Creating a User Interface for Your Robot

www.ridgesoft.com

Revision 1.0

Introduction

For just about any robot project you undertake, you will need a means to interact with your robot. By developing a user interface before you get too far into your project, you will be able to easily test and debug your robot's hardware and software as you develop it. This tutorial will take you through the process of building a simple user interface that will allow you to interact with your robot.

In addition to creating a user interface, this tutorial demonstrates designing your software as reusable components. Because it provides for object-oriented software and multi-threading, Java is an excellent language for creating reusable robotics software components. In this tutorial, you will create user interface components that allow you to easily extend and modify your robot's user interface. You will also be able to use these components over and over again as you embark on different robotics projects. And, the data displayed on the LCD screen will update without interfering with other more important tasks your robot is doing.

Before You Get Started

This tutorial builds on concepts discussed in the following tutorials:

Creating Your First IntelliBrain Program
Programming Your Robot to Perform Basic Maneuvers

If you are not already familiar with the concepts covered in these tutorials, we recommend you complete them before completing this tutorial. These tutorials are available from the RidgeSoft web site, www.ridgesoft.com.

The programming steps in this tutorial start off with the MyBot program developed in the *Creating Your First IntelliBrain Program* tutorial.

User Interface Requirements

Before starting any programming project, it is always a good idea to consider the requirements of the feature(s) you are implementing. Usually, you will need to carefully consider the problem you are solving to discover the requirements. However, for this project the requirements will be given to you.

The requirements for the user interface you will be building are as follows:

1. Display output using the IntelliBrain controller's two line LCD module.
2. Provide the ability to display multiple screens of data.
3. Allow the user to select which of the screens of data to display by using the IntelliBrain controller's thumbwheel.
4. Allow the user to select which of several pre-programmed functions to execute using the IntelliBrain controller's push buttons.

5. Periodically update the current screen without interfering with what the robot is otherwise doing.
6. Implement the user interface components in such a way that they can easily be reused.

If you wanted to support more advanced user input and remote control, you could use a TV remote control for input instead of the push buttons and thumbwheel. The IntelliBrain controller supports this via an infrared receiver module on the IntelliBrain main board. However, we will leave this as an exercise for you once you've completed this tutorial.

Creating Reusable Software Components

One of your requirements as you design and implement your user interface is to create software components that are easy to reuse in future robotics projects. Three keys to achieve this goal are:

1. Creating components that are cohesive and provide useful functionality.
2. Creating components such that they have minimal interdependencies – in other words they are loosely coupled to the rest of the system.
3. Designing generic interfaces to components that promote interchangeability.

Java and Software Reusability

Java was designed from the ground up to support object-oriented programming, a software development paradigm that is ideal for developing cohesive software components and loosely coupled software systems. In addition to being object-oriented, Java supports multi-threading, making it much easier to implement multi-tasking real-time systems, such as a robot, with minimal coupling between components.

Java has a built-in mechanism for defining and using software interfaces, allowing a variety of software components based on a common software interface to be used interchangeably.

Java also provides a means to share pre-built software components without dependencies on vendor specific development tools, like a compiler or assembler, or dependencies on a specific microcontroller. Instead, pre-built components can be built once, packaged, and shared without end users needing to be concerned about any of these things.

As you complete this tutorial, you will leverage these strengths of Java to create a simple user interface that consists of a few cohesive components that are loosely coupled to the other portions of your robot's software.

Developing User Interface Classes

Accessing the IntelliBrain Controller's LCD Display

As was demonstrated in the MyBot example in the *Creating Your First IntelliBrain Program* tutorial, you can get a reference to the object that allows your program to interact with the IntelliBrain controller's display, as follows:

```
Display display = IntelliBrain.getLcdDisplay();
```

The reference returned by this call refers to an object that implements the Display interface, an interface which defines methods for interacting with any multi-line, character-oriented display.

You can become more familiar with the details of the IntelliBrain class, the getLcdDisplay method, and the Display interface by viewing the RoboJDE API documentation. If you are not familiar with using the API documentation, the tutorial *Programming Your Robot to Perform Basic Maneuvers* includes instruction on how to use the API documentation.

