RidgeSoft™

RoboJDE™

Java™-enabled Robotics Software Development Environment

User Guide

Version 2.0

www.ridgesoft.com

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

Document Version 2.0

Table of Contents

Introduction 1
- RoboJDE Graphical User Interface 1
- RidgeSoft Command Line Loader 1
- RoboJDE Virtual Machine 1
- RoboJDE Class Library 2
- Java Programming Benefits for Robotics Software Developers 2

Installation 4
- Licensing 4

Getting Started 5
- Starting the RoboJDE Graphical User Interface 5
- Downloading the RoboJDE Virtual Machine to Your Robotics Controller 6
- Loading and Running the Hello World Example 6
- Creating a New Project 7

Using the RoboJDE Graphical User Interface ... 8
- Using Menus 8
- Using the Tool Bar 11
  - Selecting the Load Location 11
- Setting Your Project’s Properties 12
- Using the Edit Window 13
  - Editing Your Program 13
  - Building and Loading Your Program 13
  - Debugging Build Errors 14
- Using the Run Window 15

Using the RidgeSoft Command Line Loader ... 17
- Command Line Loader Syntax 17
- Command Line Loader Options 17
- Loading and Running Programs from within Eclipse 19
  - Configuring Your Eclipse Project 20
  - Configuring the Loader as an External Tool 20
  - Loading and Running Your Program 22

Programming Your Robot 23
- Using the RoboJDE Class Library 23
- Robot Controller API Quick References 23
- Learning from Examples 23
- Understanding Errors, Exceptions and Stack Traces 23
- Using Packages 26
- Optimizing Memory Usage 27
  - Indications of Low Memory 27

Minimizing Stack Size 27
Minimizing Java Executable Size.. 28
Minimizing Program Data Size... 29
Freeing Objects and Arrays that Are Not Needed.. 29
Avoiding Memory Fragmentation 29

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
- RidgeSoft Command Line Loader
- RoboJDE Class Library
- RoboJDE Virtual Machine

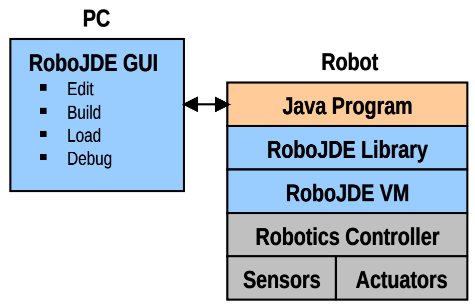


Figure 1 - RoboJDE Components

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 computer running Microsoft Windows®, Mac® OS X or Linux and communicates with the robot controller using an RS232 serial connection.

RidgeSoft Command Line Loader

The RidgeSoft Command Line Loader provides a command line tool to download a Java program from a computer running Microsoft Windows, Mac OS X or Linux to a robotics controller. The command line loader facilitates downloading programs from third-party Integrated Development Environments (IDE) such as Eclipse, from a command or terminal window or via a command script. This allows you to develop programs using your favorite Java IDE rather than the RoboJDE GUI.

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 opening the file RoboJDE\apidoc\index.html in your web browser or by clicking on the "RoboJDE API Documentation" item in the RoboJDE program group on the Windows start menu.

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 un-initialized 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 the installation program for your type of computer.

Install the RoboJDE software by running the appropriate installation program for your computer and then following the directions provided by the installation program.

Licensing

RoboJDE software is licensed according to the license agreement included with the software.

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 the RoboJDE GUI. Later sections describe how to use the RidgeSoft Command Line Loader.

Starting the RoboJDE Graphical User Interface

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

   Note: If your computer does not have a built-in serial port, you will need to use a USB to serial adapter cable or a USB to serial Bluetooth adapter. Both are available from www.ridgesoft.com.

2. Run the RoboJDE GUI from the start menu or command line of your computer.

3. Use the Tools->Settings menu item to bring up the Settings dialog box.

4. If the compiler field is empty, browse to and select the Java compiler you would like the RoboJDE GUI to use.

   Note: If you are using a Linux system which does not have a Java compiler or Software Development Kit (SDK) installed, you will first need to install a Java SDK.

5. Select the controller type you are using.

6. Select the port to which you attached the robot controller.

7. Select the baud rate your computer should use when communicating with the robot controller. For the IntelliBrain 2 robotics controller, choose 115200 baud. For the Handy Board choose 9600 baud.

