Starbuck Getting Started

VERSION 96.12.10

About this guide

This guide describes the Java development environment which Powersoft is currently referring to using the code name Starbuck. The guide assumes that you are familiar with the basic principles of using Windows, including how to:

- Start a Windows program.
- Reposition, resize, and close windows.
- Create, open, copy, and delete files and folders.
- Point, click, double-click and drag with a mouse or other pointing device.

If you are not familiar with such features, consult the Windows documentation (for example, *Introducing Microsoft Windows 95*).

This guide also assumes that you are familiar with the Java programming language.

Part I. Fundamentals

This part introduces you to programming with Starbuck.

Chapter 1. Introduction

This chapter tells you about Starbuck, how to install it, and how to use the documentation.

Description

Powersoft Starbuck is a tool for creating applications rapidly and easily:

1. You design a user interface by creating a number of *forms*. For each form, you create objects like buttons and text boxes from a standard palette of components, then arrange the objects to produce what the user will see when the program runs.
2. You set up the *properties* of the objects you have placed on each form. Properties control the behavior and appearance of the objects. For example, properties let you set the color of an object, the text used to label an object, and so on.
3. You write Java code to handle the possible *events* that may occur during program execution. For example, you might write a routine to specify what happens when the user clicks a particular button or types text into a text box. You can produce Starbuck code with a minimum of typing by using drag-and-drop code generation.
4. At any point during design and coding, you can run your program to test the interface and debug the code.

With Starbuck, you develop programs using *components* and *objects*:

- Components are *classes* in the Starbuck component library. Components may be things the user can see (such as check boxes, list boxes and buttons), or things that are internal to the program (such as timers and database queries).
- An object is an actual instance of a component.

For example, text boxes on the Component palette are represented by the TextBox component. The actual text boxes on a form are all objects.

Every object can respond to a number of events. Writing Starbuck programs is a matter of writing appropriate *event handler* routines for objects.

Starbuck and Java

You can use Starbuck to write sophisticated programs, even if you are not an experienced Java programmer. The reason is the *Starbuck component library*: a collection of Java class definitions that make it easy to work with the Windows environment.

- You don’t need to understand the low-level workings of the Java library.
- You don’t have to be able to define a Java class.
- You just have to learn a few simple techniques for using the library components that Starbuck provides.

At the same time, experienced Java programmers can use the full facilities of Java whenever necessary. The Starbuck version of Java is not a “toy” or an artificial language that can only be used with one product—it is a fully standardized implementation of Java. Code written for another Java implementation can be ported to Starbuck, just as you would to any version of Java.

Starbuck provides the best of both worlds: a simple way to produce application programs using a comprehensive component library; and the full power of a standardized programming language for those with more far-reaching requirements.

Installing Starbuck

When you install Starbuck, the setup program asks you to specify installation options, then creates folders, copies files and updates your Windows configuration.

◆ **To install Starbuck:**

1. Insert the Powersoft Starbuck CD into your CD-ROM drive.
2. If the Starbuck installation program does not start automatically, start the `Setup` application on the Powersoft Starbuck CD-ROM.
3. Follow the instructions in the installation program.

Once the installation program has finished setting up Starbuck, you should read the important last-minute information in the `ReadMe` help file.

Documentation overview

Starbuck Professional includes the following printed manuals:

- Getting Started: This manual—installing Starbuck, writing simple applications, and learning the basics of Java.
- Programmer's Guide: An explanation of how to use Starbuck to perform common and advanced programming tasks.

These manuals are also available online, to make it easy for you to look up information quickly.

Important: The online documentation contains the most up-to-date information about Starbuck. Therefore it is more authoritative than the printed version of any manual.

, Starbuck includes an extensive library of online documentation. This includes:

- Starbuck Component Library Reference, providing full details on every class in the Starbuck component library including the standard Java library that is provided with Starbuck.
- Documentation on all other parts of the Starbuck package.

The Starbuck program group and the Starbuck Master Help contents tab list the online documentation that is included in the Starbuck package.

Getting help while you work

In addition to online and printed manuals, Starbuck provides context sensitive help that explains how to use the software.

You can invoke context sensitive help in a variety of ways:

- Click the question mark button at the top right of a dialog box, then click part of the dialog box. Starbuck provides information about the part that you clicked on.
- Use the right mouse button to click on an item on the screen. If the context menu has a **What's This?** menu item, click it. Starbuck provides information about the part that you clicked on.
- Select something on the screen and press F1. This provides a description of the purpose and use of the selected item.