Creating the Screen Interface

You can print text to the IntelliBrain LCD screen by using the object reference obtained from the getLcdDisplay method. However, to meet the previously stated requirements, your user interface will need to have the ability to display multiple screens of text that you can scroll through using the thumbwheel. You can accomplish this by defining a "Screen" interface. This will allow you to develop a variety of screens that each displays different information. It will also allow each screen to be plugged into, or removed from, the user interface with minimal effort. That is, your screens will be loosely coupled to the rest of your program.

Each screen must have the ability to display its own data when it is selected. Therefore, your Screen interface will need a method to update the display, as follows:

```
public void update(Display display);
```

Your program will be able to update the text displayed on the LCD screen by calling the update method of whichever Screen object the user selects using the thumbwheel.

Perform the following steps to create the Screen interface:

1. Start by using RoboJDE to open the project you want to add the user interface to. We will use the MyBot project developed in the *Creating Your First IntelliBrain Program* tutorial.
2. Select File->New Class menu item.

3. Enter “Screen” as the class name in the New Class dialog, and then click OK.
4. Enter the following statements in the window for the Screen.java class:

```
import com.ridgesoft.io.Display;

public interface Screen {
    public void update(Display display);
}
```

5. Click the Save All button on the tool bar.

Implementing a Screen to Display Static Text

Now that you’ve defined the Screen interface, you need to create a class that implements the Screen interface and displays some interesting data. A good screen to start with is one that simply displays two lines of unchanging text. You will be able to use this to display the name and version number of your program.

Use RoboJDE’s File->New Class menu item and creating a “StaticTextScreen” class. Edit the new class to include the following code:

```
import com.ridgesoft.io.Display;

public class StaticTextScreen implements Screen {
    private String mLine1;
    private String mLine2;

    public StaticTextScreen(String line1, String line2) {
        mLine1 = line1;
        mLine2 = line2;
    }

    public void update(Display display) {
        display.print(0, mLine1);
        display.print(1, mLine2);
    }
}
```

This class declares that it implements the Screen interface; therefore, Java requires that it also implement the update method defined by the Screen interface. The update method simply prints the two predefined Strings to the two lines of the display. You will specify the two lines of text by passing them as arguments to the constructor when you construct each StaticTextScreen later in this tutorial.

Managing Multiple Screens

Your next step is to create a class that will keep track of several Screen objects and periodically call the update method of the currently selected Screen.

Recalling the requirements, you need to allow the display to be updated while your robot is doing other things. Fortunately, Java provides multi-threading which makes it easy for your programs to perform multiple tasks at the same time. In case you are not familiar with the concept of multi-threading, let's digress briefly to discuss it.

Multi-Threading Explained

Multi-threading allows your program to perform multiple tasks concurrently. To implement multi-threading, you must structure your program into separate tasks that will execute concurrently.

As a simple analogy, consider a chef cooking the main course of a meal. The chef must prepare all of the items that make up the main course concurrently so the entrée, sauces and side dishes are all ready to eat at the same time. The chef prepares the meal by breaking his work into several smaller tasks which consist of preparing each individual item according to a recipe for that item. Rather than cooking each item in its entirety before starting to cook the next item, the chef prepares all of the items concurrently by repeatedly working on each item for a short period of time then proceeding to tend to the other items. By working in this fashion the chef performs multiple tasks at the same time, preparing each of the individual dishes to accomplish the larger goal of creating the entire main course.

As you develop your robotics program, you will find you need to program your robot to do many things concurrently. Just as a chef organizes his work into smaller tasks, you must organize your robot's work into tasks that execute concurrently. Fortunately, Java provides multi-threading support to help with this.

To make use of Java's multi-threading features, you will need to organize your program into separate tasks that consist of their own "thread" of work. In the analogy above, the chef executes each individual item's recipe as a separate thread of work. That is, when working on a particular item, he gives his exclusive attention to that item, picking up that thread of work where he last left off and continuing to execute that item's recipe. When it comes time to tend to another item, he suspends that current item's thread of work and switches his attention to another item, continuing the thread of work for that item.

Conveniently, maintaining up to date text on the LCD screen is a well defined cohesive task that can be easily isolated from other tasks your robot will perform. Therefore, it can be considered its own thread of work that you can implement by extending the Thread class provided by Java.

Creating a ScreenManager Class

Your ScreenManager class will be responsible for updating the text displayed on the LCD screen. To do this task, your program will need to periodically read the position of the thumbwheel to determine which of a list of screens to display. The

program will then need to call the selected screen's update method to update the text on the display. You can take advantage of Java's multi-threading capability by extending the Thread class, which is part of the RoboJDE class library.

