are trademarks of their respective owners.

Table of Contents

Introduction	1
RoboJDE Graphical User Interface	1
RoboJDE Virtual Machine.....	1
RoboJDE Class Library	1
Java Programming Benefits for Robotics Software Developers	2
Installation	4
Licensing	4
Getting Started	5
Starting the RoboJDE Graphical User Interface	5
Installing Your License Key	6
Downloading the RoboJDE Virtual Machine to Your Robotics Controller	6
Loading and Running the Hello World Example	7
Creating a New Project.....	7
Using the RoboJDE Graphical User Interface ...	9
Using Menus.....	9
Using the Tool Bar.....	12
Selecting the Load Location.....	12
Setting Your Project's Properties.....	13
Using the Edit Window.....	14
Editing Your Program.....	15
Building and Loading Your Program	15
Debugging Build Errors	16
Using the Run Window	17
Programming Your Robot.....	18
Using the RoboJDE Class Library	18
Robot Controller API Quick References	18
Learning from Examples.....	18
Understanding Errors, Exceptions and Stack Traces	19
Using Packages.....	21
Optimizing Memory Usage	22
Indications of Low Memory	22
Minimizing Stack Size	22
Minimizing Java Executable Size	23
Minimizing Program Data Size.....	24
Freeing Objects and Arrays that Are Not Needed.....	24
Avoiding Memory Fragmentation	24
Appendix A - Programming Native Methods....	25
Overview of Native Methods.....	25

Declaring Your Native Method.....	25
Adding the Method Signature to the Project Properties.....	26
Implementing Native Methods.....	26
Debugging Native Methods.....	27
Accessing I/O Registers and Control Ports.....	28

Introduction

By providing a modern, easy to use, software development environment built for robotics applications, the RoboJDE™ Java™-enabled robotics software development environment opens the door to object oriented software development for educational and hobby robotics projects. RoboJDE enables you to quickly and easily develop software to control your robot.

RoboJDE includes the major components shown in Figure 1:

- RoboJDE Graphical User Interface
- RoboJDE Class Library
- RoboJDE Virtual Machine

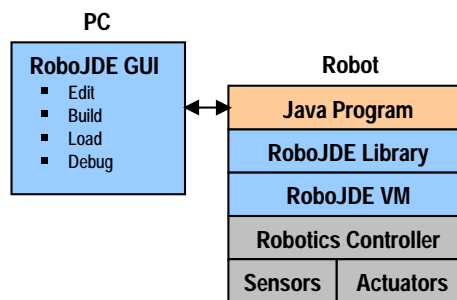


Figure 1 - RoboJDE Components

RoboJDE Graphical User Interface

The RoboJDE Graphical User Interface (GUI), shown in Figure 2, provides an easy to use tool to edit, build, load and debug robotics programs. The RoboJDE GUI runs on a PC running Microsoft Windows® and communicates with the robot controller using an RS232 serial connection.

RoboJDE Virtual Machine

The RoboJDE virtual machine executes Java programs on the robotics controller and provides multi-threading, thread synchronization, memory management, exception handling and device drivers. The virtual machine also provides protection against common programming errors by detecting errors such as null references, out-of-bounds array indexes, casting errors and stack overflows.

RoboJDE Class Library

The RoboJDE class library includes standard Java classes, robotics classes and controller specific classes that provide an excellent foundation for building your robot's intelligence. You can access full documentation of the RoboJDE class library by clicking on the blue book icon in the RoboJDE GUI.

Java Programming Benefits for Robotics Software Developers

Java programming offers many benefits that make it an excellent language for robotics projects. The following is a partial list of the benefits of programming your robot using the Java language.

- **Object Oriented Programming.** When you use object oriented programming techniques your robotics software will be easy to understand and easy to debug. You will be able to make use of the RoboJDE Class Library. You will be able to share the components you develop with other developers. If you are working as a team, you will be able to break your software into components that team members can develop and test independently.
- **Ease of Debugging.** The Java language and virtual machine are designed to prohibit or protect against common software problems such as uninitialized variables, null pointers, wild pointers, casting errors, out-of-bounds array indexes, and stack overflows. Many problems that would otherwise be difficult to debug are caught by the compiler or the virtual machine. A problem caught by the virtual machine generates an exception when it occurs and includes a stack trace pinpointing the source file and line number where the problem occurred. System crashes are virtually eliminated with Java software.
- **Built-in Operating System Features.** The Java language and virtual machine standardize many operating system functions such as multi-threading, thread synchronization and memory management.
- **Software Libraries.** The Java specifications standardize the format of compiled Java programs (class files), eliminating the need to share source or header files or require a specific compiler when sharing software. As a result, you can easily use libraries developed by others, such as the RoboJDE Class Library, and others can easily use software you develop. In addition, the Java language provides a mechanism, called Javadoc, for generating comprehensive programming interface documentation in a standard format that can be viewed using a Web browser.
- **Readily Available Supply of Related Books, Literature and other Resources.** The popularity of Java programming has produced a seemingly limitless supply of resources to help you learn to program using the Java language.
- **Portability.** Because Java software is designed to be platform independent and the Java virtual machine is well specified, Java software is very portable. Using the Java language to develop your robotics software will help ensure it is portable to other robotics controllers in the future. In addition, you may find when programming in Java it is convenient to develop and test certain

components of your software on your PC before ever executing it on your robot.

- **Learn Modern Software Development Skills.** Many languages used for robotics software development are either special purpose or nearing obsolescence. The Java language is widely used and Java tools are available for free on every major computing platform. The skills you develop using RoboJDE are modern skills that can be applied to other projects on other computing platforms.

Installation

If you do not already have the latest version of the RoboJDE™ Java-enabled robotics software development environment, visit the RidgeSoft™ web site at www.ridgesoft.com to download it.

Install the RoboJDE software by double clicking on the RoboJDESetup.exe file, then follow the installation directions provided by the setup program.

Licensing

RoboJDE software is licensed according to the license agreement included with the software. The capabilities of the software depend on the type of license you have. When you initially install the software, it operates under the RoboJDE Lite license, which is free. If you wish to upgrade to the full functionality of RoboJDE, you may purchase a license and install the associated key to unlock the full power of RoboJDE.

For more information regarding licensing options, visit the RidgeSoft web site (www.ridgesoft.com).

Getting Started

Once you have installed the RoboJDE™ software, the following sections describe how to get started using it.

Starting the RoboJDE Graphical User Interface

1. Connect your robot controller to a serial port on your PC, as described in the robot controller documentation.

Note: If your PC does not have any serial ports you will need to use a USB to serial adapter cable such as the FTDI US232B USB to serial adapter.

