REALbasic QuickStart

Welcome to REALbasic!

REALbasic is an application builder based on a modern version of the BASIC programming language. The REALbasic development environment is made up of a rich set of *graphical user interface* objects (commonly referred to as a *GUI*), an object-oriented language, a debugger, and a cross-platform compiler.

REALbasic provides you with all the tools you need to build virtually any application you can imagine.

If you are new to programming, you will find that REALbasic makes it fun and easy to create full-featured Mac OS and Windows applications. If you are an intermediate or advanced programmer, you will appreciate REALbasic's rich set of built-in tools.

This *QuickStart* is for people who are new to programming and new to REALbasic. It will give you a gentle introduction to the REALbasic development environment, lead you through the development of a real application, and show you what kinds of other applications can be built with REALbasic.

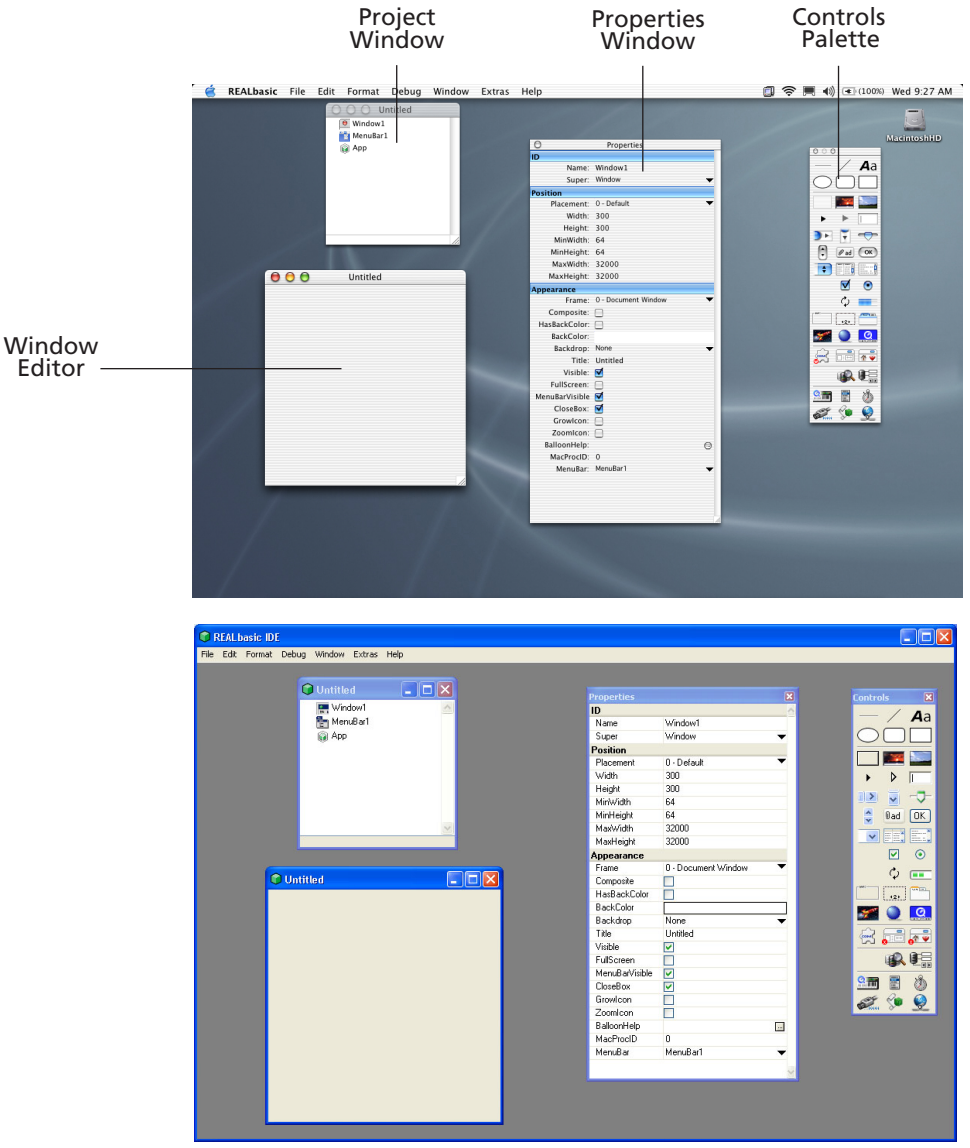
It should take you no longer than 30 minutes to complete this QuickStart.

Note If you have experience with other versions of the BASIC language or have experience with other programming languages, you'll want to check out the *Tutorial* and *User's Guide*. In the *Tutorial*, you'll develop a more complete application, which includes menus, dialog boxes and sheet windows, and standalone objects called *modules*.

Starting REALbasic

If you haven't done so already, install REALbasic on your hard disk and double-click the REALbasic application icon. In a few moments, the REALbasic *integrated development environment* (IDE) appears. If you're using Windows, the Project Window and Window Editor appear within the REALbasic IDE multiple document interface window and the Properties Window and Controls Palette float above this window.

Figure 1. The REALbasic IDE (Mac OS X and Windows XP).



Note In Figure 1, some windows have been moved from their default positions so you can see them better.

REALbasic's Windows

As you can see in Figure 1 on page 2, there are four windows that open when you start up REALbasic:

- The *Project Window* contains a list of all of the major items that make up your REALbasic application. For example, the Project Window includes items for all the windows that your application uses, the menu bars, and objects such as sounds, pictures, databases, and QuickTime movies. By default, the Project Window includes an item for the application's main window, *Window1*, its default menu bar, *MenuBar1*, and an item for code associated with the application as a whole, *App*. You double-click an item in the Project Window to edit or view it.
- A *Window Editor* is where you build windows, dialog boxes, alert boxes, and palettes for the application. Each such window is opened in its own Window Editor and all the windows in the application are listed in the Project Window. By default, an application has one window that opens automatically when you launch the application. In Figure 1, the Window Editor is empty because you haven't added any interface items yet. You can add as many windows as you like.
- The *Controls Palette* contains icons representing interface objects that you can add to a window. You build your application's interface by dragging and dropping icons from the Controls Palette to a Window Editor.
- The *Properties Window* lists the properties and their values of an item you select in a Window Editor or a menu item. You can add or modify an object's properties by editing the values in the Properties Window. In Figure 1, the main window is selected, so the Properties Window shows the properties of this window.