Sample projects

Starbuck comes with a number of sample projects that you may like to examine or experiment with. For information about the supplied samples, look for **Starbuck Samples** in the Powersoft Starbuck folder on your desktop or under the Windows **Start** menu.

Note: You will only be able to find samples under your Starbuck folder if you installed the samples. If you did not install samples originally and decide that you want to add them, run the Starbuck installation process again.

◆ **To load a sample project:**

1. In the **File** menu of the main Starbuck menu bar, click **Open Project**. This displays the **Open Project** dialog box that lets you choose the project you want to open.
2. Use the dialog box to find the main Starbuck folder.
3. Double-click the folder icon for `Samples`. This displays a new list of folders, corresponding to various sample programs.
4. Double-click one of the folder icons, making sure you double-click the icon, not the folder name. This displays the contents of the associated folder.
5. Double-click the file in this folder. (The name of this file should be the same as the name of the folder, with the extension `.WXP` added.)

When you finally double-click this project file, Starbuck loads the associated sample project. You can run this program in the usual way to see how it works. You can also examine the properties of each object on the program’s forms and examine the event handlers for these objects.

Document conventions

This Getting Started guide uses the following typographic conventions.

Typeface or symbol	**Meaning**
`Monospace type`	A monospaced font is used for code or for anything you must type. It is also used for the names of files and folders.
Bold face	Bold face is used for menus and other interface elements such as buttons and labels. It is also used for the names of components, Java classes, methods and events.
Italics	Italicized text is used to emphasize words such as new terms, the names of object properties and for text that is acting as a placeholder.
SMALL CAPITALS	Small capitals are used for the names of keys and combinations of keys, such as ENTER or SHIFT.
◆	This symbol denotes the beginning of a procedure for performing a task.
%%%	This symbol is used in beta documentation to indicate areas that are still under development. Either the appropriate documentation has not been written, or it is subject to change before the product is finalized.

Important: A paragraph placed in a box often describes an exception to general principles given in the main body of the text.

You and the user

When the documentation contains a phrase like “you click the **OK** button”, the word “you” refers to the person using Starbuck to develop a program. When the documentation contains a phrase like “the user clicks the **OK** button”, “the user” refers to the person who will use the programs you develop using Starbuck.

Using the mouse

Unless specified otherwise, you use the *left* button in all actions with the mouse. For example, if the guide tells you to click or double-click an object on the screen, you use the left mouse button. Similarly, you use the left mouse button for all drag-and-drop operations, unless the documentation explicitly says to use a different button.

Chapter 2. A first application

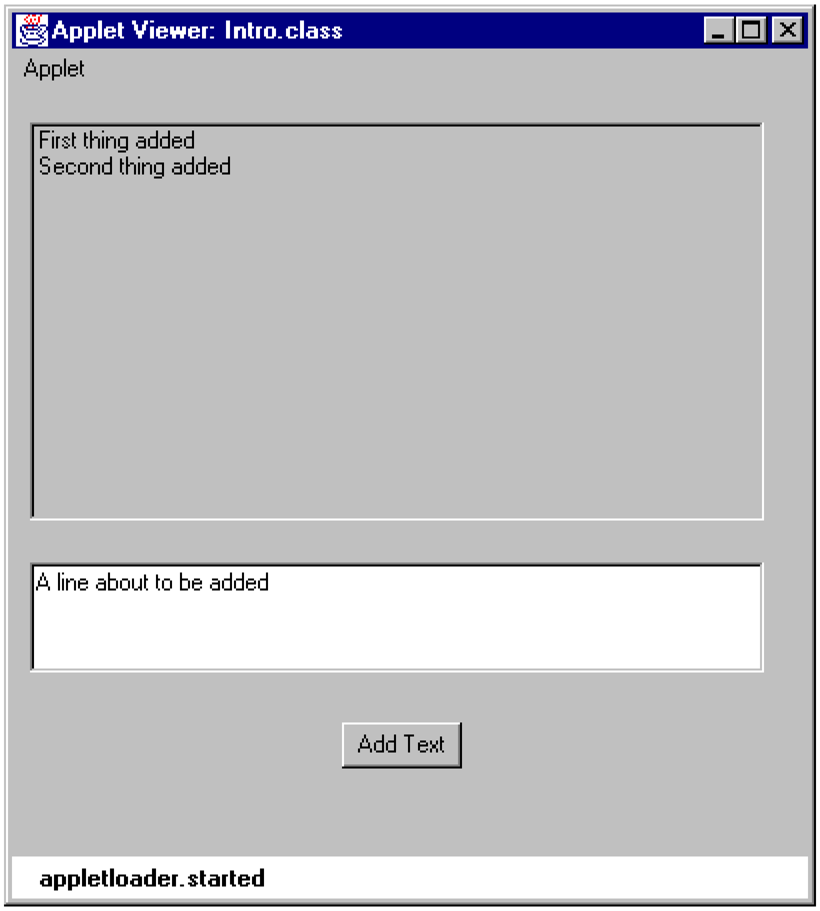
This chapter guides you through the creation of a simple application in Starbuck, showing you how to use the Starbuck design environment and drag-and-drop programming.