2. Run the RoboJDE GUI from the Windows® start menu.
3. Read the License Agreement and, if you agree to its terms, select the agree option and click Continue.
4. Use the Tools->Settings menu item to bring up the Settings dialog box.
5. Select the controller you are using.
6. Select the port to which you attached the robot controller.
7. Select the baud rate the PC should use when communicating with the robot controller.

Note: The supported baud rate settings vary depending on the robot controller. Consult the robot controller documentation for instructions on setting the baud rate to match the setting you make here. The default baud rate for the IntelliBrain™ robotics controller is 38400 baud. The Handy Board and Sumo11 boards only support 9600 baud.

8. Click OK.

Note: If another program is using the port you selected, you will receive an error message. If the error message appears, you must either shut down the other program or use a different port. It usually is not obvious which program is using the port. Programs that use a serial port, such as the Palm Pilot Hot Sync program, frequently have an icon displayed in the System Tray in the lower right corner of the monitor. You can normally close a program by clicking on its icon in the System Tray and choosing Close.

9. Turn on your robotics controller. The RoboJDE GUI will display “Load,” “Ready” or “Not Responding” in the VM state field when you turn your

robotics controller on.

10. If the RoboJDE version displayed on your robotics controller LCD is not the latest version, you must download the RoboJDE virtual machine to your robotics controller following the instructions in the next section.

Installing Your License Key

When you initially install RoboJDE, it operates in RoboJDE Lite mode. You can unlock the full functionality of RoboJDE by installing a license key. If you do not have a license key and would like to obtain one, visit the RidgeSoft web site, www.ridgesoft.com.

Use the following procedure to install your license key:

1. Select the Tools->Install License Key menu item to display the Install License Key dialog.
2. If you received your license key electronically, copy the license key and paste it into the License Key field in the dialog (click in the field and type Ctrl+V). Otherwise, type the key in manually, being careful to enter the hyphens letters and numbers correctly.

Note: License keys contain only the numbers 0-9 and letters A-F and do not include the letter "O".

3. Click OK.

Downloading the RoboJDE Virtual Machine to Your Robotics Controller

Note: IntelliBrain and IntelliBrain 2 robotics controllers have the RoboJDE virtual machine pre-loaded. Therefore, you only need to use this procedure with an IntelliBrain controller when you need to upgrade the virtual machine loaded on the controller.

1. Select Tools->Download Virtual Machine from the RoboJDE menu.
2. Follow the steps provided.

Note: If you experience problems at this step, repeat the process, being extremely careful to follow the steps exactly as prompted by RoboJDE. In particular, do not put your robot controller into download mode until RoboJDE displays the prompt indicating to do so.

3. When you have completed downloading and have restarted your robotics controller, you should see "RoboJDE" followed by the software version

number displayed on your robotics controller's LCD screen.

4. The RoboJDE software is now ready to load a Java program.

Loading and Running the Hello World Example

1. Using the RoboJDE GUI, click the Open Project button (see Figure 3).
2. Browse to the "Examples\Basic\HelloWorld" folder in the folder where you installed the RoboJDE software.
(default: C:\Program Files\RoboJDE\Examples\Basic\HelloWorld).
3. Select the file: HelloWorld.rjp
4. Click on the build and load button (see Figure 3). The program will be compiled, linked and loaded. The load program progress bar will display for several seconds.
5. Once the progress bar has disappeared, click on the run button (see Figure 3).
6. The message "Hello World" will be displayed briefly on the LCD screen and will appear in the Run window in the RoboJDE GUI.

Creating a New Project

RoboJDE uses project files to make it easy for you to switch between different projects you may be working on. You can have multiple projects in one folder, but it is usually preferable to keep your projects in separate folders.

Use the following steps to create a new project:

1. Select File->New Project menu item in the RoboJDE GUI. The Project Properties dialog will appear.
2. Click the browse button to the right of the project folder field. Browse to and select the folder you want to create the new project in.

Note: You can create a new folder by browsing to where you want to create the folder then pressing the create folder button. A folder titled "New Folder" will appear. Click on its name to change the name to a name you choose, then click on the folder icon to the left of the name and click OK.

3. Enter a name (for example, "MyBot") in the "Main class" field.
4. Click OK.

RoboJDE will automatically create the main class source file and display it in an editor window.

5. Choose where to load the program (RAM or FLASH) by selecting your desired load location on the tool bar (see Figure 3).

The Handy Board and Sumo11 do not have flash memory, so the only available choice is “RAM” when using these controllers.

Note: It is better to use RAM when developing your program to avoid wearing out the flash memory.

6. Click the load icon to build and download the newly created program to the robot controller. If the robot controller is not connected, you can build the program by clicking on the build icon instead of the download icon.
7. Click the start button to run the program. Since the program does nothing, it will exit immediately.
8. Edit the main class file to begin developing software to control your robot.

Using the RoboJDE Graphical User Interface

The RoboJDE™ graphical user interface is shown in Figure 2. The user interface consists of:

- menus
- a tool bar
- an “Edit” window
- a “Run” window

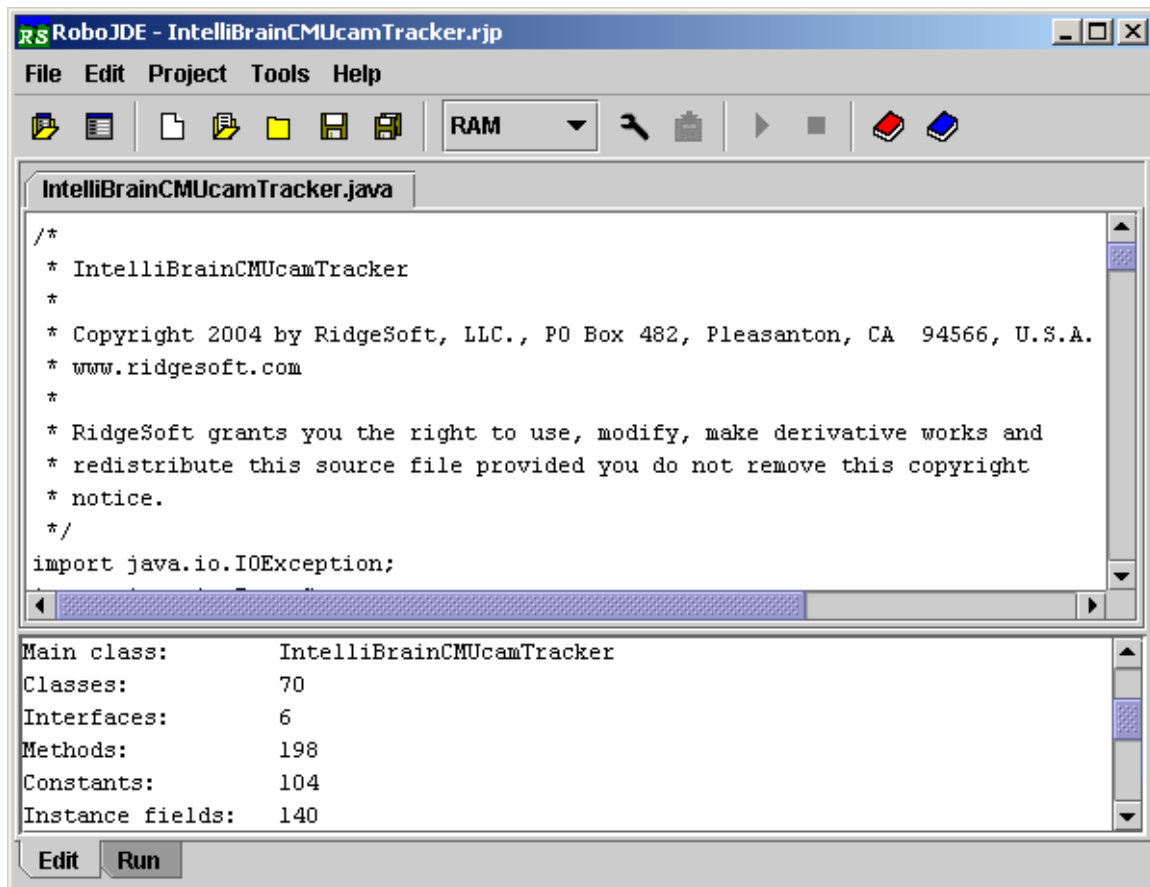


Figure 2 - RoboJDE Graphical User Interface

Using Menus

The following tables describe the items available on each menu.

Table 1 - File Menu

Menu Item	Description
New Project	Creates a new project, prompting you for the project's properties, creating a Java source file and displaying it in the Edit window. (See <i>Creating a New Project</i>).
Open Project	Allows you to browse to and open an existing project.
Save Project	Saves the current project.

Menu Item	Description
Save Project As	Allows you to save the current project under a different name or in a different location.
New Class	Creates a file for a new Java class and opens it in the edit window.
New Native File	Creates a new native method file and opens it in the edit window. This feature is supported only on the Handy Board and Sumo11. See <i>Appendix A - Programming Native Methods</i> for more information.
Open File	Opens a source file in the Edit window.
Save File	Saves the file currently displayed in the Edit window.
Save File As	Allows you to save the file currently displayed in the Edit window with a different name or in a different location.
Save All	Saves all modified files and the project file.
Page Setup	Allows you to set page printing parameters.
Print	Prints the source file currently displayed in the Edit window.
Exit	Terminates the RoboJDE user interface.

Table 2 - Edit Menu

Menu Item	Description
Cut	Cuts the selected text in the Edit window.
Copy	Copies the selected text in the Edit window.
Paste	Pastes text from the clipboard to the cursor position in the Edit window, replacing any text that is selected.
Find	Allows you to search for text in the Edit window.
Find Next	Repeats the last search proceeding from the current cursor position.
Replace	Allows you to replace text in the Edit window.
Go To Line	Allows you to move to and select a line by line number in the current file in the Edit window. This is useful for tracking down compilation errors.
Select All	Selects all of the text in the current Edit window.

Table 3 - Project Menu

Menu Item	Description
Assemble	Assembles the native method file, if there is one.
Compile	Compiles all of the files that make up the current project, starting with the main class you specify in the Project Properties (see <i>Setting Your Project's Properties</i>).

Menu Item	Description
Link	Links the current project and library classes to create a loadable image file to download to your robot. You can use a development environment other than RoboJDE to edit and compile your Java classes, then use this menu item to create an executable you can download to your robot.
Build	Assembles, compiles, and links the current project, but does not load it. If the “Compile when building and loading” box in the Project Properties dialog is unchecked, the assembly and compilation steps will be skipped.
Load	Builds the current project and downloads the program image to the robot controller. The program will be downloaded to the load location selected on the tool bar (see <i>Using the Tool Bar</i>). The controller must be attached and powered on with RoboJDE loaded, otherwise this item will be disabled. If the “Compile when building and loading” box in the Project Properties dialog is unchecked, the assembly and compilation steps will be skipped.
Run	Runs the currently loaded program – this may be different from the program you are editing. The controller must be attached and powered on with RoboJDE in the “Ready” state, otherwise this item will be disabled. This item Continues the current program if RoboJDE is in the “Breakpoint” state.
Stop	Stops the current program running on the controller. If no program is running this item will be disabled.
Properties	Displays the Project Properties dialog (see <i>Setting Your Project’s Properties</i>).

Table 4 - Tools Menu

Menu Item	Description
Download Virtual Machine	Downloads the RoboJDE virtual machine (see <i>Downloading the RoboJDE Virtual Machine to Your Robotics Controller</i>).
Install License Key	Installs a license key, enabling the full functionality of RoboJDE (see <i>Installing Your License Key</i>).

Menu Item	Description
Settings	<p>Displays the settings dialog, allowing you to specify the robot controller you are using, the communication port on your host computer to use, and the baud rate to use.</p> <p>Note: The baud rate you choose must be configured on the robot controller to match the setting you make here. See the robot controller documentation for more information. The default baud rate for the IntelliBrain robotics controller is 38400 baud. The Handy Board and Sumo11 controllers only support 9600 baud.</p>

Table 5 - Help Menu

Menu Item	Description
User Guide	Displays this document. The Adobe Acrobat reader must be installed.
API Documentation	Displays the Javadoc API documentation in a web browser (see <i>Using the RoboJDE Class Library</i>).
About	Displays information about RoboJDE.

Using the Tool Bar

Figure 3 shows the RoboJDE tool bar. With the exception of the load location selector, the icons on the tool bar have the same function as the equivalent menu item (see *Using Menus*).