  In addition, the complete REALbasic *Language Reference* is online. Choose Help ▶ Language Reference to display the online reference. Use it as a convenient alternative to the printed or electronic (PDF) version of the *Language Reference*.

Getting Started

In this QuickStart, you'll build an application that manages URLs and email addresses. A URL (which stands for *Uniform Resource Locator*) is the address of a web page that you type into the Address area in your web browser. This application will launch your default web browser application and display the web page that you entered.

Figure 2 shows a URL in a browser Address area:

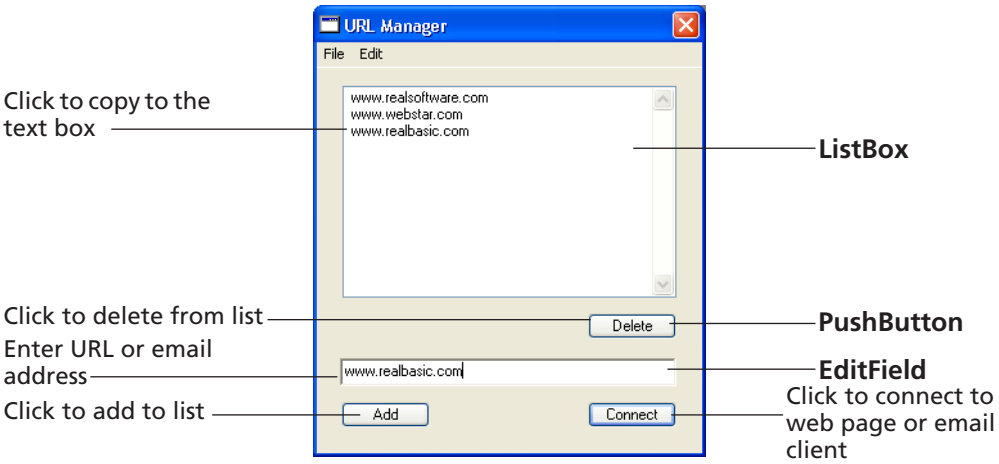
Figure 2. A URL entered into a browser.

If you enter an email address instead of a URL, the application will launch your email application, create a new blank email, and enter the address into the "To" box. Figure 3 shows how to enter an email address into a browser's Address area.

Figure 3. An email address entered into a browser's Address area.

When you are finished, the application will look like this. The labels in bold indicate the types of controls that were used in creating the application. The functions of the controls are described in plain type.

Figure 4. The finished URL Manager application.



You use the EditField to enter the URL or email address you want and then click the Connect pushbutton. To save it in the list, you click the Add button. To select a URL in the ListBox, highlight it in the list and then click Connect. If you don't need the URL any more, highlight it in the list and then click the Delete button.

The application uses three types of prebuilt interface elements do to most of the work for you:

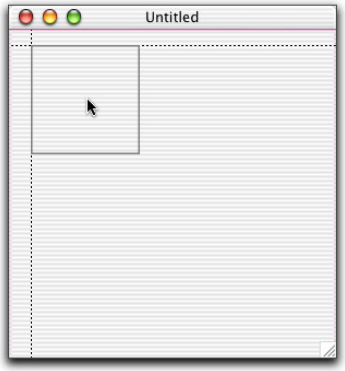
- A **ListBox** is the type of control that holds scrollable lists. It can hold both single- and multiple-column lists and scroll horizontally and vertically.
- An **EditField** is the type of control that holds text. It can be used to hold either a single row of text (as it is here) or as a text editor.
- A **PushButton** is a standard pushbutton. It is most often used to initiate an action (as it is here).

Creating the Interface

The your first task is to build the interface. You already have the REALbasic development environment open.

1 In the Controls palette, hold down the mouse button on the ListBox icon  and drag it to the top-left corner of the Window Editor.
When you position it as shown below, alignment guides appear.

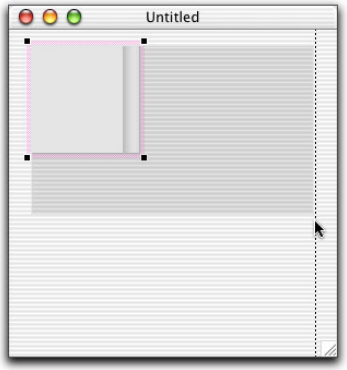
Figure 5. Alignment guides help you position the ListBox and align other controls.



2 Release the mouse button.
A square ListBox appears in the Window Editor.

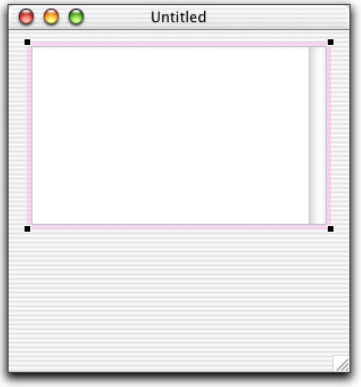
3 Use the resizing handle in the bottom-right corner and drag diagonally to enlarge the ListBox. When the alignment guide appears on the right, release the mouse button, as shown in Figure 6.

Figure 6. Enlarging the ListBox.



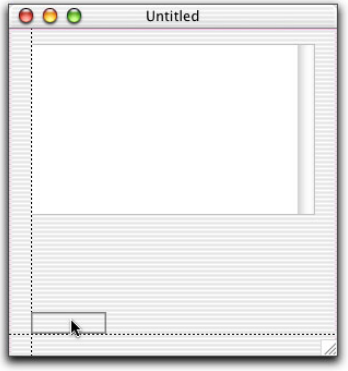
Center it in the upper area of the window, as in Figure 7.

Figure 7. A ListBox added to the Window.