   Note: For the IntelliBrain 2 robotics controller, only use a lower baud rate if you find communication is unreliable using 115200 baud. If RoboJDE is unable to communicate to the robotics controller, make sure that you have selected the correct port. Also make sure the IntelliBrain 2 robotics controller is configured to use 115200 baud (see the *IntelliBrain 2 User Guide* for more information).

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.

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.

Downloading the RoboJDE Virtual Machine to Your Robotics Controller

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

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.

3. Select the file: HelloWorld.rjp

4. Click 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 the run button (see Figure 3).

6. The message “Hello World!” will be displayed briefly on the LCD screen and it will also appear in the Run window in the RoboJDE GUI.

Creating a New Project

The RoboJDE GUI 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 because it is faster to download to and it avoids 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.

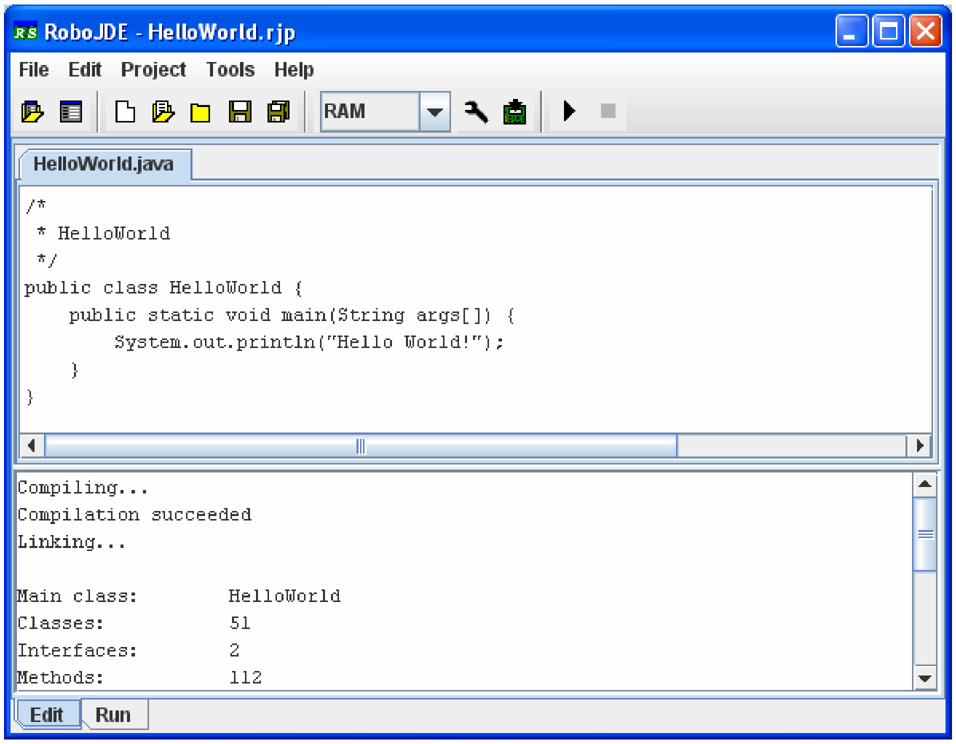
7. Click the start button to run the program. Since the program does nothing, it will exit immediately.

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 (GUI) is shown in Figure 2. The user interface consists of:

- menus
- a tool bar
- an "Edit" window
- a "Run" window



```
/*
 * HelloWorld
 */
public class HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello World!");
    }
}
```

```
Compiling...
Compilation succeeded
Linking...

Main class:          HelloWorld
Classes:             51
Interfaces:          2
Methods:             112
```

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 *Creating a New Project*).
Open Project	Allows you to browse to and open an existing project.
Save Project	Saves the current project.
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.

Menu Item	Description
Open File	Opens a source file in the Edit window.
Close File	Closes the file currently displayed 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
Compile	Compiles all of the files that make up the current project, starting with the main class you specify in the Project Properties (see *Setting Your Project's Properties*).
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	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 compilation step will be skipped.

Menu Item	Description
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 *Using the Tool Bar*). 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 compilation step 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 *Setting Your Project's Properties*).