Steps to creating an application

With Starbuck, you create an application in three fundamental steps:

1. Design the interface.
2. Specify the properties of the objects you have designed.
3. Add code to deal with user actions.

The rest of this section guides you through this process by creating a simple Java applet, then enhancing it in several ways. The sample applet consists of a list box, a text box, and a command button. When you run the applet, you will see a form similar to the following:



When the program is running, you can type a line of text into the text box, then add that line to the list box by pushing the command button.

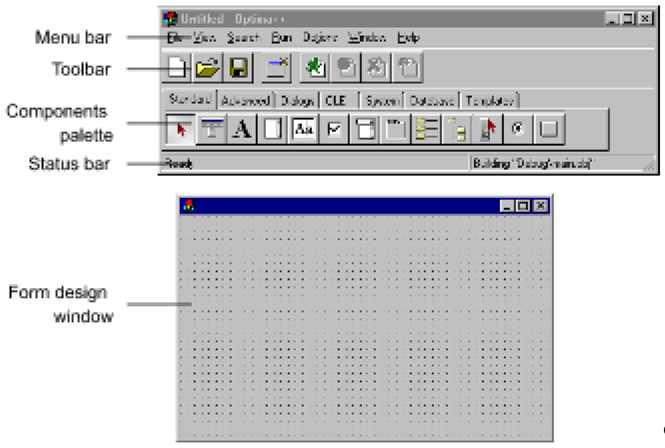
Starting Starbuck

You can start Starbuck in the same way as any other application.

◆ **Start Starbuck:**

1. In the Windows desktop, click **Start**, then point to **Programs**.
2. Point to the **Powersoft Starbuck** folder.
3. Click the **Starbuck** program.

Starbuck displays an identification screen, then its initial design environment.

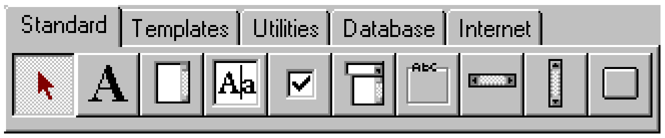


%%%

The top toolbar is the main Starbuck toolbar. Beneath that is the Java component palette. Beneath that is the *form design window*.

Designing the user interface

To design the user interface, you lay out one or more objects on the given form. The first step is to click on a type of object in the Java component palette.



Then you move the cursor to the form design area, press and hold down the left mouse button. Continue to hold down the mouse button and drag the cursor diagonally to specify the object's position and size. If you just click on the form instead of dragging, Starbuck adds a standard sized component.

Tooltips: Each button in the Java component palette has a picture that represents the type of object associated with the button. If you point to one of the buttons for a second or so, Starbuck displays the name of the component. Popup descriptions of this type are called *tooltips*.

In the following steps, you use three components from the **Standard** page of the Java component palette:

List box

Text box

Command button

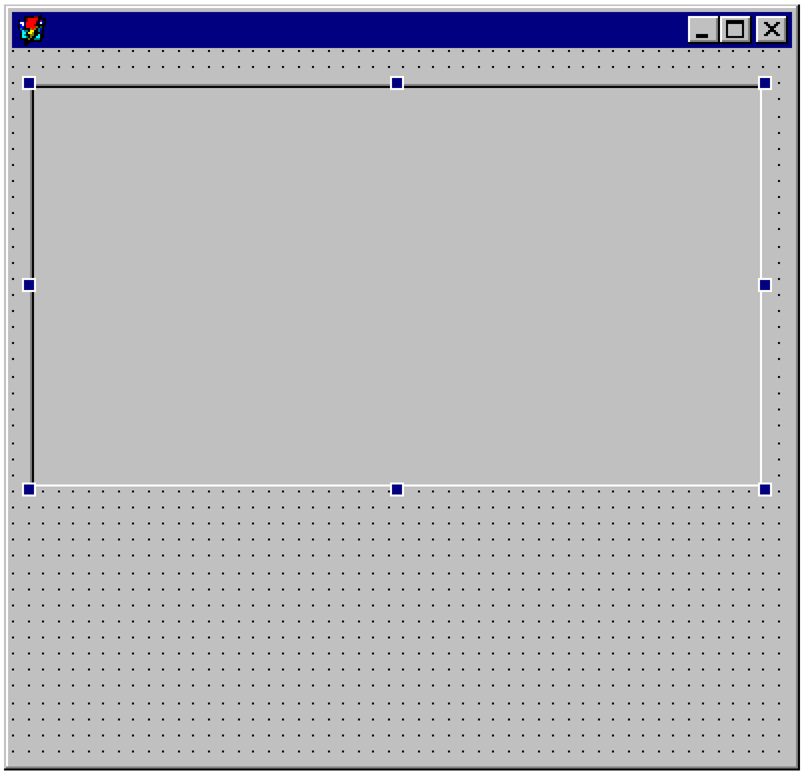
Step 1. Adding the list box

The first step in the design is to add a list box at the top of the form.

◆ **Add a list box:**

1. Click the list box button (indicated by the ListBox tooltip) on the **Standard** page of the Java component palette.
2. Move the cursor to the form design window, pointing anywhere in the upper left quarter of the window. The cursor changes from an arrow to a crosshair.
3. Hold down the left mouse button and drag the cursor diagonally across the form. While you are dragging, a rectangular outline shows you the size that the list box will be.
4. Release the mouse button.

This creates a list box on the form.



You can move the list box by dragging it. Starbuck displays the outline of the box as you drag.
You can also change the list box's size. If the list box is not selected, click the list box to select it and display resizing handles on the box's corners and edges. Then drag one of these handles to adjust the size of the list box.

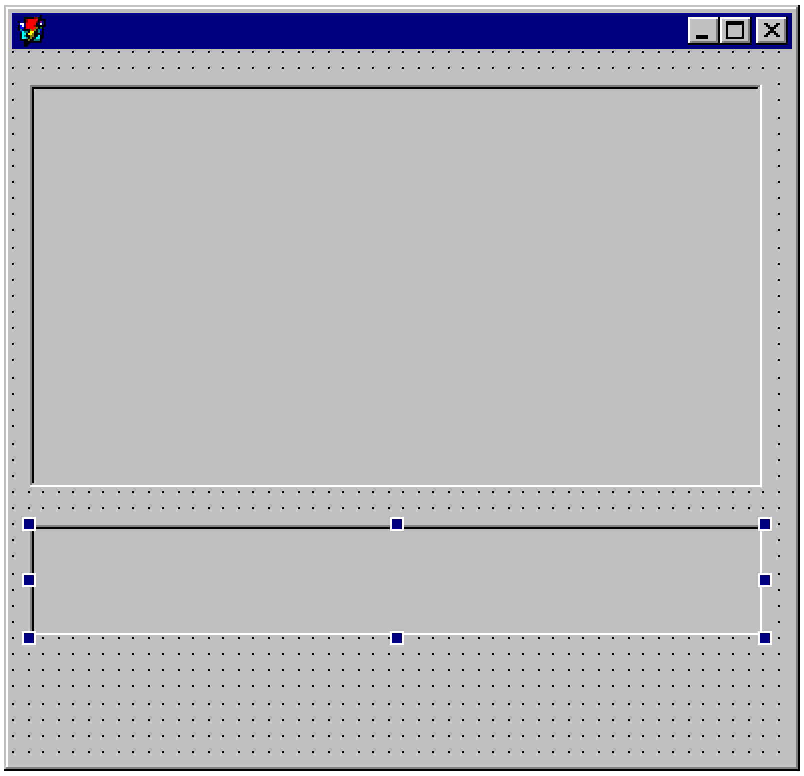
Step 2. Adding the text box

The next step in designing the form is to add a text box.

◆ **Add a text box:**

1. Click the text box button (TextBox) on the **Standard** page of the Java component palette.
2. In the form design window, drag diagonally below the list box to create a text box.

The form now has a list box and text box.