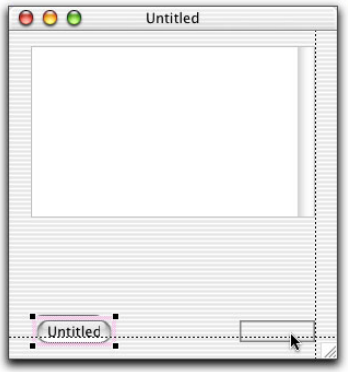
4 Next, drag a Pushbutton control OK from the Controls Palette to the lower-left portion of the window (where the "Add" button in Figure 4 on page 4 is).
As you drag toward the left edge of the ListBox, a vertical alignment guide appears. Use the vertical alignment guide to position the Pushbutton, as shown in Figure 8.

Figure 8. Aligning the Add pushbutton.



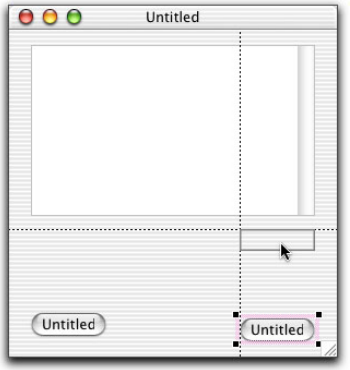
5 With the Pushbutton selected, press ⌘-D or Ctrl+D (or choose Duplicate from the Edit menu) to make a copy of the Pushbutton. Move it to the right of the previous button and aligned with the right edge of the ListBox.
When you reach the approximate position, both vertical and horizontal alignment guides appear. Align the baselines of the two PushButtons' captions, as shown in Figure 9.

Figure 9. Aligning the Add and Connect pushbuttons.



6 With this Pushbutton selected, hold down the Option key (Alt key on Windows) and drag the PushButton vertically, just below the ListBox. Release the mouse button when the horizontal alignment guide appears.

Figure 10. Adding the Delete button by Option-dragging the Connect button.

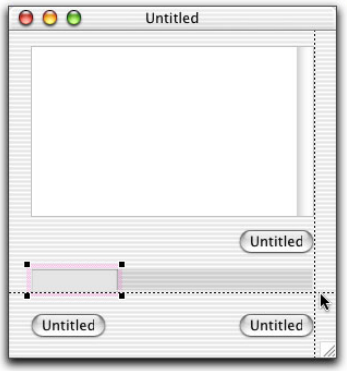


7 In the Controls palette, hold down the mouse button on the EditField icon and drag it into position in the window, midway between the Delete and Connect buttons (see Figure 11).

8 Align the left edge of the EditField with the left edge of the ListBox.

9 Use one of the EditField's selection handles to stretch it so that its right edge aligns with the ListBox.

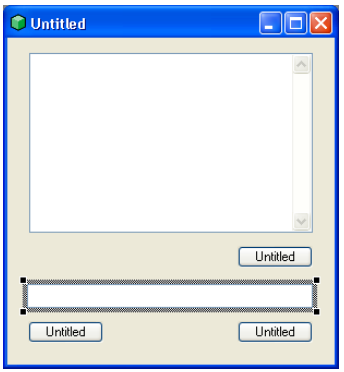
Figure 11. Aligning the right side of the EditField with the ListBox and Pushbuttons.



Note You can also position a control by clicking on it and moving it one pixel at a time using the arrow keys.

Your application's interface is now complete! It should look something like Figure 12 on page 8:

Figure 12. The finished interface in the development environment.



Before going any further, save the project.

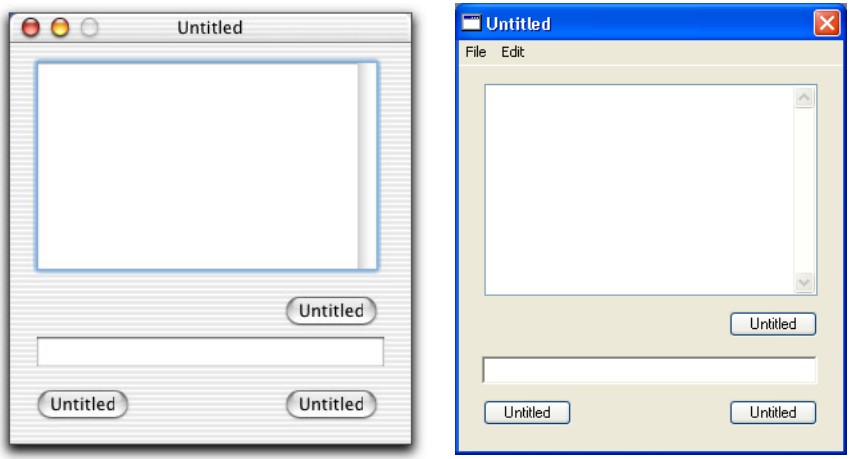
- Choose File ▶ Save (⌘-S or Ctrl+S) and name it **URL Manager.rb**.

At this point, you can actually try it out. Of course it won't do anything since we haven't told any of the interface elements what to do.

1. Just for fun, choose Run from the Debug menu.
REALbasic builds the application and opens it in its own window.

The first version of the application should look like this:

Figure 13. The first version in the Runtime environment.



In this state, the PushButtons work — that is, you can click them and they highlight — but they don't do anything because we haven't told the PushButtons what to do when they're clicked. You can enter text into the EditField — but it doesn't go anywhere because there are no instructions to process this text. And the ListBox is all set to display and scroll items but we don't have any way to get text into the ListBox yet.

But it's a start. We need to go back to the development environment to get this application operational.

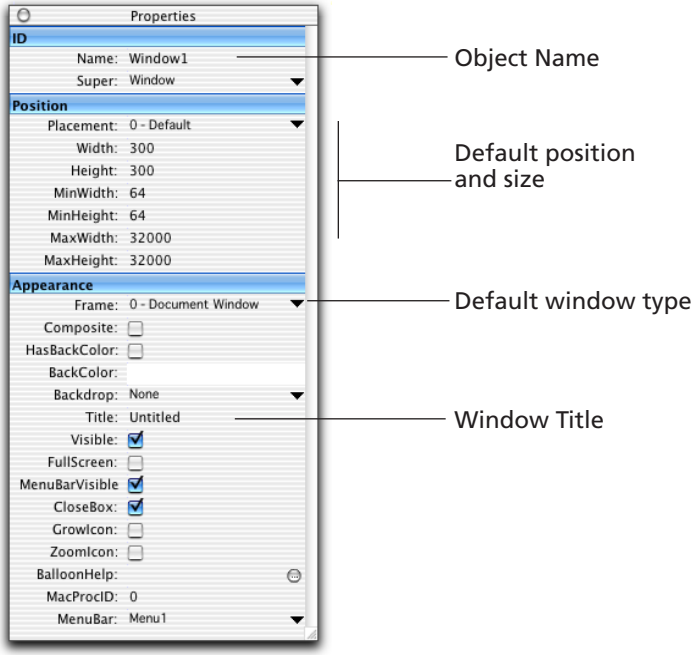
2 Choose File ▶ Exit on Windows or REALbasic ▶ Quit (in Mac OS X) to return to the integrated development environment.