Table 4 - Tools Menu

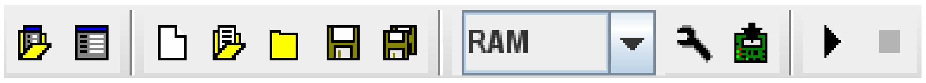
Menu Item	Description
Download Virtual Machine	Downloads the RoboJDE virtual machine (see *Downloading the RoboJDE Virtual Machine to Your Robotics Controller*).
Settings	Displays the settings dialog, allowing you to specify the Java compiler to use, the robot controller you are using, the communication port on your host computer to use, and the baud rate to use. 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 2 robotics controller is 115200 baud. The Handy Board and Sumo11 controllers only support 9600 baud.

Table 5 - Help Menu

Menu Item	Description
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*).



Project	**File**	**Build and Load**	**Run**
Open	New Class	Select load location	Run
Properties	Open File	Build	Stop
	Close File	Build and load	
	Save File		
	Save All		

Figure 3 - RoboJDE Tool Bar

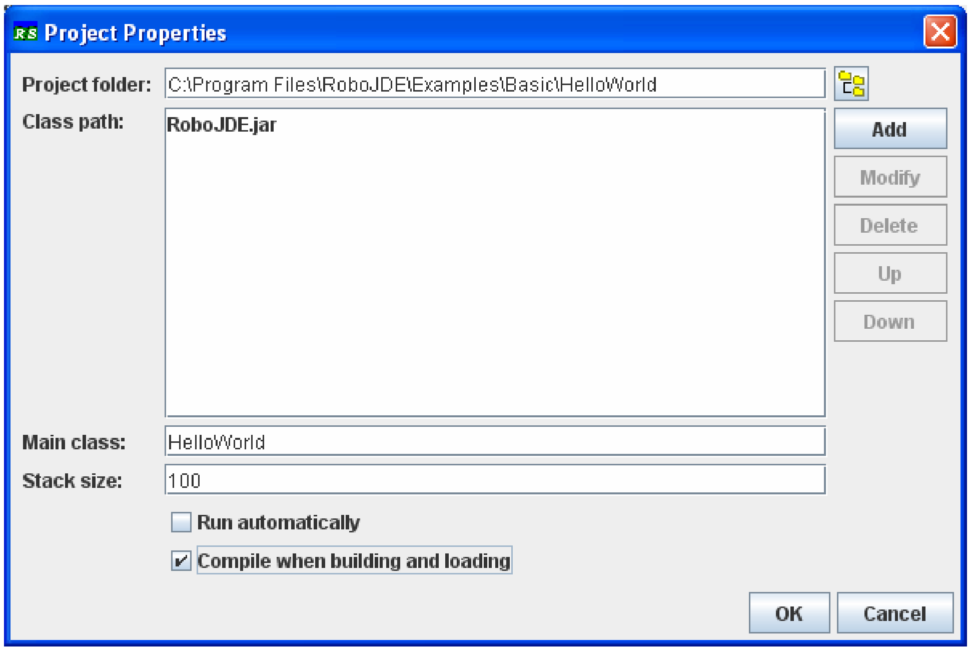
Selecting the Load Location

The load location selector is the control at the left of the build and load grouping of controls on the tool bar (see “RAM” in Figure 3). It allows you to select the target program bank when downloading your program.

The IntelliBrain 2 controller supports two load locations, RAM and FLASH. When stored in RAM, your program will be lost when you turn the IntelliBrain 2 controller off. When stored in flash memory, your program will not be lost when you turn the IntelliBrain 2 controller off. However, flash memory takes longer to load and 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 RoboJDE virtual machine will always choose to run it over a program stored in flash memory. If you want to run a program that is stored in flash memory when there is also a program stored in RAM, turn the IntelliBrain 2 controller off and back on to erase the program stored 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 erased when the robot is powered off. However, the program and virtual machine will need to be reloaded if the battery loses its charge.



Project folder: C:\Program Files\RoboJDE\Examples\Basic\HelloWorld

Class path: RoboJDE.jar

Main class: HelloWorld

Stack size: 100

[ ] Run automatically

[✓] Compile when building and loading

Figure 4 - Project Properties Dialog

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.

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	Enter the list of places RoboJDE should search for classes when building the project. In most cases you will not need to change the default setting of this field. 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 (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, although you may change its position in the list.