Use RoboJDE's File->New Class menu item to create a "ScreenManager" class. Edit the new class to include the following code:

```
import com.ridgesoft.io.Display;
import com.ridgesoft.robotics.AnalogInput;

public class ScreenManager extends Thread {
    private Display mDisplay;
    private Screen[] mScreens;
    private AnalogInput mUserInput;
    private int mPeriod;

    public ScreenManager(Display display,
                         Screen[] screens,
                         AnalogInput scrollDevice,
                         int threadPriority,
                         int period) {
        mDisplay = display;
        mScreens = screens;
        mUserInput = scrollDevice;
        mPeriod = period;
        setPriority(threadPriority);
        start();
    }

    public void run() {
        try {
            int divisor = mUserInput.getMaximum() + 1;
            while (true) {
                try {
                    int index = (mUserInput.sample() *
                                 mScreens.length) / divisor;
                    mScreens[index].update(mDisplay);
                } catch (Exception e) {
                    e.printStackTrace();
                }
                Thread.sleep(mPeriod);
            }
        } catch (Throwable t) {
            t.printStackTrace();
        }
    }
}
```

The ScreenManager class declaration states that this class "extends Thread" which means it extends the Thread class in the RoboJDE class library. To extend the Thread class, the ScreenManager class must implement a run method. A Thread class's run method executes on its own thread and is responsible for performing all of the work for that thread. Therefore, the

ScreenManager class's run method is responsible for executing all of the code necessary to periodically update the LCD screen.

In order to update the display, the ScreenManager class's run method needs: 1) a reference to the list of available screens to display, 2) a reference to the analog input for the thumbwheel, 3) a reference to the display to output to and 4) the period of time to sleep between updates to the screen; therefore, the ScreenManager class includes four member variables (mScreens, mUserInput, mDisplay and mPeriod) corresponding to these items.

The constructor method for the ScreenManager class initializes the four member variables with data provided as arguments to the constructor when the object is created. The constructor also has one additional argument to specify the priority of the ScreenManager thread. After initializing the member variables, the constructor sets the thread priority and calls the start method to start the thread. This causes the run method to start executing on a separate thread.

Finally, the ScreenManager class's run method contains the code to periodically update the text displayed on the LCD screen.

Out of the dozen or so lines of code in the run method, just two lines do all of the work of the ScreenManager. The line

```
int index = (mUserInput.sample() * mScreens.length) / divisor;
```

samples the user input (thumbwheel) and scales the sampled value to calculate the array index of the screen to display.

The list of screens is an array, which is a basic type of data in Java. An array consists of a number of elements that can be accessed using a zero-based index. For example, if there were three screens to choose from, the first screen would be accessed using zero as the index (mScreens[0]), the second screen using one as the index (mScreens[1]) and the last screen using two as the index (mScreens[2]). Arrays also have a fixed length. The number of available screens is the same as the length of the array (mScreens.length). If there were three screens, the length of the array would be three.

The line,

```
mScreens[index].update(mDisplay);
```

calls the update method of the selected Screen, which will print the appropriate data to the display.

These lines of code are within a "while" loop that runs forever. The loop also includes a call to the Thread.sleep method, which tells Java's thread scheduler to stop execution of the thread for the number of milliseconds specified by the

argument. Other threads can continue to perform their work while this thread sleeps. Also, other higher priority threads can suspend this thread even when it is not sleeping.

You will find making use of the thread priority mechanism is very useful in robotics programs. This allows you to specify which tasks are most important. If multiple threads are eligible to execute at the same time, the thread priority value enables the Java virtual machine to select the most important thread to execute. For example, you will most likely want navigation of your robot to take priority over updating the LCD display. By giving the ScreenManager lower priority than navigation threads the Java virtual machine will choose to delay screen updates in favor of executing the navigation threads to keep your robot on course.

The run method also includes two try-catch blocks. The way errors are reported in Java is by “throwing” an exception object that contains specific information about the error. The exception object gets thrown from the currently executing method back through the method call chain to the first method that catches it. If no method catches the exception, Java exits (terminates) the thread. The nice thing about handling errors this way is that a method that can’t do anything about a particular error doesn’t need to even consider the error. In procedural languages like C, you frequently end up with a lot of messy code in every function to deal with errors. With exception-based error reporting, it isn’t necessary to include a lot of error handling code in every method.