Giving Objects Meaningful Names and Labels

You've already noticed that quite a few objects have the default label "Untitled." They also have default names like PushButton1, PushButton2, etc. Before getting too far into the project, we should give the objects meaningful names and labels. The names are used internally in our REALbasic code and, of course, the labels are presented to the user. If you'll refer to an object in your code, you should give it a meaningful name at the start of the project.

1 In the Development environment, click on the Window Editor — not one of the controls in the window — and then take a look at the Properties Window. (If the Properties Window is not onscreen, choose Window ▶ Show Properties.)
It has the default name "Window1", which is shown in the first line.

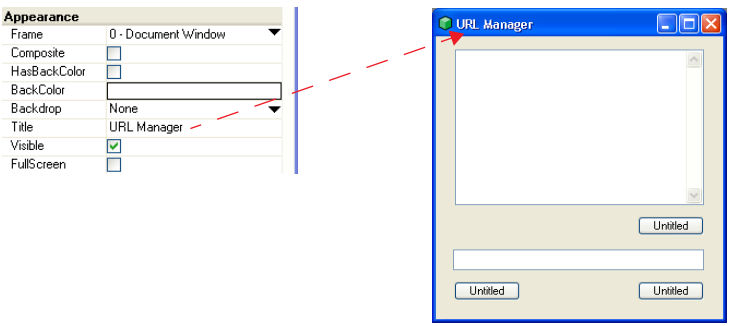
Figure 14. Window1's Properties Window.



The text that appears in the Title bar of the window is the Title property.

2 Select the default Title, "Untitled", in the Properties Window and replace it with **URL Manager**, and press the Return key. When you press Return, the new title appears in the Title bar of the Window Editor as well as in the Properties window.

Figure 15. Changing Window1's Title property.



Similarly, we need to replace the default names and labels for the controls in the window.

3 In the Window Editor, click on the Untitled button in the lower left.

Notice that the Properties Window changes to show the properties of this control. The three PushButtons are named PushButton1, PushButton2, and PushButton3.

They were named in the order they were created. We'll never remember which one is which, so it's best to rename them at the same time we're entering their labels.

4 Change its Name property to **AddURL** and its Caption property to **Add**. Press Return to save each new property value.

Notice that the Caption text immediately replaces "Untitled" in the Window Editor when you press Return. When you are finished, the Properties Window for the AddURL pushbutton should look like this.

Figure 16. The Properties Window for the Add button.

5 Click on the Untitled button in the lower right to select it. Use the Properties Window to change its name to **ConnectURL** and its Caption property to **Connect**.

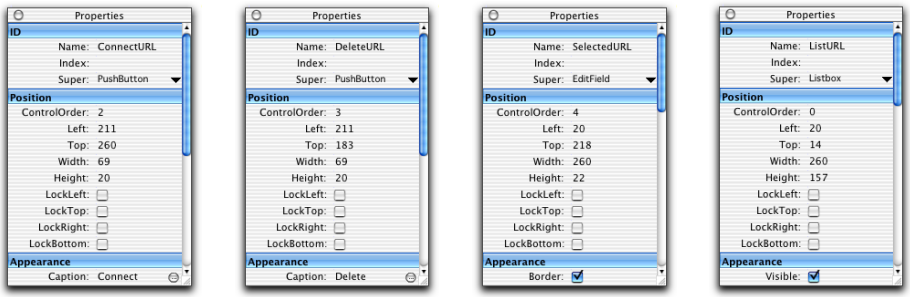
6 Click on the Untitled button between the ListBox and the EditField and change its Name property to **DeleteURL** and its Caption property to **Delete**.

7 Click on the EditField and change its name to **SelectedURL**.

8 Click on the ListBox and change its Name property to **ListURL**.

That takes care of it. The Properties windows for these objects should look as shown in Figure 17.

Figure 17. The Properties Windows for the other controls.



Connect button | Delete button | EditField | ListBox

Check your work to be sure that the items are named correctly. If there is a spelling error, code that is supposed to refer to an item will not work.

The three buttons should now look like this:

Figure 18. The three PushButton controls after renaming.

9 Choose File ▶ Save to save your changes. Choose Debug ▶ Run to test it.
It doesn't do any more than the last version, but at least all the interface elements are labelled correctly.

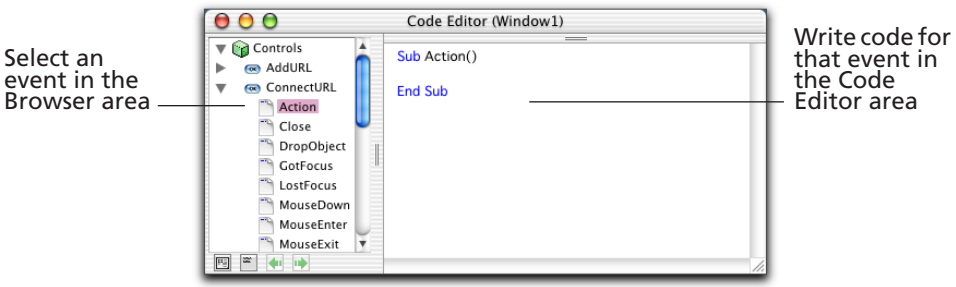
10 Choose File ▶ Exit (on Windows) or REALbasic ▶ Quit (Mac OS X) to return to the Development environment.
Be sure to quit out of the built application before resuming your work in the REALbasic Development environment.