Menu Item	Description
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 *Minimizing Stack Size* for more information.
Run automatically	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. Note: Holding the START button down while switching power on disables this option, allowing you to prevent your program from starting automatically.
Compile when building and loading	If you are using another development environment to edit and compile your program and using the RoboJDE GUI to load and run your programs, un-check this box. Note: Another alternative when using Eclipse or another integrated development environment is to use the RoboJDE Command Line Loader to load your programs. See *Using the RidgeSoft Command Line Loader*.

Using the Edit Window

The Edit window, shown in Figure 2, allows you to edit the Java source files that make up your program. This window also displays the output of the 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 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 compilation step. 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 uses the compiler specified in the Tools->Settings dialog to compile your Java files and generate class files. RoboJDE launches the compiler as a separate process and redirects its output to the Edit window's output pane. The ".class" files generated by the 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). On Windows systems, RoboJDE defaults the compiler setting to the Jikes compiler, which is installed with RoboJDE. On Mac and Linux systems, RoboJDE defaults the compiler setting to the default Java compiler, if one is installed.

RoboJDE links the program using its built-in optimizing linker. The linker analyzes your program 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 need by your program. This allows the class library and the classes you develop to provide 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.)

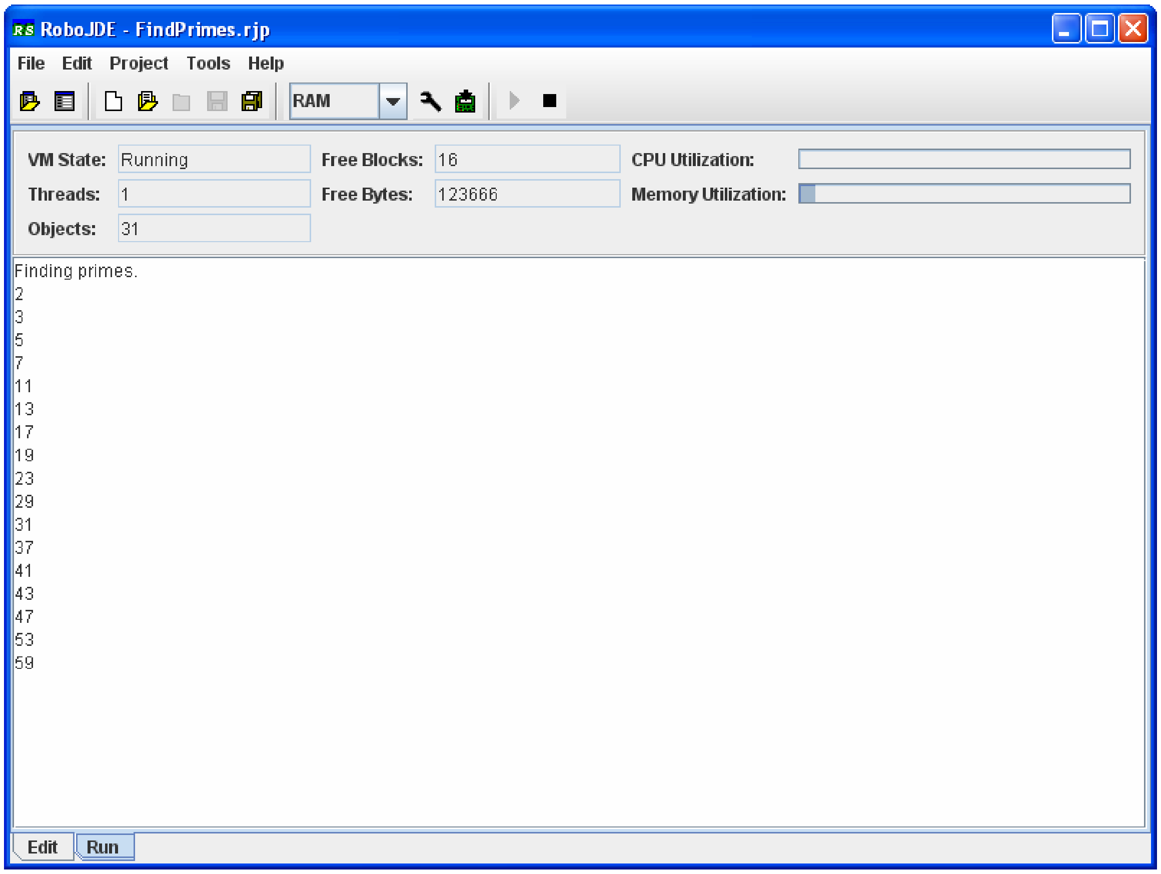
The RoboJDE GUI communicates with the RoboJDE virtual machine to download the image file to the controller's memory.