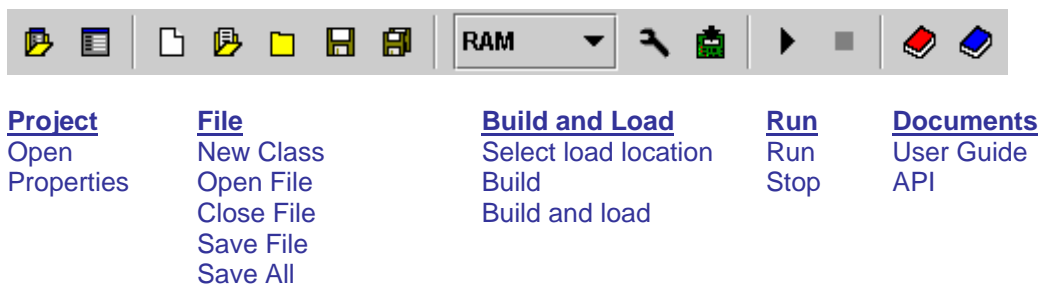


Figure 3 - RoboJDE Tool Bar

Selecting the Load Location

The load location selector at the center of the tool bar allows you to select the target program bank when downloading your program.

The IntelliBrain controller supports two load locations, RAM and FLASH. When stored in RAM, your program will be lost when the IntelliBrain is powered off. When stored in flash, the program will not be lost when the IntelliBrain is

powered off; however, flash memory wears out if it is written too many times. It is best to develop and debug your programs using RAM.

Note: If there is a program in RAM, the IntelliBrain will always choose to run it over a program stored in flash memory. If you want to run the program that is in flash memory when there is also a program stored in RAM, switch the IntelliBrain off for then back on to erase the program in RAM. This will allow you to run the program stored in flash memory.

The Handy Board and Sumo11 have a single load location, RAM. The RAM on these boards is battery backed up, so the program will not be lost when the robot is powered off, though on the Handy Board it will be lost if the battery loses its charge.

Setting Your Project's Properties

To edit your project's properties, display the Project Properties dialog, shown in Figure 4, by either clicking on the project properties tool bar icon or using the Project->Properties menu item. If you are using the Handy Board, this dialog has two tabs, "General" and "Native." You only need to be concerned with the Native tab if you are developing native methods (see *Appendix A - Programming Native Methods*).

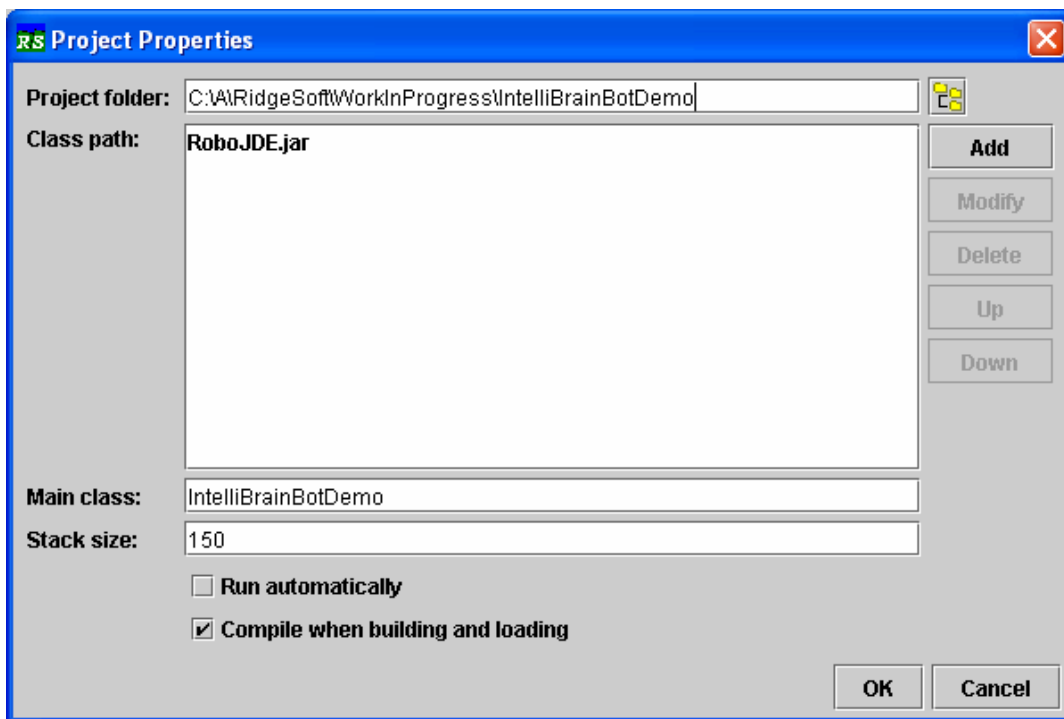


Figure 4 - Project Properties Dialog

Table 6 – Project Properties Fields

Menu Item	Description
Project folder	Browse to select the folder that will hold the project file and the source files for your project by clicking on the button to the right.
Class path	<p>Enter the list of places RoboJDE should search for classes when building the project.</p> <p>In most cases you will not need to change the default setting of this field.</p> <p>RoboJDE searches the project folder before searching this list from top to bottom. You can change the search order by using the buttons to the right to reorder the items on the list. The list may contain JAR files or the root folder for a package (The folder is typically named “classes”). The RoboJDE class library is in the RoboJDE.jar file. This file is automatically placed on the list. You must not remove RoboJDE.jar, though you may change its position in the list.</p>
Main class	Enter the name of your main class. If you are creating a new project and the source file for the main class does not exist, RoboJDE will create the file for you.
Stack size	Enter the number of stack elements that RoboJDE should allocate when starting a new thread. See <i>Minimizing Stack Size</i> for more information.
Run automatically	<p>Check this box if you want your program to start automatically when you power on your robot. This feature is not supported on the Handy Board and Sumo11 robotics controllers.</p> <p>Hold the START button down while switching power on to prevent your program from automatically running when you have selected run automatically.</p>
Compile when building and loading	If you are using another development environment, such as Eclipse, to edit and compile your program, un-check this box. This will allow you to edit and compile using that development environment then link and download your program using RoboJDE.

Using the Edit Window

The Edit window, shown in Figure 2, allows you to edit the Java and assembly language source files that make up your program. This window also displays the

output of the assembler, compiler, and linker in the lower window pane when you build your program.

Editing Your Program

You may open and make changes to multiple files at once. Each file is displayed in its own edit window under the tab named by the source file name. You may make changes to your files using the keyboard, mouse and the Edit menu. You may add a new Java class by using the File->New Class menu item. When you have completed your edits, click on the build icon, the load icon, the save file icon or the save all icon on the tool bar.

Building and Loading Your Program

When you click on the build or load icon, RoboJDE will save any source files you have changed then assemble, compile, link and load your program. If you have un-checked the “Compile When Building and Loading” field in the Project Properties, RoboJDE will skip the assembly and compilation steps. As RoboJDE builds your program, output will display in the lower pane of the edit window. If there are no errors, RoboJDE will display build statistics, and if you clicked the load icon, RoboJDE will download the program to the controller.