Making the URL Manager Do Something

Now that the interface is designed and its appearance is touched up, it's time to make the controls to their jobs. We'll start with the Connect button.

1 In the Window Editor, double-click the Connect button.
The Code Editor for the application window appears. On the left side is a browser area that lists all the controls that we've added to the window, among other things. (For the QuickStart, we only need to work with the controls.) On the right side is the code editing area. It holds the code for the item that is selected in the browser.

Figure 19. The Code Editor for Window1.



In order to get the Connect button to do something, we need to write some code that will run only when the button gets clicked. Fortunately, the REALbasic application itself monitors all user interface activity while any application is running and it knows whenever this happens. We need to write the instruction that connects the user to the web site that the user enters it in the EditField.

You'll see a list of events that REALbasic continuously monitors while the application is running. The one we need is the "Action" event. This takes place

when the user clicks the button. This event is selected automatically when you double-click a PushButton in a window.

On the right, you can write the code that runs automatically when the user clicks the button. (Notice that the first line in the Code Editor, "Sub Action()", indicates which event the code is for). The instruction to open a web site in the user's browser (or open the email application) is simple. The instruction is ShowURL and its syntax is:

```
ShowURL text
```

Where *text* is the URL (or email address).

As soon as you enter "ShowURL", a "tips" window appears, showing the syntax. By referring to the Tips window you can avoid having to look up a command's syntax in the Online Reference whenever you need help.

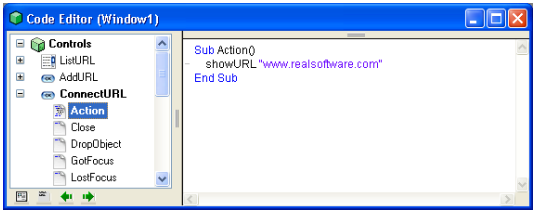
Figure 20. The 'Tips' window showing the syntax for ShowURL.

2 To test out this button, enter the following line in ConnectURL's Action event:

```
ShowURL "http://www.realsoftware.com"
```

Your Code Editor should now look like this:

Figure 21. The code for ConnectURL's Action event.



3 Choose Debug ▶ Run to test out the Connect button.

4 Click the Connect button.

In a few moments, your default web browser will launch and bring up the REAL Software home page. (This, of course, assumes your computer has a connection to the Internet and you have a default browser application.)

Figure 22. The REAL Software home page.

Of course, we need to modify the code so that the text passed to the ShowURL command can be entered by the user while the application is running. When we use ShowURL "http://www.realsoftware.com", the particular URL is "hardcoded."

5 Choose File ▶ Exit on Windows (or REALbasic ▶ Quit) to return to the Development environment.

In this case, we want to use the contents of the EditField. The user can enter a URL or an email address while the application is running or click on an item in the ListBox to copy it into the EditField.

Since the EditField is named "SelectedURL", you might think that we could write:

```
ShowURL SelectedURL
```

but that won't work because "SelectedURL" is the name of the object itself. It has lots of properties — like its position in the window, whether it takes several lines of text or just one, whether it can accept styled text, and so forth. If you use "ShowURL SelectedURL", REALbasic would have no idea what you mean.

When you need to refer to one of an object's properties, you write the name of the object, followed by a dot, followed by the name of the property. In other words, you use this syntax:

```
objectname.propertyname
```

In this case, the EditField is named "SelectedURL" and the EditField property that we want is its "Text" property. This means the following expression accesses the contents of the EditField:

```
SelectedURL.Text
```

That is, "SelectedURL" is the name of the object and "Text" is the name of the object's property that we need.

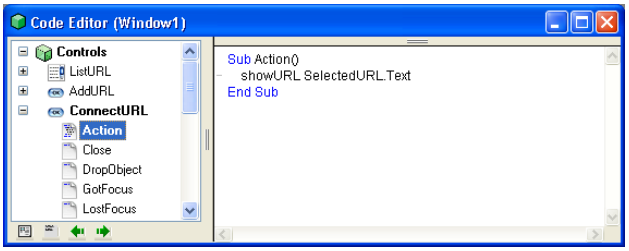
1 In the Code Editor for the Action event, modify the code to read:

```
ShowURL SelectedURL.Text
```

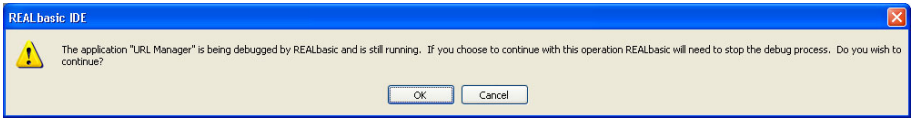
This expression SelectedURL.Text refers to the text property only.

Your Code Editor should now look like this:

Figure 23. The revised code for the Action event.



Note If you have trouble entering this line of text, be sure you have quit out of the test application before returning to the REALbasic Development environment. If you try to add code while your test application is running, you will see a message like this:



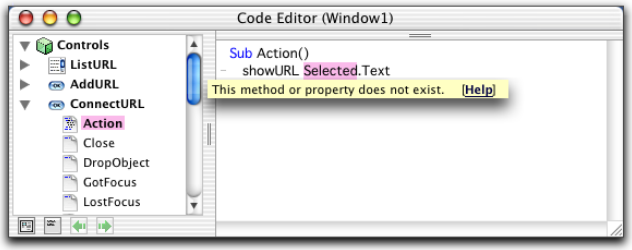
Simply click OK to quit the test application and return to the Development environment.

2 Save the project (File ▶ Save) and switch over to the Runtime environment (Debug ▶ Run).

3 Enter a URL in the EditField, such as "www.realsoftware.com" and click Connect. Your default web browser should launch and open the web page you entered.

4 When you're finished, quit out of the Runtime environment (File ▶ Exit or REALbasic ▶ Quit) and go back to the Development environment.

Note If REALbasic was unable to switch to the Runtime environment, its because it couldn't recognize a term you entered into the Code Editor. For example, if you misspelled either "ShowURL" or "SelectedURL," REALbasic stops and points out the term it doesn't recognize. For example, in the following illustration a user has misspelled the name of the EditField.