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 compiler reports errors, note the source file and line number of each error. Click on the tab for the file with the error(s), or open the file, then type "Ctrl-G" or use Edit->Go to Line 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 program references. This may 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.



VM State: Running
Threads: 1
Objects: 31
Free Blocks: 16
Free Bytes: 123666
CPU Utilization:
Memory Utilization:

```
Finding primes.
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
```

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 when 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	Displays the current state of the virtual machine. ▪ 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 `VM.breakpoint()`. ▪ 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 *Downloading the RoboJDE Virtual Machine to Your Robotics Controller*).
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 *Avoiding Memory Fragmentation* 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.

Using the RidgeSoft Command Line Loader

The RidgeSoft™ Command Line Loader allows you to load and run your programs from a command prompt, an Integrated Development Environment (IDE), such as Eclipse, a shell script or a batch file.

Command Line Loader Syntax

The command line loader uses the following syntax:

```
rsload [options] [class]
```

The “class” argument is used to specify the name of your program’s main class. The available option arguments are listed in Table 8.

Example

```
C:\My Documents>rsload -port COM2 HelloWorld
RidgeSoft Loader 2.0.0
(c) Copyright 2009 RidgeSoft, LLC. All rights reserved.
Loading HelloWorld..............
Download complete
```

Command Line Loader Options

Table 8 - Command Line Loader Options

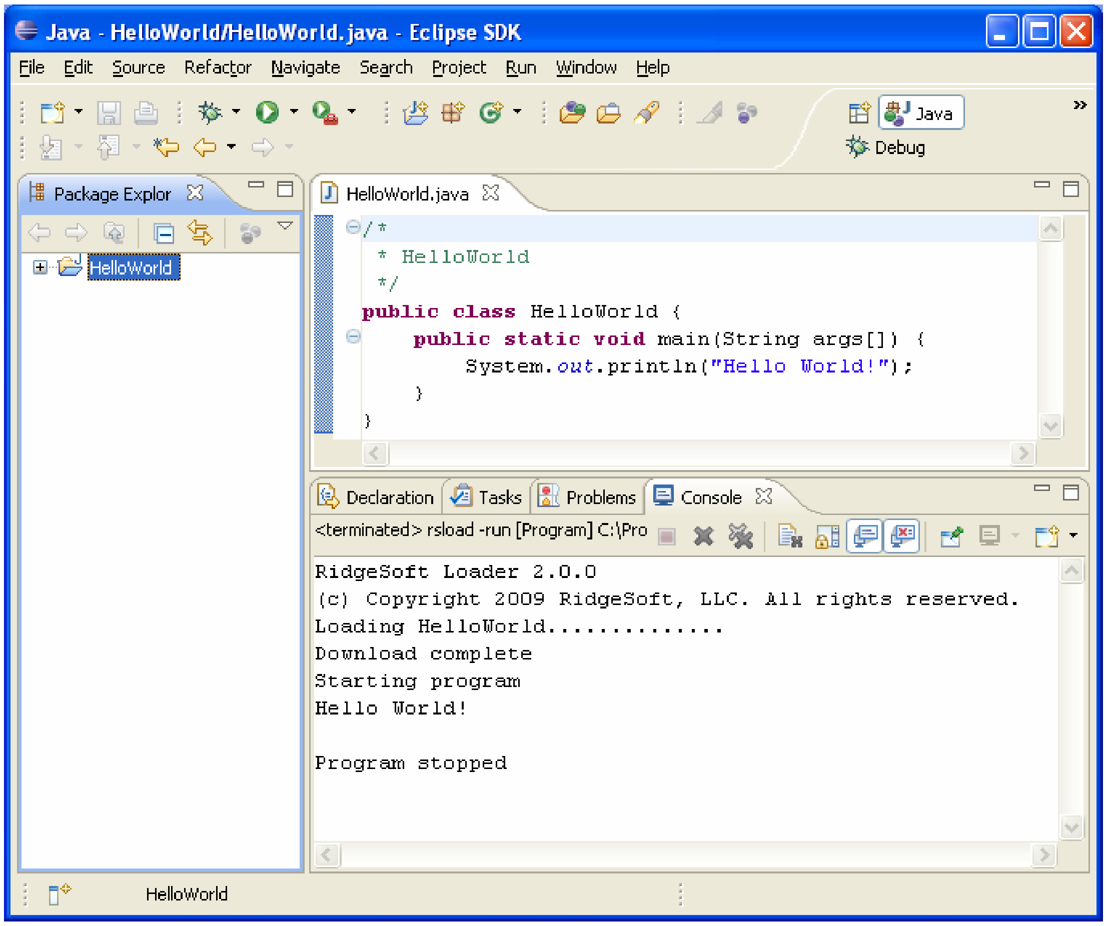
Option	Description
`-autorun`	Run the program automatically after reset. `rsload -autorun HelloWorld`
`-bank <bank>`	The memory bank to load. RAM – load the program to RAM memory FLASH – load the program to flash memory The FLASH option is not available for the Handy Board or Sumo11 controllers. `rsload –bank FLASH HelloWorld`
`-baud <rate>`	The baud rate to use. The baud rate must match the setting on the target controller. `rsload –baud 115200 HelloWorld`