RoboJDE only runs the assembler if your program includes an assembly language native method file. (This is only supported for the Handy Board and Sumo11 controllers. See *Appendix A - Programming Native Methods*.) RoboJDE launches the assembler as a separate process and redirects its output to the Edit window’s output pane. The assembler generates a loadable image file with the “.S19” extension in the project folder.

RoboJDE uses Jikes, an open source compiler, to compile your Java files and generate class files. RoboJDE launches the Jikes compiler as a separate process and redirects its output to the Edit window’s output pane. The “.class” files generated by the Jikes compiler are placed in the project folder, or in the “classes” folder below the project folder, if you are using packages (see the *Using Packages* section).

RoboJDE links the program using its built-in optimizing linker. The linker analyzes your program, starting at the main method, to determine which classes, methods and fields are needed to execute your program. The linker then builds an executable image and creates a map file. The linker minimizes the size of the load image by automatically eliminating classes, methods and fields that are not accessed by your program. This allows the class library and the classes you develop to provide a rich functionality while not inflating the size of your programs. The linker outputs the loadable image to a “.hex” file and the map to a “.map” file in the project folder. (See the *Understanding Errors, Exceptions and Stack Traces* section for more information on the map file.)

RoboJDE communicates with the virtual machine – or the bootstrap loader, if you are loading to flash memory – to download the image file to the controller’s memory.

Once your program has been downloaded, you may click on the run icon to run your program. RoboJDE will switch the user interface to the Run window.

Debugging Build Errors

If the assembler or compiler report errors, note the source file and line number of the errors. Click on the tab for the file with the error(s), or open the file, then type “Ctrl-G” or use Edit->Go To to view the line where the error was reported. Once you have corrected the error click on the build or download icon to rebuild and/or download your program.

The RoboJDE linker and loader will report errors if they cannot find classes or methods your programs references. This will happen if you have the wrong controller type configured in the Tools->Settings dialog or if the class path or main class in the Project Properties is incorrect.

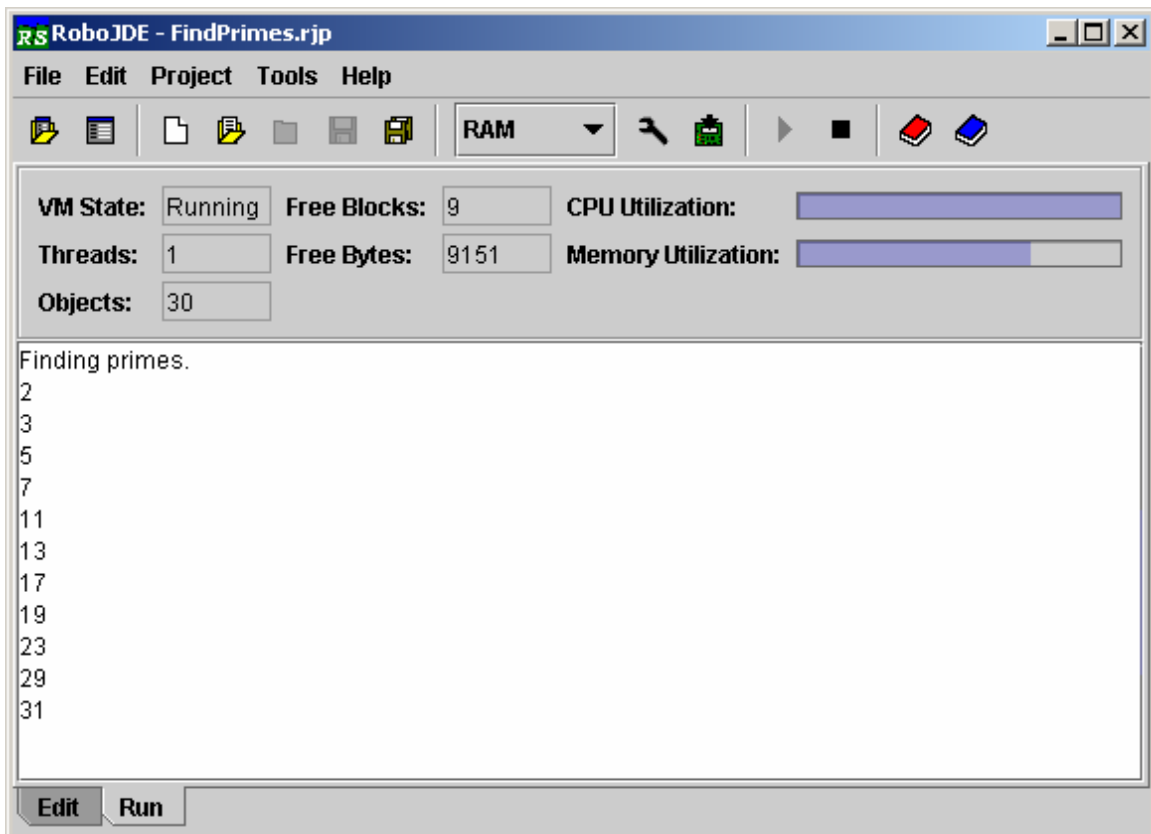


Figure 5 - RoboJDE Run Window

Using the Run Window

The RoboJDE Run Window is shown in Figure 5. The upper portion of the Run Window shows the status of the virtual machine. The lower portion shows output from the program running on the robot. By default, your program output to `System.out` and `System.err` will display in this window if you have the serial cable attached. The run window also decodes stack traces to convert addresses to a source file, method and line number.

Table 7 describes the Run Window status fields.

Table 7 – Virtual Machine Status Fields

Field	Description
VM State	<p>Displays the current state of the virtual machine.</p> <ul style="list-style-type: none"> ▪ Load – the virtual machine is waiting for an program to be loaded ▪ Ready – a program is loaded and ready to run ▪ Running – a program is currently running ▪ Breakpoint – the virtual machine encountered a break point. You can add breakpoints to your code by calling the method <code>VM.breakpoint()</code>. ▪ Invalid Port – the communication port has not been set. Use the Tools->Settings to set the correct port. ▪ Not Responding – the virtual machine is not responding to status requests from the user interface. This could be due to the controller being off, the serial cable not being connected, or the virtual machine not being loaded (see <i>Downloading the RoboJDE Virtual Machine to Your Robotics Controller</i>).
Threads	Displays the number of currently running threads.
Objects	Displays the current number of Java objects in use.
Free Blocks	Displays the number of blocks of memory. Memory may become fragmented into many free blocks if your program interleaves creation of long-term objects with short-term objects. See <i>Avoiding Memory Fragmentation</i> for more information.
Free Bytes	Displays the number of bytes of memory which aren't currently allocated.
CPU Utilization	Displays a bar indicating roughly what portion of time the controller's CPU is not idle. If your program executes a loop without sleeping or calling a method that will cause it to block, the CPU will be 100% utilized. If your program spends most of its time sleeping or waiting for some input or output to happen, the CPU utilization will be close to 0%.
Memory Utilization	Displays a bar indicating how much of the memory available to programs is currently being used.