The inner try-catch block in the ScreenManager class’s run method will catch exceptions of class Exception that occur when calling the update method of the selected screen. The handler code in the catch block simply prints the stack trace and allows the screen update loop to continue executing. If there is a problem displaying one screen, it won’t prevent the ScreenManager from displaying another screen when you turn the thumbwheel.

The outer try-catch block in the ScreenManager class’s run method catches all exceptions of class Throwable, which is the base class for all exceptions. Errors that aren’t caught by the inner try-catch block will be caught by the outer try-catch block. The outer try-catch block will report any error that causes the ScreenManager thread to exit. Without the outer try-catch block, errors would cause the thread to exit silently, and it wouldn’t be obvious the ScreenManager thread wasn’t running anymore.

Function Selection

The final requirement you will need to address is the ability to select which of several preprogrammed functions the robot should execute. This only needs to run briefly when the program is starting. Therefore, you can put the selection code in the main method of the MyBot class, as will be described later.

To keep your program well organized, you will want each function to be implemented independently of other functions. You can make use of the Runnable interface to create separate function classes that can be selected and run. The Runnable interface requires implementing classes to provide a method named “run.” The run method needs to contain the code to carry out the specific function. You can create a couple of simple function class for testing purposes.

Use RoboJDE’s File->New Class menu item to create a DoBeep class to beep the IntelliBrain controller’s buzzer, as follows:

```
import com.ridgesoft.intellibrain.IntelliBrain;
import com.ridgesoft.io.Speaker;

public class DoBeep implements Runnable {
    public void run() {
        Speaker speaker = IntelliBrain.getBuzzer();
        speaker.beep();
    }

    public String toString() {
        return "Beep";
    }
}
```

The run method of the DoBeep class beeps the buzzer. The toString method will allow your function selection code to display the name of the function.

Use RoboJDE’s File->New Class menu item to create a second test class named DoNothing, as follows:

```
public class DoNothing implements Runnable {
    public void run() {
    }

    public String toString() {
        return "Do Nothing";
    }
}
```

Tying it all Together

You will now need to update your main class, MyBot, to incorporate the user interface classes you have created. The main method in your MyBot class will need to create a list of functions that you can select from and a list of screens you can select from when you run your program.

Update the MyBot class as follows:

```
import com.ridgesoft.io.Display;
import com.ridgesoft.robotics.PushButton;
import com.ridgesoft.intellibrain.IntelliBrain;
```

```
public class MyBot {
    public static void main(String args[]) {
        try {
            Display display = IntelliBrain.getLcdDisplay();
            PushButton startButton =
                IntelliBrain.getStartButton();
            PushButton stopButton =
                IntelliBrain.getStopButton();

            Runnable functions[] = new Runnable[] {
                                        new DoBeep(),
                                        new DoNothing(),
                                    };

            startButton.waitReleased();
            IntelliBrain.setTerminateOnStop(false);
            int selectedFunction = 0;
            display.print(0, "Function");
            display.print(1,
                functions[selectedFunction].toString());

            while (!startButton.isPressed()) {
                if (stopButton.isPressed()) {
                    if (++selectedFunction >= functions.length)
                        selectedFunction = 0;
                    display.print(1,
                    functions[selectedFunction].toString());
                    stopButton.waitReleased();
                }
            }
            IntelliBrain.setTerminateOnStop(true);

            Screen[] screens = new Screen[] {
                new StaticTextScreen("MyBot", "Version 0.3"),
                new StaticTextScreen("Screen 1", "abcd"),
                new StaticTextScreen("Screen 2", "1234"),
            };

            new ScreenManager(display,
                              screens,
                              IntelliBrain.getThumbWheel(),
                              Thread.MIN_PRIORITY,
                              500);

            functions[selectedFunction].run();
        }
        catch (Throwable t) {
            t.printStackTrace();
        }
    }
}
```

The updated MyBot main method now contains the code to allow you to select one of two functions for the robot to perform – DoBeep or DoNothing. It also contains the code to initialize the screen manager with three screens.

The function selection code runs first before the screen manager starts. The function selector uses both of the IntelliBrain controller's push buttons. In order to use the STOP button, the RoboJDE Java virtual machine has to be told to not terminate the program when the STOP button is pressed. The call to the setTerminateOnStop method with the argument false does this.