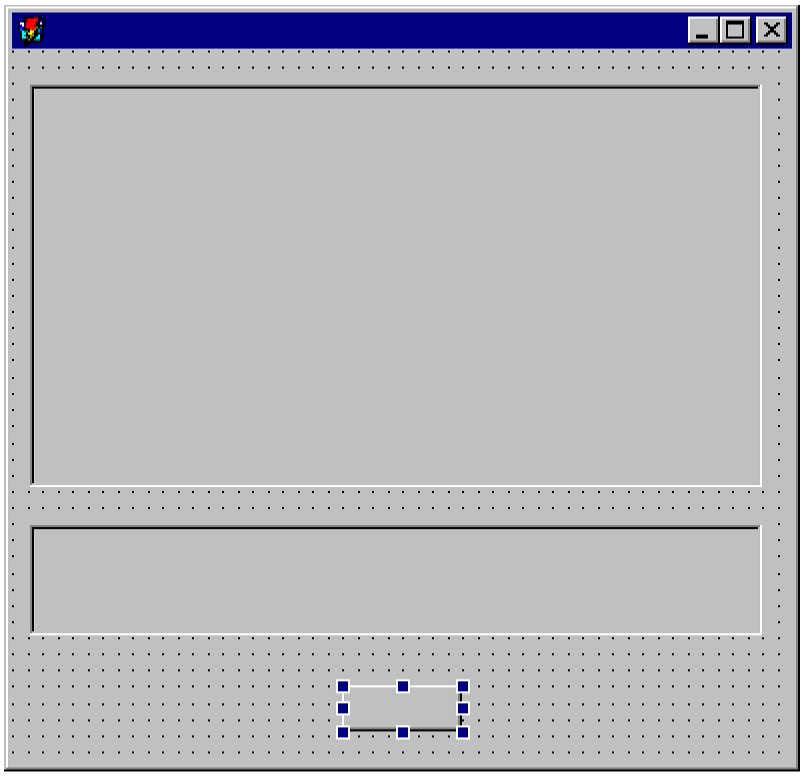
You can change the size and position of the text box in the same way as the list box. The form design window’s grid of dots makes it easier to line up objects on the form.

Step 3. Adding the command button

◆ **Add a command button:**

1. Click the command button icon (CommandButton) on the **Standard** page of the Java component palette.
2. In the form design window, drag diagonally under the text box to create a command button.

This creates a command button on the form.



Step 4. Testing

You can now run the applet to see the default behavior of the controls that you have added.

◆ **Test the interface:**

1. In the **Run** menu of the main Starbuck menu bar, click **Run**. This tells Starbuck to prepare the program for execution.
2. Wait for the applet to be compiled and executed. A new window appears containing the objects you placed on the form. This is the program that you have created.
3. Experiment with the new program. Notice that you can edit text in the text box and press the command button, but the list box does not change.
4. Close the new program by clicking the close button on the right of the title bar.

Specifying object properties

The next step in creating a program is to adjust the *properties* of the objects you placed on the form. The properties affect the appearance and behavior of the objects.

To begin with, you should put a title on the form. You do this by assigning a value to the **Text** property of the form.

◆ **Add a title to the form:**

1. Use the right mouse button to click a blank area of the form and click **Properties** to display the property sheet for the form.
2. On the **General** page, click in the **Text** box, then type

```
Sample program
```

3. Click **OK**.

Next, you should put text on the command button to explain what the button does.

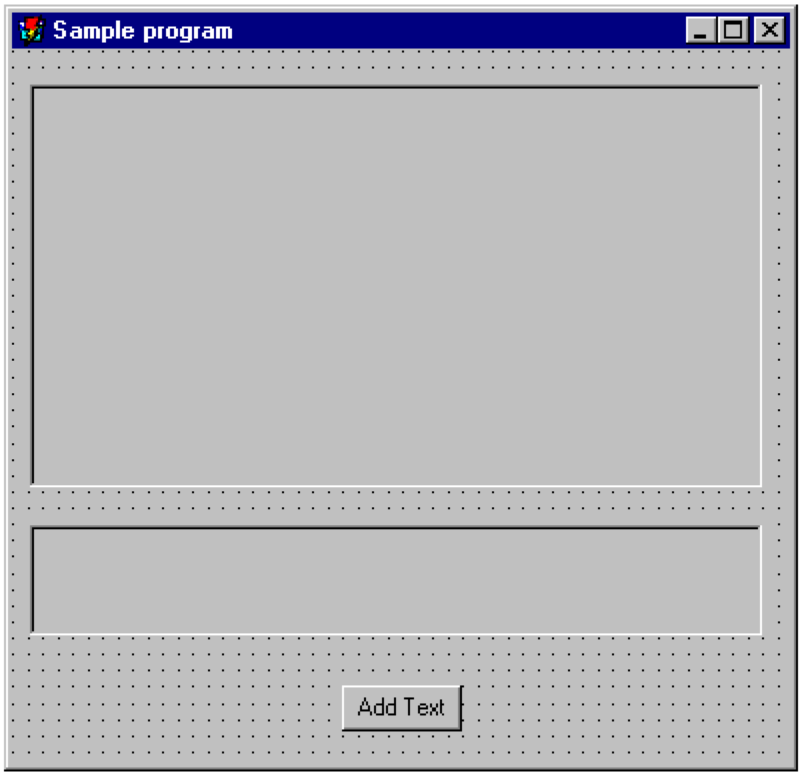
◆ **Add text to the command button:**

1. Use the right mouse button to click the command button and click **Properties** to display the property sheet for the command button.
2. On the **General** page, click in the **Text** box, then type

```
Add Text
```

3. Click **OK**.

The command button is now labeled.