Programming Your Robot

In order to program your robot you will need to be familiar with:

- Java™ programming
- the RoboJDE™ class library
- the robot controller, sensors, and effectors you will be using

Using the RoboJDE Class Library

The RoboJDE class library provides the foundation classes you will use to program your robot. The class library provides the Application Programming Interface (API) to the robot controller hardware and various sensors and effectors. In addition, the class library includes higher level classes that provide a foundation to help you build an intelligent robot. The class library also provides a subset of classes and methods found in the `java.lang`, `java.util`, `java.io`, and `javax.comm` packages defined by the Java specifications.

Detailed documentation of the class library is included with RoboJDE in standard Javadoc format. You can view this documentation in the web browser by clicking on the blue book icon in the RoboJDE user interface (see Figure 3).

Alternatively, you can view the documentation by clicking on the “index.html” file in the “apidoc” folder or by browsing to it using a web browser.

Robot Controller API Quick References

The “docs” folder contains a quick reference graphic summarizing the API to each supported robot controller.

Table 8 - Controller API Quick References

Controller	File Name
IntelliBrain 2	IntelliBran2API.pdf
IntelliBrain	IntelliBrainAPI.pdf
Handy Board	HandyBoardAPI.pdf
Sumo11	Sumo11API.pdf

Learning from Examples

RoboJDE provides more than sixty example programs which demonstrate how to interface a Java program to various robot controllers, sensors and effectors. Numerous examples of programming the IntelliBrain-Bot are also included, as well as examples of several other complete robot programs.

The example programs may be found in the Examples folder (default: `C:\Program Files\RoboJDE\Examples`).

Note: Many of the examples are too large to run with RoboJDE Lite and require a full RoboJDE license. A `LicenseError` exception will occur if the program is too

large to run with RoboJDE Lite. See the section titled *Installing Your License Key* for more information.

Understanding Errors, Exceptions and Stack Traces

All run-time errors except internal virtual machine errors are reported to your Java program as exceptions. Understanding exceptions and stack traces will improve your ability to quickly identify and resolve problems in your Java programs. If you are not familiar with debugging using stack traces, it will be well worth your time to learn how to use stack trace output to find and resolve problems in your programs.

RoboJDE catches all uncaught exceptions in your main thread and prints a stack trace should an exception occur. By default, the output of the `printStackTrace()` method goes to both the Run window in RoboJDE and to the LCD screen. This method prints the name of the exception class, the exception message and a stack trace. You can view specific information on an exception class by viewing the API documentation for the class.

Note: Under certain circumstances, such as `OutOfMemoryError` and `StackOverflowError` exceptions, the error condition may prevent RoboJDE from printing a stack trace, resulting in the program exiting without an indication of the cause.

The following example illustrates how you can use a stack trace to find the cause of an `ArithmeticException` when the example executes. Line numbers have been added to the left for the purpose of this discussion.

```
1 public class DivideByZero {
2     public static void main(String args[]) {
3         System.out.println("Result: " + divide(5, 0));
4     }
5
6     private static int divide(int dividend, int divisor) {
7         return dividend / divisor;
8     }
9 }
```

Running this example results in the following output in the RoboJDE Run window:

```
ArithmeticException
  at DivideByZero.divide(DivideByZero.java:7)
  at DivideByZero.main(DivideByZero.java:3)
```

The first line of output indicates an `ArithmeticException` occurred. The API documentation indicates this exception is thrown by the virtual machine when the

program attempts to divide by zero. The first line of the stack trace indicates the exception occurred in the class `DivideByZero` and in method `divide()` at line 7 in the source file `DivideByZero.java`. Examining line 7 in the source file, shown above, reveals the program was attempting an integer division and the `divisor` argument to the `divide()` method must have been zero. The `divide` method is correct, but the method that called `divide` must have passed 0 as the `divisor`. The second line of the stack trace shows that the `main()` method called `divide()` at line 3 in `DivideByZero.java`. Examining line 3 shows that `divide()` is being called with a divisor of 0, which is the cause of the exception.

Note: You can view a particular line in a source file by opening the file in the Edit window, then using `Ctrl-G` or `Edit->Go to Line` to go to that line.

If you are using multiple threads, it is good practice to include a try-catch block around all of the code in your run methods, as shown in the following example. This will ensure a stack trace will be printed before the thread exits if an uncaught exception occurs; otherwise, the thread may exit without any indication of what caused it to terminate.

```
// A Thread's run method
public void run() {
    try {
        // your code
        :
        :
    }
    catch (Throwable t) {
        t.printStackTrace();
    }
}
```

The stack trace printout to the LCD screen lists addresses rather than the class, method, source file and line number information printed out in the RoboJDE Run window. This is because this detailed information is not available on the robot controller. You can refer to the map file `RoboJDE` generated in the project folder when you built your program to determine which methods were executing at the time of the exception.

Internal errors should rarely or never occur. They are the result of errors in the virtual machine. You can't fix internal errors by changing your program, though you may be able to avoid them by changing your program. If your program encounters an internal error, observe the instruction pointer (ip) / program counter value displayed by the virtual machine when the error occurred. Look this value up in the map file (see below) to determine what method your program was executing when the error occurred. This may provide some insight into what triggered the internal error, and perhaps it will allow you to change your code to avoid it. Also, refer to the RidgeSoft web site for support information or send email to support@ridgesoft.com to report the error.

Map Files

A map file is generated each time you build your program. The map file is named by the name of your program's main class followed by "\$.map" (for example, "HelloWorld\$.map") and placed in the project folder. The map file provides statistics generated by the linker followed by the list of methods included in the linked image. Each method's name in the listing is preceded by the hexadecimal starting and ending addresses of the executable portion of the method. The addresses are non-contiguous because they only refer to the executable code and not the non-executable class, method and field data contained in the executable image.

Using Packages

If your robotics projects become large or you are developing your own class library, you may find it convenient to organize your project(s) into packages. If you are working with small projects, it is best to avoid the complexity of packages.