As long as the START button is not pressed, the function selector code executes a while loop checking to see if the STOP button is pressed. Whenever the STOP button is pressed the current selection is changed to the next function on the list. When the end of the list is reached, the selection returns to the beginning of the list. Once the START button is pressed the loop exits and the current selection is the function the program will execute.

The program next proceeds to create an array of three screens and then initializes the ScreenManager. This will start the screen manager thread and the LCD screen will begin updating.

Finally, the program calls the run method of the selected function to carry out the function.

The class diagram in Figure 1 shows the relationship of the classes you've developed in this tutorial.

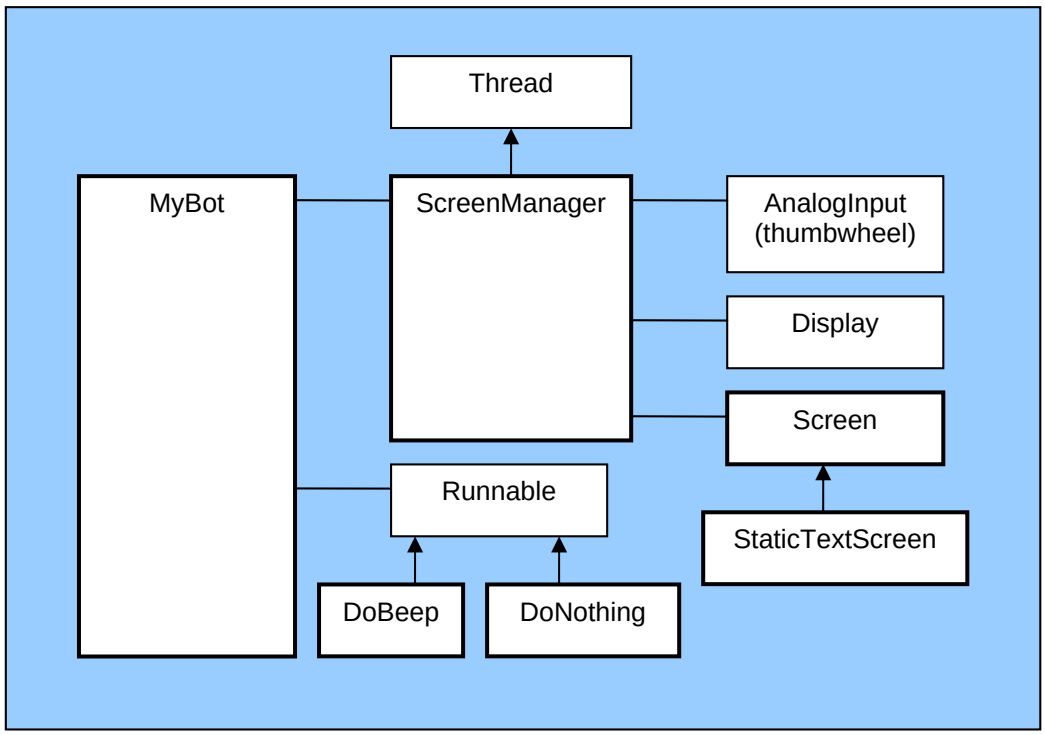

Figure 1 - Class Diagram

Testing Your Program

You are now ready to test our program. You can compile and download the program by clicking on the download button in RoboJDE. You can start the program executing by pressing the button labeled START on the IntelliBrain controller. Pressing the STOP button repeatedly will step you though the list of functions, displaying the name of the next choice of function with each button press. When you've found the function you want to run, press the START button again. You can then use the thumbwheel to scroll through and view the three screens.

Conclusion

You have created several components that provide a simple and easy to extend user interface to your robot. As you add features to your robot you will be able to extend the user interface to incorporate the new features. You will find the ability to add new screens extremely useful when adding new sensors. You can add screens to display data sampled from each new sensor you add to your robot, enabling you to understand and verify the function of each sensor. You will also find it handy to display data calculated by your program to verify the function of your program.

Exercises

1. Add a new screen to display your name.
2. Add a new screen to sample and display the current value of an analog input port.
3. Add a new screen to sample and display the current value of a digital input port.
4. Add a new function to blink the green status LED once per second.
5. Add a new function to make the robot go forward for five seconds.
6. Enhance your user interface to use a TV remote control as an input device instead of using the thumbwheel and push buttons for input.

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