You can use similar steps to change the properties of any object on the form. For example, you could experiment with property sheets by changing the colors used to display the list box.

Adding code

The last step in creating this program is to write code that responds to user actions. For this program, you need to add the code that will be invoked when a user clicks the command button. When this happens, the program should:

- Retrieve the text from the text box.
- Add the text to the list box.

You can use the Starbuck drag-and-drop programming feature to create this code with minimal typing.

◆ **Create code to be invoked when the user clicks the command button:**

1. Use the right mouse button to click the command button and point to **Events**.
2. Click the **Click** menu item. This opens a *code editor window* for the **Click** event handler.
3. Move the code editor window to the bottom right of the screen so that you can see at least part of each object in the form design window.

You can now create the code that responds to a user clicking on the command button.

Note: By default, Starbuck starts off in "big editor" mode, which lets you edit all the source code associated with a form in a single window. Starbuck also has a "small editor" mode, where each code editor window shows a single event handler function. For a description of how to switch from "big editor" to "small editor", see The big editor vs. the small editor. However, the difference between the two modes will have no effect on the work you do in this tutorial.

❑ Drag-and-drop programming

Drag-and-drop programming

With the drag-and-drop programming feature of Starbuck, you follow these steps:

- Drag the object that you want to act on, from the form design window to the code editor window where you want to specify an action. This opens the *Reference Card* which lists all the actions you can perform on that object.

  When dragging over the code editor, the insertion point will move as you drag to show you where code will be added.
- Choose an action from the Reference Card to open the *Parameter Wizard*, which helps you express that action in Java code.
- The Parameter Wizard then pastes appropriate code into the code editor window.

Once the Reference Card is open, you can drag from it to the code editor, instead of starting with objects on the form. If you want to keep the Reference Card open, open it from the **Help** menu.

The rest of this section shows how to create the necessary Java code, using drag-and-drop programming to construct every function call. In practice, you will probably use a combination of normal typing and drag-and-drop: typing the code you find easy to type, and using drag-and-drop from the Reference Card for more complicated or unfamiliar constructs. You can also use drag-and-drop actions within the code editor to copy and move selected code.

Here is the final code that you should produce:

```
public boolean cb_1_Click(powersoft.jcm.event.ClickEvent event)
{
    java.lang.String     text;
    text = textb_1.getText();
    lb_1.add( text );

    return false;
}
```

You can use drag-and-drop to create this code with very little typing. Before you start, you should have a code editor window open to the **Click** event handler for `cb_1`. The code editor window should be positioned so that it doesn’t completely cover up the form design window.

◆ **Retrieve the text from the text box:**

1. Drag from the text box in the form design window to the blank line before the `return` statement of the **Click** event handler in the code editor window. Release the mouse button. This opens the Reference Card and displays a number of categories related to text boxes.
2. Expand the **Properties** category for TextBox, and click the **Text** property.
3. Click on the line **java.lang.String getText( )**, then click **Parameters** to open the Parameter Wizard.
4. Make sure the **Store in variable** check box is checked. Type `text` in the blank area under the check box.
5. Click **Finish** to generate code in the editor.

The **Click** event handler now has an additional variable, which is assigned a copy of the string in the text box.

◆ **Copy the text to the list box:**

1. Drag from the list box on the form design window to the blank line before the `return` statement of

the **Click** event handler in the code editor window.

2. In the Reference Card for ListBox, click the **Find** tab.
3. In step 2 of the **Find** page, type `add` and wait a moment for list to show only the entries containing "add".
4. In step 3 of the **Find** page, click **add**.
5. In step 4 of the **Find** page, click the shortest version of **add** (the first one shown).
6. Click **Parameters** to open the Parameter Wizard.
7. Use the right mouse button to click in the **java.lang.String item** text box, then click **Variables** on the context menu.
8. Click **text** (the variable holding the string) in the Variables dialog box, then click **OK**. Click **Next**.
9. Make sure the **Store in variable** check box is clear, then click **Finish** to generate code in the editor.