Since it can't find an object called "Selected," it can't create the application for you. REALbasic knows it would never be able to figure out what to do when a user clicks the Connect button. Be sure you've renamed the controls as described and referred to their correct names in the Code Editor.

Now, we'll make the other controls do their jobs.

The Add Button

The Add button is supposed to take the text in the EditField and add it to the end of the list in the ListBox. That's easy.

1 If the Code Editor for the window is not already open, double-click the Add button in the window (If the Window Editor is not open, double-click its name in the Project Window).

2 Enter the following code into the Add button's Action event:

```
ListURL.Addrow SelectedURL.Text
```

The first part of the expression, ListURL.Addrow calls a built-in method belonging to a ListBox. The AddRow method is a command that adds a row of text to the end of whatever list is already in the ListBox. Not surprisingly, it needs to be passed the text of the new item. We already know that SelectedURL.Text refers to the contents of the EditField, so that is what we use.

Note A *method* is a command that performs an action. Technically, ShowURL is a *global method* because it isn't attached to any particular object. It can be called by any object that can call a method. We just happen to be calling it from a PushButton. (We could, for example, design the application so that ShowURL is called when the user chooses a menu item instead of clicking a button.)

Just as objects can have properties (like their name, size, position, and label), they can also have their own methods. AddRow is also a method but it "belongs" only to ListBoxes. It has a specialized action that only makes sense when applied to lists in ListBoxes.

The Delete Button

The Delete button removes the selected item in the ListBox. It's also pretty simple.

1. In the Code Editor, expand the DeleteURL item and highlight the Action item.
2. Enter the following line of code:

```
ListURL.RemoveRow ListURL.ListIndex
```

In this case, we are calling the built-in RemoveRow method of the ListBox. Instead of text, it needs the number of the row (line) to delete. The ListIndex property contains that number, so we pass that number to the RemoveRow method.

The ListBox

The ListBox itself has the job of copying the item the user selects into the EditField so the user can connect to that URL or email address. You can easily do this by writing an *assignment statement* that runs when the user highlights an item in the list.

1. With the Window Editor in the front, double-click on the ListBox.
   The Code Editor window changes to select the ListBox's Change event. The Change event runs whenever a different item is highlighted in the ListBox.
2. Enter the following code into the Change event:

```
SelectedURL.Text=ListURL.Text
```

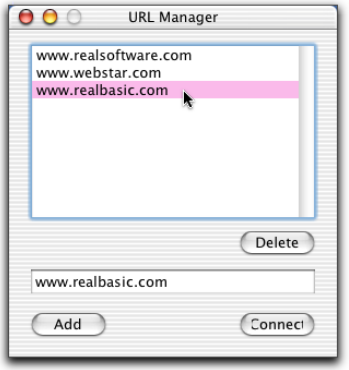
The ListBox's Text property contains the text of the highlighted item. The Text property of SelectedURL contains the text that's currently displayed. This assignment statement copies this text into the Text property of the EditField.

Testing the Application

That's the basics of this application. Now it's time to test out all these features.

1. Choose File ▶ Save to save your changes.
2. Choose Debug ▶ Run and test it out.
   The application looks the same as it did, but all the controls work! For example, enter **http://www.realsoftware.com** into the EditField and click the Add button. Add a few other URLs to the ListBox in this manner and then highlight one in the ListBox to move it to the EditField.

Figure 24. Clicking a URL to move it to the EditField.



You can:

- Enter a URL into the EditField and connect to the site using your default web browser.
- Add the URL to the ListBox.
- Select any URL in the ListBox to copy it into the EditField.
- Delete the selected URL in the ListBox.

If you want to send an email, enter it in the following way:

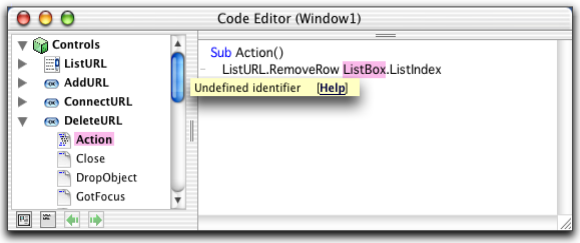
```
mailto: username@domain.com
```

If the Application doesn't run

Like any development environment, REALbasic's language has a vocabulary and syntax. If you misspell a term or use incorrect syntax, you won't be able to try out the application until you correct the mistake.

If something is wrong with your code, REALbasic will stop and point out the first error it finds when you choose Debug ▶ Run. For example, in Figure 25 this user has entered the name of a general class of objects ("ListBox") rather than the particular ListBox itself:

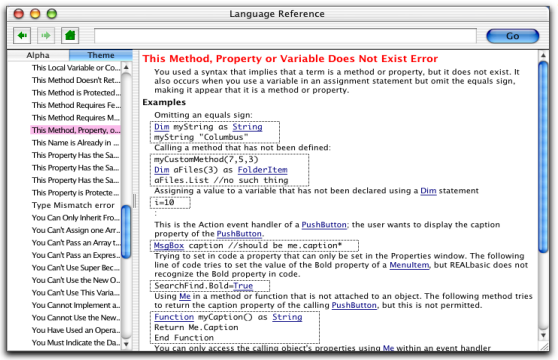
Figure 25. A vocabulary error.



The error message tells you the term that REALbasic doesn't understand; for more information about the error, you can click the "Help" button within the error

message. REALbasic will then open the Online Reference to the entry for the error. In this case, it displays the following page.

Figure 26. The error in the Online Reference.



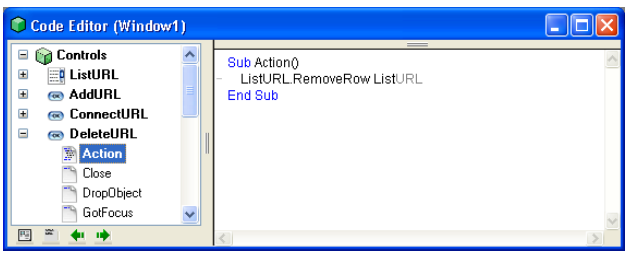
Of course, you need to correct the error before the application can run. Each time you test the application, REALbasic will stop at the first error, so there may be others.