If storing all of your files for a particular project in single folder isn't practical, RoboJDE supports organizing your project in a package hierarchy, as shown in Figure 6. The root folder is the project folder you specify in the Project Properties dialog. The "src" folder is the root of the source code files for your package and the "classes" folder is the root of the compiled class files for your project. These folders and the subordinate folders are created automatically under the project folder by RoboJDE when you include a package name when creating a new class.

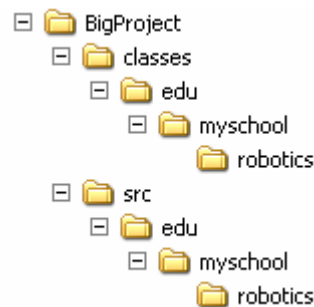


Figure 6 - Example Package Structure

All you need to do to create a project that uses a package structure is provide package names when specifying the names of your classes. For example, if you were to enter the main class name "edu.myschool.robotics.AClass" when creating a project the package structure shown in Figure 6 would be created. The steps you need to follow to build, load and run your program don't change whether you choose to organize your project using a package hierarchy or not.

Optimizing Memory Usage

As you create more sophisticated intelligence for your robot you may find it is a challenge to fit your program into the small amount of memory on the robot controller. This section provides many tips that will help you make the most of the available memory on your robot controller.

Memory optimization is much less important if you are using the IntelliBrain controller. As shown in Table 9, the IntelliBrain has more than ten times the memory available for your program than does the Handy Board or Sumo11.

Table 9 - Controller Memory Comparison

Controller	Memory
IntelliBrain	<ul style="list-style-type: none"> ▪ 128K flash program memory (60K available for your Java program) ▪ 132K RAM (128K available for program data)
Handy Board / Sumo11	<ul style="list-style-type: none"> ▪ 32.25K RAM (~16K available for your Java program and its data)

Indications of Low Memory

The statistics on the RoboJDE Run window indicate how much memory your program is using, how much memory is free and how fragmented the free memory is. These statistics provide the primary indication of low memory.

If your program attempts to create an object or to start a thread and there is not enough memory to satisfy the request, the virtual machine will throw an `OutOfMemoryError`. This exception can be caught by your program and an error message displayed provided there is still enough memory to execute your exception handling code. If there is not enough memory to execute the exception handler, the thread may exit silently. You can determine if this has happened by observing if the Threads statistic in the Run window indicates if all of your program's threads are running. If you only have one thread, the program will exit and the virtual machine state will revert from "Running" to "Ready."

Minimizing Stack Size

Minimizing the number and size of stacks is normally the easiest way to optimize your program's memory use. The default stack size for your program is set in the Project Properties dialog. By default, RoboJDE sets the stack size to 200 4-byte elements (800 bytes). The fewer nested calls your program makes, the smaller the stacks your program will need. The minimum stack size needed to execute the startup code before your main method is called is about 100 elements.

Each thread you create requires its own stack. Reducing the number of threads you use will result in a substantial savings of memory due to fewer stacks. In addition, the threads you create normally do not require as much stack space as the main thread because they don't need to execute the startup code as the main thread does. By default, the threads you create have the stack size specified in

the Project Properties dialog. However, your program can adjust a thread's stack size prior to starting a thread by using the following method:

```
VM.setStackSize(Thread thread, int stackSize);
```

Often the stack size can be trimmed down to 60 elements or less. Finding the right size is a trial and error process. If a thread's stack is too small, a `StackOverflowError` will be thrown, but if there is not enough stack space to execute your catch block code, the thread may exit silently. By monitoring the Threads statistic in the Run window you can determine if all of your program's threads are running.

Finally, if your main thread requires a lot of stack space to get the program up and running, it may be advantageous to have another thread with a smaller stack take over control and have the main thread exit.

Minimizing Java Executable Size

The executable program is normally the biggest consumer of memory. RoboJDE does a lot to minimize its size, such as not including classes, methods and member variables in the executable program that will never be used. However, reducing the number of classes, methods and member variables your program references will reduce the amount of memory the program uses. Often less abstraction and a more procedural approach will reduce the program size.

For the Handy Board and Sumo11, the `HandyBoard` class often provides two means to access the controller's features, a static method that carries out an operation directly, or a method that returns an object supporting a generalized interface. In cases where there isn't a lot of benefit to using the abstraction of the generalized object, calling the static method directly may save on executable size.

RoboJDE reports the size of the executable in the build output pane of the Edit window.

Table 10 - Data Type Sizes

Type	Range	Size in		
		Field	Array	Local/Stack
boolean	true, false	1 byte	1 bit	4 bytes
byte	-128 - 127	1 byte	1 byte	4 bytes
short	-32768 - 32767	2 bytes	2 bytes	4 bytes
char	0 - 65768	2 bytes	2 bytes	4 bytes
reference	n/a	2 bytes	2 bytes	4 bytes
int	-(2 ³¹) - 2 ³¹ -1	4 bytes	4 bytes	4 bytes
float	~±10 ³⁸	4 bytes	4 bytes	4 bytes
long	-(2 ⁶³) - 2 ⁶³ -1	8 bytes	8 bytes	8 bytes
double	same as float	8 bytes	8 bytes	8 bytes

Minimizing Program Data Size

Program data typically does not consume a large amount of space unless you are using arrays. Never the less, using the smallest suitable data type for your member variables and array elements will save space. Table 10 summarizes the space consumed by each of the Java primitive data types.

Note: All local variables consume one stack element (4 bytes), except for long and double, which consume two. Therefore, using byte, short and boolean for local variables will not necessarily save memory space.

Freeing Objects and Arrays that Are Not Needed

When objects are no longer referenced by the program, RoboJDE automatically recovers their memory. Being careful to avoid lingering references to objects that are no-longer needed will allow RoboJDE to more quickly recover memory. When an object is no-longer need, setting references to it to `null` will allow RoboJDE to recover the memory it uses. If you have data structures that make circular references, be sure to break the circular references by setting them to `null` so RoboJDE will recognize the objects that are free to be garbage collected.

Avoiding Memory Fragmentation

Generally fragmentation isn't a big issue because robotics programs tend to allocate most of their long term control objects during initialization. However, interleaving the creation of long term objects with creation of short term objects may cause memory to become fragmented. Fragmentation problems are usually indicated by an ever increasing "Free Blocks" count in the Run window – each free block is a "fragment" of free memory. If your program is collecting information, such as objects holding mapping data, it may be advantageous to pre-allocate objects at startup to avoid memory fragmentation.