The code is now complete and should look like this:

```
public boolean cb_1_Click(powersoft.jcm.event.ClickEvent event)
{
    java.lang.String     text;
    text = textb_1.getText();
    lb_1.add( text );

    return false;
}
```

If there are any differences, change your code to match the above sample.

Running the application

Now you should test the code that you have created.

- **Run the program:**

1. In the **Run** menu of the code editor window, click **Run** to start the program.
2. Wait for the applet to be compiled and started.
3. Type text in the text box, then click the command button. Notice that the text is added to the list box.
4. Modify the text in the text box, then click the command button. The modified text is added to the list box.
5. Experiment further. Close the program when you are finished.

You have now created a simple Java applet using Starbuck.

Saving your project

Every *Starbuck project* is a group of files which go together to make a software application. This includes one or more *targets*, and the source files used to create each target.

A target may be a Java applet, a web application, a standalone Java application, a Java library, or a DLL that can be used with a web browser (Internet Explorer or Netscape Navigator). Source files can be Java source code files, HTML files, and library files. For more information on the kinds of files that may be associated with a project, see Using targets and projects.

You can save the project you have just created so that you can run it again or modify it sometime in the future. The rest of this section tells you how.

Important: Since there are many files associated with a single Starbuck target, you must keep each target in a separate folder. This prevents the files for one target from overwriting the files for another.

Every project has an associated *project file* which lists information about the project. Starbuck project files use the extension `.WXP`.

When you save a project, you must first name a folder for the project and then name the project file.

◆ **Save your project:**

1. On the **File** menu of the main Starbuck menu bar, click **Save Project**. This displays the contents of the `Projects` folder in your Starbuck folder.
2. Under **Folder name**, type the name `Test1` and click **Save**. This creates a folder, called `Test1`, for the project.
3. Under **File name**, leave `Test1.wxp` as the project file name.
4. Click **Save**.

Starbuck creates a project file named `Test1.wxp`, in the new folder named `Test1`, along with other files and folders of the project.

Enhancing the program

This section shows how to improve your simple program in several ways. In particular, the new version of the program will clear the text box after adding a new line to the list box. It will also check to see if the text box actually contains text; if it does not, the program does not try to add the blank line to the list box.

The new source code that will do this work will be added to the existing **Click** event handler for the command button. Here is the final code that you should produce:

```
public boolean cb_1_Click(powersoft.jcm.event.ClickEvent event)
{
    java.lang.String text;
    int textLength;
    textLength = textb_1.getTextLength( 0 );
    if ( textLength > 0 ) {
        text = textb_1.getText();
        lb_1.add( text );
        textb_1.setText( "" );
        textb_1.setFocus( true );
    };

    return false;
}
```

Adding the enhancements

Adding the enhancements

In the code editor window, press ENTER at least once after the existing declaration for `text`, so that you have room to add new source code.

◆ **Generate code to determine if there is text in the text box:**

1. Drag from the text box on the design form to the line in the code editor window after the declaration of `text`. This opens the Reference Card for TextBox objects.
2. In step 2 of the **Find** page, type `text`.
3. In step 3 of the **Find** page, click **TextLength**.
4. In step 4, click **int getTextLength(int)**, then click **Parameters**.
5. Type `0` (zero) under **int line**. This stands for the first line in the text box.
6. Make sure the **Store in variable** check box is checked and that `textLength` is entered in the blank area under the check box.
7. Click **Next** and then **Finish** to generate code in the editor.

The editor window now contains a declaration for `textLength` and assigns the result of **getTextLength** to `textLength`.

To avoid adding empty lines to the list, the next lines of the code should be conditional on having a string of at least one character in the text box.

◆ **Check for a non-zero string length**

1. After the **getTextLength** line, start a new line and type:

   ```
   if ( textLength > 0 ) {
   ```

2. Select the next two lines (containing **getText** and **add**). On the **Edit** menu, click **Indent** to indent the lines.

The code in the editor window should look like this:

```
public boolean cb_1_Click(powersoft.jcm.event.ClickEvent event)
{
    int textLength;
    java.lang.String     text;
    textLength = textb_1.getTextLength( 0 );
    if ( textLength > 0 ) {
        text = textb_1.getText();
        lb_1.add( text );

    return false;
}
```

The `if` statement checks whether the length is greater than zero. If it is, your code can obtain the text that the user typed.