Option	Description
`-bootclasspath <path>`	Specifies where to find the bootstrap class files. Only use this option if you need to substitute your own classes ahead of classes in the RoboJDE class library. Default: RoboJDE.jar in installation folder. `rsload –bootclasspath "My.jar;C:\Program Files\RoboJDE\RoboJDE.jar" HelloWorld`
`-classpath <path>`	Specifies where to find user class files. Default: Current folder then the “classes” folder within the current folder, if it exists. `rsload –classpath My.jar HelloWorld`
`-help`	Displays usage help. `rsload –help`
`-loadvm`	Download the RoboJDE virtual machine to the target controller. `rsload –target IntelliBrain –port COM2 –loadvm`
`-port <name>`	Port to use to communicate with the target controller. Default: First serial port on the computer. COM1 on Windows computers. `rsload –port COM2 HelloWorld`
`-run`	Run the program after loading, or if no class argument is specified, run the currently loaded program. `rsload –run HelloWorld`
`-stacksize <size>`	Sets the default number of stack elements the virtual machine uses for each thread it creates, including the main thread. Each stack element occupies four bytes of memory. Default: 200 `rsload –stacksize 250 HelloWorld`

Option	Description
`-target <name>`	Specifies the type of the target controller. IntelliBrain – IntelliBrain 2 or IntelliBrain robotics controller HandyBoard – Handy Board or Sumo11 robotics controller Default: IntelliBrain `rsload –target HandyBoard HelloWorld`
`-verbose`	Display verbose output. `rsload –verbose HelloWorld`

Loading and Running Programs from within Eclipse

By configuring the RidgeSoft Command Line Loader, rsload, as an external tool, you can easily download and run your programs directly from Eclipse. Figure 6 illustrates loading and running a program from within Eclipse.


Java - HelloWorld/HelloWorld.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Debug
Package Explor
HelloWorld
HelloWorld.java
Declaration Tasks Problems Console
<terminated> rsload -run [Program] C:\Pro
HelloWorld

```java
/*
 * HelloWorld
 */
public class HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello World!");
    }
}
```

```
RidgeSoft Loader 2.0.0
(c) Copyright 2009 RidgeSoft, LLC. All rights reserved.
Loading HelloWorld..............
Download complete
Starting program
Hello World!

Program stopped
```