Using Autocomplete

One way to avoid using incorrect terms is to take advantage of REALbasic's "autocomplete" feature. As you type, REALbasic tries to guess what you are typing. If you type the first few characters of a REALbasic language object — either built in or a variable, method, or property that you created — it shows its guess in light gray type. If the guess is correct, press the Tab key to complete the entry.

Here is an example:

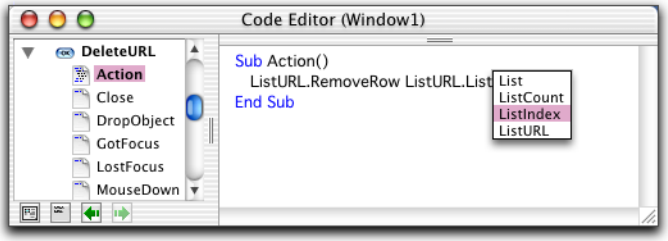
Figure 27. Autocomplete in action.



The user has just typed "List" and REALbasic has suggested "ListURL". Pressing Tab completes the word.

If REALbasic knows several alternatives, it displays them in a contextual menu when you press Tab. For example, after accepting "ListURL" and typing the period key and the first two letters of the next term, REAL basic can suggest only four possible correct matches. Press Tab when you see the ellipsis ("…") and, when the contextual menu appears, use the up or down arrow keys to select the correct term. Press the Return key to select it.

Figure 28. A contextual autocomplete menu.



How can the application be improved?

Although all the controls do their jobs, they do their jobs when it is inappropriate to do so. For example,

- If the EditField is blank, clicking the Add button adds a blank row to the ListBox and clicking the Connect button tries to open a non-existent URL.
- If no item is selected in the ListBox, the Delete button tries to delete a non-existent row.

We should make it impossible for the buttons to be clicked unless the conditions are right or, at least, give the user some appropriate feedback.

1. Choose File ▶ Exit on Windows or REALbasic ▶ Quit (Mac OS X) to return to the Development environment.
2. Double-click on the Connect button in the Window Editor to open the code for its Action event.
   It currently reads:

```
ShowURL SelectedURL.Text
```

   We need to make this conditional — so that the line is executed only when there is some text in the EditField. For now, we'll have to assume that the user knows how to enter a valid URL or email address.
3. Modify the code to read:

```
If SelectedURL.Text <> "" then
  ShowURL SelectedURL.Text
Else
  MsgBox "Please enter a URL or email address."
End if
```

   This code first tests to see whether the contents of the EditField is blank. The <> symbol means "not equal to" and the quotes with nothing in between specify blank text (Be sure to leave no spaces between the quote marks). If the EditField is blank, we use the MsgBox command to display an alert box with an informative message.
4. Expand the DeleteURL item in the Code Editor and select the Action event.

We need to follow the same logic. This code should run only if the user has selected an item in the ListBox—not all the time.

5 Modify the code to read as follows:

```
If ListURL.Text <> "" then
  ListURL.RemoveRow ListURL.ListIndex
Else
  MsgBox "Please select an item in the list."
End if
```

6 Expand the AddURL item in the Code Editor and select the Action event.

In this case, we can add an extra test. If the item is already selected, it doesn't need to be added, so we can test for that condition as well. The first test prevents the Add button from adding a blank row and the second test prevents it from adding a duplicate row (We could also write a separate method that tests whether the new item matches *any* item in the ListBox, but we will leave that as an exercise!)

7 Modify the code to read as follows:

```
If SelectedURL.Text <> ListURL.Text then
    ListURL.Addrow SelectedURL.Text
Else
  MsgBox "Please enter a different URL or email address."
End if
```

Next, we want to prevent the user from using the Add and Connect buttons if there is nothing in the EditField.

8 Expand the SelectedURL item in the Code Editor and select the TextChange event.

The TextChange event runs whenever the text in the EditField has changed. We want to take action based on the EditField's state just after the text has changed. If the EditField is now blank, we will disable these two buttons so the user can't use them. And, if there is text in the EditField, we will enable the buttons and make the Connect button the default button. The default button has an outline around it on Windows and Mac OS "classic" and it "throbs" on Mac OS X.

9 Enter the following code into the TextChange event.

```
If Me.Text <> "" then
  ConnectURL.Enabled=True
  ConnectURL.Default=True
  AddURL.Enabled=True
Else
  ConnectURL.Enabled=False
  ConnectURL.Default=False
  AddURL.Enabled=False
End if
```

One thing you notice about this code is that the first line uses the pronoun "Me" instead of the name of the control. "Me" always refers to the control the code belongs to; since we are inside the EditField, "Me" refers to the EditField. In other words, this line is the same as if we wrote If SelectedURL.Text….

The last step is to modify the code for the ListBox. This code needs to test whether the user has highlighted an item before trying to copy text into the EditField. It also should manage the Delete and Connect buttons. There's no reason the user should be able to click Delete if no item is selected.

10 Expand the ListURL item in the Code Editor and select the Change event.

11 Modify the code to read as follows:

```
If Me.Text <> "" then
  SelectedURL.Text=ListURL.Text
  ConnectURL.Enabled=True
  DeleteURL.Enabled=True
Else
  DeleteURL.Enabled=False
End if
```

12 Save the project.

13 Choose Debug ▶ Run to test it out.

When you first open the application, you can test the alert messages that you've put in each PushButton. Then add a URL and see how the application behaves.

14 Whey you're finished, return to the development environment.

Building A Standalone Application

Now that you've a finished version of the application, you're ready to create a standalone application. The *standalone* version of the application runs just like the application you've been testing, but it doesn't require the REALbasic application itself. It can be double-clicked from the desktop, just like a commercial application.

The Professional version of REALbasic can create standalone applications for both the Macintosh and Windows platforms. The Standard version allows you to build demo versions for the platform on which your copy of REALbasic is *not* running. A demo version quits automatically after five minutes. You can build fully functional applications for your platform with either the Standard or Professional versions.

Building Your Application

Building a stand-alone version of your project as an application couldn't be easier than it is in REALbasic.

1 Choose File ▶ Build Application.

REALbasic compiles your project, creates a standalone application, and brings it to the front window.

Figure 29. The URL Manager built application icon.