Appendix A - Programming Native Methods

Overview of Native Methods

The RoboJDE™ software allows you to implement low-level functions, including interrupt service routines in assembly language on the Handy Board and Sumo11 controllers. The native method interface allows your Java program to call native methods implemented in assembly language. The NativeMethod example provides an example of this.

The native methods you implement normally reside in a single assembly language file. You can create a native method file in your project using the File->New Native File menu item in the RoboJDE user interface. This will create a template you can modify to implement your native methods.

When you build and load your program, RoboJDE will automatically assemble your native method file and download it to the robotics controller. A header in the assembled native file allows the RoboJDE virtual machine to register the native methods.

Use the following procedure to implement a native method:

1. Select the File->New Native File menu item to create an assembly language native method file. Note: The file name must be eight characters or less, excluding the “.asm” extension.
2. Declare the new method in a Java class (see below).
3. Add the native method signature to the project properties (see below).
4. Implement the native method in the assembly language file (see below).
5. Load and debug the native method.

These steps are described in more detail in the following sections.

Declaring Your Native Method

You must declare your native methods in your Java classes similar to the way you would declare an abstract method or an interface method. You declare the method but do not include a method body; instead, you create the method body in assembly language in the native method file. This allows you to call the method elsewhere in your program just like you would call any other method. It is best to declare your native methods `static` and pass variables as arguments rather than try to access an object's member variables in your assembly language file.

In the NativeMethod example, the statement:

```
private static native int increment(int value);
```

defines a native method. The implementation of this method is in the Native.asm file.

Adding the Method Signature to the Project Properties

You must add the native method signature to your project's properties so the RoboJDE linker can establish the linkage to the assembly language implementation. Use the following procedure to do this.

1. Select the Project->Properties menu item to open the project properties.
2. Click on the Native tab.
3. Click on the Add button.
4. Enter the prototype of the method you are adding, then click OK. The prototype looks similar to the declaration in the Java source file; however, the method name must be preceded by the class name. In the NativeMethod example the prototype is:

```
int NativeMethod.increment(int value)
```

5. If this is not the first method you have added, use the Up and Down buttons to arrange the method prototypes to be in the exact order the methods are ordered in the vector table in the native method file.

Implementing Native Methods

In order to implement native methods you will need to be familiar with assembly language programming. Many good resources are available if you are not already familiar with programming the Motorola 68HC11 in assembly language, such as the *68HC11 Technical Reference*¹ and the books *Robotic Explorations*² and *Mobile Robots*³.

The RoboJDE native method interface calls the initialization entry point in your native method file while initializing your program. This routine may be used to initialize variables, initialize hardware and to configure interrupt service routines.

The native method vector table allows you to define multiple native methods. However, the number of native methods you will be able to use may be limited by the RoboJDE license you have. The RoboJDE Lite license allows you to have

¹ Motorola, *M68HC11 Technical Reference*, www.motorola.com.

² Fred Martin, *Robotic Explorations*, Prentice Hall, 2001.

³ Jones, Flynn, Seiger, *Mobile Robots*, A K Peters, 1999.

one native method. If your license allows, you may add native methods by adding new entries to the vector table. The order of the methods in the vector table must be identical to the order of the methods configured in the project properties.

When your native method begins executing, the X register points to the method’s arguments on the Java stack. (Note: the Java stack is separate from the native stack, which is pointed to by the SP register.) Each location on the Java stack is 32 bits (4 bytes). All data types consume one stack location, except long and double, which consume two stack locations. The right most argument to the method will be at offset zero (0,X). The preceding argument will be at offset 4 (4,X), and so on.

If your method has a return value, the value must be returned in Y:D. If you are returning a data type smaller than 32 bits you must still put a 32 bit integer in the Y:D registers because all integers are 32 bits on the Java stack. The Y register contains the most significant bits and the D register contains the least significant bits.

Table 11 - Built-in Debugger Displays

Display Format	Description
PPPP SSSS CC DDDD XXXX YYYY	Register display. PPPP – value in the program counter register (the address of the next instruction) SSSS – value in the stack pointer register CC – value in the condition code register DDDD – value in the D register, first two digits are the A register, last two digits are the B register XXXX – value in the X register YYYY – value in the Y register
R AAAA MMMMMMMM MMMMMMMMMMMMMMMM	Memory display. There are 4 displays that use this format. R – register identifier – ‘S’ (stack pointer), ‘X’ (X register), ‘Y’ (Y register) and ‘W’ (watch point). AAAA – value (address) in the register MM... - values in memory pointed to by the register
Program output	The data on the LCD screen prior to hitting the breakpoint.

Debugging Native Methods

RoboJDE has a built-in debugger that you may find useful when debugging your native methods. The debugger allows you to insert breakpoints into your assembly code. When a breakpoint is hit, the debugger provides several displays that allow you to examine the contents of registers and memory. The RoboJDE GUI will display “Breakpoint” as the VM State.

The debugger provides the displays listed in Table 11. When the first breakpoint is hit, the register display will be displayed on the LCD screen. You may scroll through the displays by pressing the STOP button on the robot controller. You may resume the program by pressing the START button. The next time a breakpoint is hit, the display that was showing when you resumed the program will display.

To insert a breakpoint, add the `swi` (software interrupt) instruction in your assembly language code where you want the breakpoint to occur, then rebuild, download and run your program.

Accessing I/O Registers and Control Ports

If you desire to access the motor and expansion board control ports on the Handy Board or Sumo11, you must be careful to avoid conflicts with the virtual machine's use of these ports. In general, it is best to avoid using these ports in native methods. Instead, use the methods provided in the class library to use these features.

If you need to write to the Handy Board motor port at address \$7000, you must not use the motor control functions supplied by the class library.

If you need to manipulate bits in the Handy Board expansion control register at address \$4000, or the Sumo11 expansion control register at \$7000, you must save the flag register, disable interrupts, load a copy of the value in the register, change the appropriate bits, update the copy and write the new value to the register. A copy of the data in the expansion control register is stored at memory address \$0000. The following code illustrates the process:

```
tpa          ; transfer flags to A register
sei          ; disable interrupts
ldab $0000  ; read copy of value in reg.
orab $80    ; set digital out 8
stab $0000  ; save copy
stab $7000  ; write register
tap         ; restore interrupt state
```

If you need to manipulate the digital output port on the Handy Board expansion board at address \$5000, you must use the procedure just described. In this case the copy of the data in the digital output register is stored at memory address \$0001.

If you need to access an I/O register in the 68HC11 that is shared by the RoboJDE virtual machine – for example, the interrupt mask register – you must use the procedure above, except the current value may be read directly from the register.