Figure 6 - Loading and Running HelloWorld from Eclipse

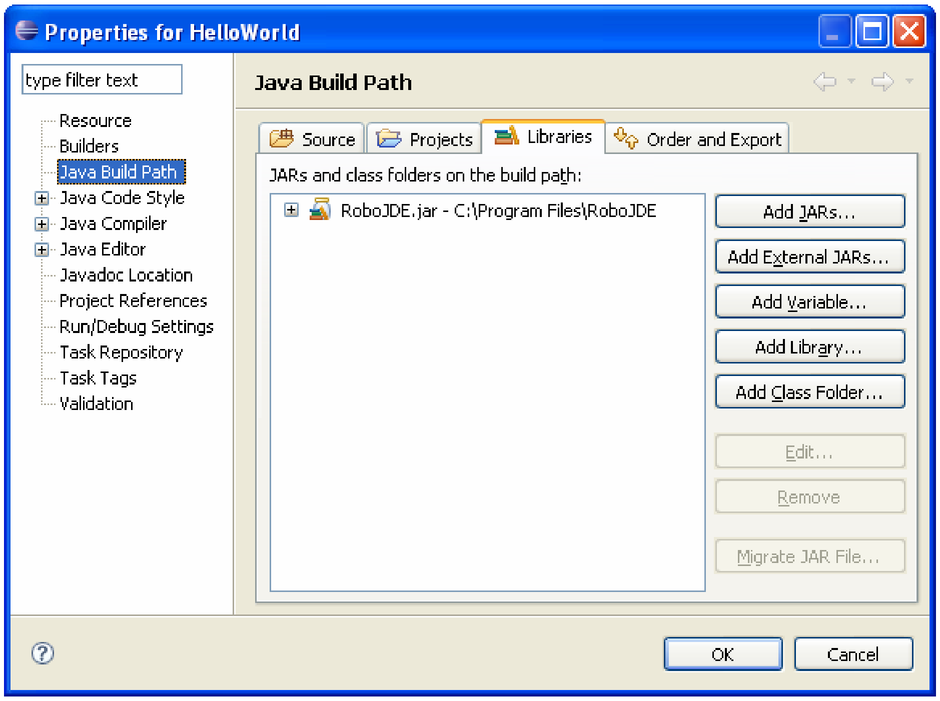
Configuring Your Eclipse Project

When creating an Eclipse project for your program, you must replace the default Java class library with the RoboJDE class library, RoboJDE.jar. Use the following steps to accomplish this.

1. Navigate to the “Libraries” tab on the “Java Build Path” page of the project properties dialog, as show in Figure 7.

2. Select and Remove the default class library from the list.

3. Click the “Add External JARs…” button, browse to and select “RoboJDE.jar” from where you installed RoboJDE.



JARs and class folders on the build path:

RoboJDE.jar - C:\Program Files\RoboJDE

Figure 7 - Configuring the RoboJDE Class Library

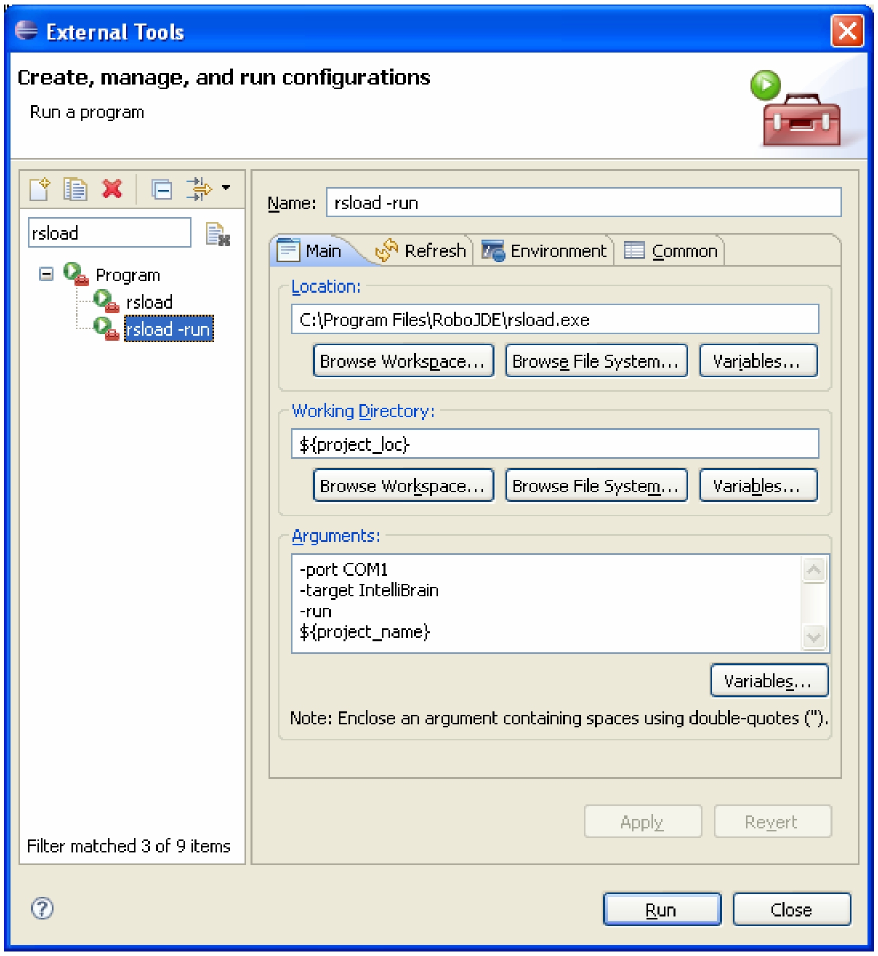
Configuring the Loader as an External Tool