By default, it uses the name “My Application” or “My Application (Mac OS X)” for the Mac OS X platform). Also, REALbasic opens the window that contains the built application and brings it to the front.

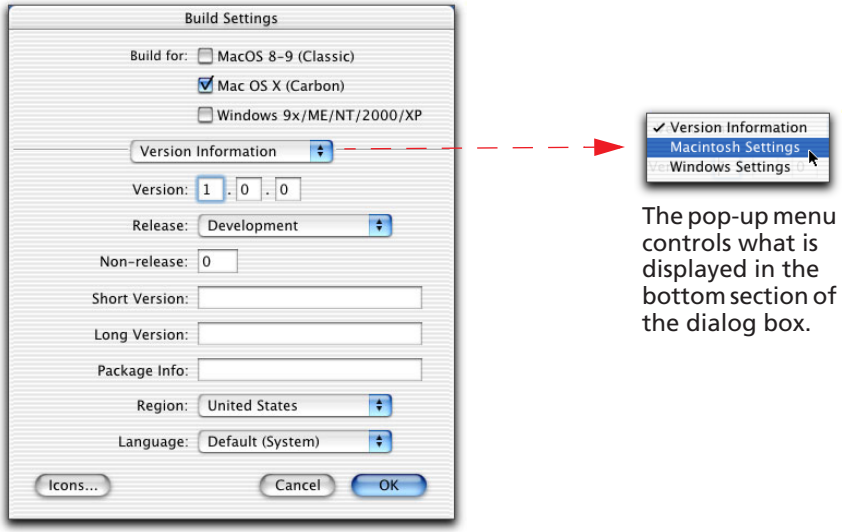
2 Double-click the “My Application” icon and try out the program. When finished, choose Quit to put away the URL Manager and go back to the REALbasic project.

Customizing the Standalone Application

Before building a standalone application, you can set several options. For example, you can change the application’s name, build for other platforms, set memory requirements for Mac OS 8-9, and some other options. You use the Build Settings dialog box for this. After choosing your options, the Build Application command uses your current Build Settings automatically.

1 Choose File ▶ Build Settings.
The Build Settings dialog box appears.

Figure 30. -The Build Settings dialog box.



The top area lists the target platforms for the build. Your choices are:

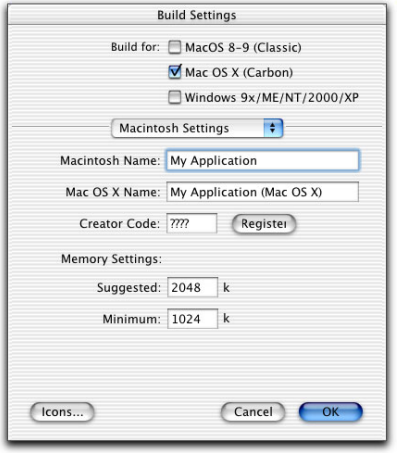
- Macintosh OS 8-9 “classic”— any version of the Mac OS prior to Mac OS X that runs REALbasic.
- Mac OS X or a “classic” version with CarbonLib installed.
- Any version of Windows from Windows 95 to XP Professional.

By default, the OS that you are currently running is selected. You can build for up to three platforms simultaneously.

2 If you are using the Windows version of REALbasic or have a Windows computer available, check the Windows checkbox.

3 Choose Macintosh Settings from the popup menu shown in Figure 30.
The Macintosh Settings panel enables you to enter the names of the Macintosh builds and set memory requirements for the "classic" build (The Mac OS X application, of course, manages memory dynamically).

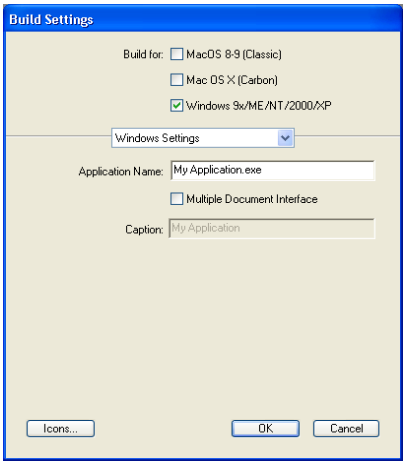
Figure 31. The Macintosh Settings panel.



4 Enter **URL Manager** as the name of the application for the version of the Macintosh OS that you're using.

5 If you're creating a version of the application for Windows, choose Windows Settings from the pop-up menu.

Figure 32. The Windows Settings panel.



6 Enter **URLManager.exe** in the Application Name area.

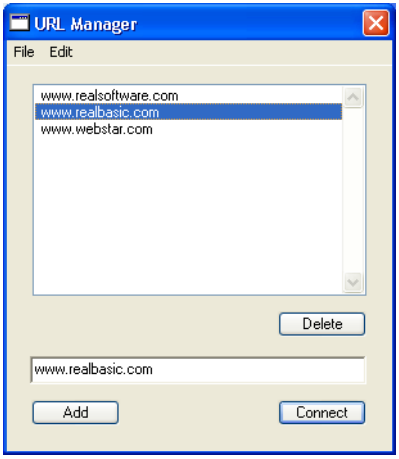
7 Click OK.

8 Choose File ▶ Build Application.

9 Quit out of the REALbasic IDE (File ▶ Exit on Windows) or REALbasic ▶ Quit REALbasic on Mac OS X).

10 Double-click the new application and try it out. For example, in Figure 33 the application is running under Windows XP™.

Figure 33. The Windows version of the URL Manager application.



What's Next

The *QuickStart* shows how easy it is to develop a simple application. In the *Tutorial*, you build a more elaborate application — a word processor that supports styled text, creating, opening, and saving files, and printing. It illustrates more REALbasic features, including sheet windows and dialog boxes, menus and menu items, writing and calling methods, creating classes and modules, and compiling platform-specific code for Windows and Macintosh builds.

With the skills you'll learn in the *Tutorial*, you'll be able to add a File menu to this application with Open and Save items that will allow you to save URL lists to disk and open saved lists. Another way to save URL lists is to use a database. REALbasic includes a database that can be used for this purpose. It is described in the User's Guide.

Also, be sure to check out the REALbasic web site at http://www.realsoftware.com for other tutorials and how-to's.