Configure rsload as an external tool using the following procedure.

1. Open the External Tools dialog by clicking the arrow to the right of the external tool button ( ) on the Eclipse tool bar. The External Tools dialog is shown in Figure 8.

2. Create a new launch configuration item using the new button.

3. Enter “rsload –run” in the name field.



Filter: rsload
Filter matched 3 of 9 items

Name: rsload -run

Location:
C:\Program Files\RoboJDE\rsload.exe

Working Directory:
${project_loc}

Arguments:

```
-port COM1
-target IntelliBrain
-run
${project_name}
```

Note: Enclose an argument containing spaces using double-quotes (").

Figure 8 - Adding an External Tools Configuration

4. Click the “Browse File System…” button, browse to and select rsload from the location where you installed RoboJDE.

5. Enter “${project_loc}” in the Working Directory field.

6. Add an argument to indicate the port your computer uses to communicate with the controller. For example, enter “-port COM2” if your computer uses COM2 to communicate with the controller.

7. Add an argument to indicate the target controller type, either “-target IntelliBrain” or “-target HandyBoard.”

8. Add the “-run” argument to run the program as soon as the load completes.

9. Add “${project_name}” as the last argument.

   Note: This assumes you name your projects using the name of the main class. If this is not the case, you can either enter the literal main class name or use another Eclipse variable of your choosing that will translate to the main class name. If you enter the main class name directly, you will need to modify the launch configuration or create a new configuration each time you switch to a different project.

10. Click Apply.

11. Select the “rsload -run” configuration and click the duplicate launch configuration button.

12. Change the name of the new configuration to “rsload.”

13. Delete the “-run” argument.

14. Click Apply, and then click Close.

You have created two launch configurations, one named “rsload” and the other named “rsload –run.” The former will load the selected program and the later will load the program and run it immediately.

Loading and Running Your Program

Once you have created the external tool configurations described in the previous section, you can load your programs or load and run your programs by clicking on the appropriately named external tool item in Eclipse by using the following procedure.

1. Click on the project folder icon in the Package Explorer.

2. Click on the arrow to the right of the external tool button ().

3. Select either “rsload” or “rsload -run,” whichever is appropriate.

The output from rsload and your program will appear in an Eclipse Console window.

Eclipse sets the default external tool to the most recently used external tool. For subsequent invocations, you can simply click on the project and then the external tool button.

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 the documentation by clicking on the "index.html" file in the "apidoc" folder or by browsing to it using your web browser.

Robot Controller API Quick References

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

Table 9 - 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 many 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.

Understanding Errors, Exceptions and Stack Traces

All run-time 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 program's 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"). The map file is created 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 9. 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.

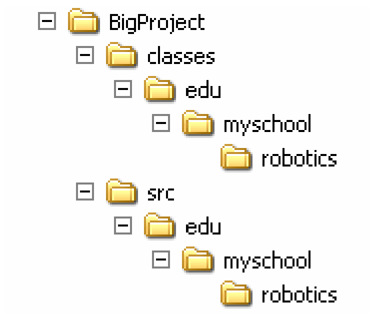


Figure 9 - Example Package Structure

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 9 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 2 controller. As shown in Table 10, the IntelliBrain 2 has more than ten times the memory available for your program than does the Handy Board or Sumo11.

Table 10 - Controller Memory Comparison

Controller	Memory
IntelliBrain 2	▪ 128K flash program memory (60K available for your Java program) ▪ 132K RAM (128K available for program data)
Handy Board / Sumo11	▪ 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 program’s 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 provide 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 11 - 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 11 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.