◆ **Clear the text box:**

1. Drag from the text box to the blank line after the **add** statement in the code editor window. This opens the Reference Card for TextBox.
2. In step 2 of the **Find** page for TextBox, type `text` (if it is not already there) and click **Text** for step 3.

3. In step 4, click **void setText(java.lang.String)**, then click **Parameters** to open the Parameter Wizard.
4. Type two double quotes (`""`) in **java.lang.String text** to specify an empty string.
5. Click **Finish** to generate code in the editor.

The editor now has code to clear the text box. Note that it is at the same level of indentation as the preceding code.

◆ **Return the focus to the text box:**

1. In the **Help** menu, click **Reference Card**. In step 2 of the **Find** page for TextBox, type `focus`. In step 3, click **Focus** and then double-click **void setFocus(boolean)** in step 4.
2. Click the arrow of the **ObjectPrefix** dropdown list, then click **textb_1**.
3. Under **boolean focus**, type **true**.
4. Click **Finish** to generate code in the editor.

The editor now has code to set the focus to the text box.

◆ **Complete the code:**

1. In the code editor, type a closing curly brace (`}`), then start a new line.

Here is the final code:

```
public boolean cb_1_Click(powersoft.jcm.event.ClickEvent event)
{
    int textLength;
    java.lang.String text;
    textLength = textb_1.getTextLength( 0 );
    if ( textLength > 0 ) {
        text = textb_1.getText();
        lb_1.add( text );
        textb_1.setText( "" );
        textb_1.setFocus( true );
    }

    return false;
}
```

Run the program. Notice the change in input focus after you click the command button. Click the command button when the text box is empty to see that the program does not try to add the blank line to the list box.

Debugging the application

This section examines how to use the debugging features of Starbuck with the application you have just created. In particular, the section demonstrates the use of *breakpoints*. If you set a breakpoint on a statement in your program's source code, program execution stops when it reaches that statement. While the program is stopped at the breakpoint, you can perform a variety of operations to examine the current state of your program.

Set a breakpoint

◆ **Set a breakpoint in your code:**

1. Use the right mouse button to click on the command button, then click **Events** and click **Click**. This opens a code editor window showing the **Click** event handler.
2. Use the right mouse button to click on the line

   ```
   textLength = textb_1.getTextLength( 0 );
   ```

   and then click **Toggle Breakpoint**.

The **Toggle Breakpoint** action sets a breakpoint on a line if there isn't a breakpoint there already. If there is a breakpoint there already, **Toggle Breakpoint** removes the breakpoint.

When you set a breakpoint on a statement, Starbuck marks the statement with a red stop sign on the left side of the window.

Set run options

Once you have a breakpoint in your code, you are ready to run your applet again. The breakpoint will only work if you are running the applet with a virtual machine that supports the debugging facilities needed by Starbuck. For example, you can use Microsoft's Internet Explorer to run the applet (provided you have turned on the option that allows Internet Explorer to run Java programs).

Note: At present, debugging does not work if you are using the Sun implementation of the Java virtual machine.

By default, Starbuck runs your applet using the Applet viewer and Microsoft's implementation of the Java virtual machine. This configuration supports Starbuck debugging. If you have Internet Explorer installed on your system, you can run your applet under Internet Explorer instead of the Applet viewer.

◆ **To run your applet under Internet Explorer:**

1. From the **View** menu of the main Starbuck menu bar, click **Targets**. This opens the Targets window.
2. In the Targets window, use the right mouse button to click on the name of your target (`Test1`) and then click **Run Options**. This opens the Run options dialog.
3. On the **General** page of the Run options dialog, click **Use a web browser**.
4. Click the **Browse** button and use the resulting file dialog menu to locate Internet Explorer on your system (`Iexplore.exe`).
5. Click **OK**.

Starbuck automatically fills in the name of the HTML file requested by the Run options dialog.

Run the program to the breakpoint

Whether you are running the program under Internet Explorer or the Applet viewer, you can now run the program to see what happens when execution reaches the breakpoint.

◆ **Run the program up to the breakpoint:**

1. From the **Run** menu, click **Run**.
2. When the form appears, click the command button.

Starbuck builds the program and runs it under whatever viewer you specified. When you click the command button, the program executes the **Click** event handler and runs into the breakpoint that you set. Program execution then "freezes" so that you can examine the current state of your program.

Starbuck opens a code editor window showing the **Click** event handler. You will see a yellow arrow in the left margin, pointing to the statement where you placed the breakpoint. This shows that execution has frozen at the breakpoint.

Note: When you set a breakpoint, execution stops just *before* executing the statement where the breakpoint is set.

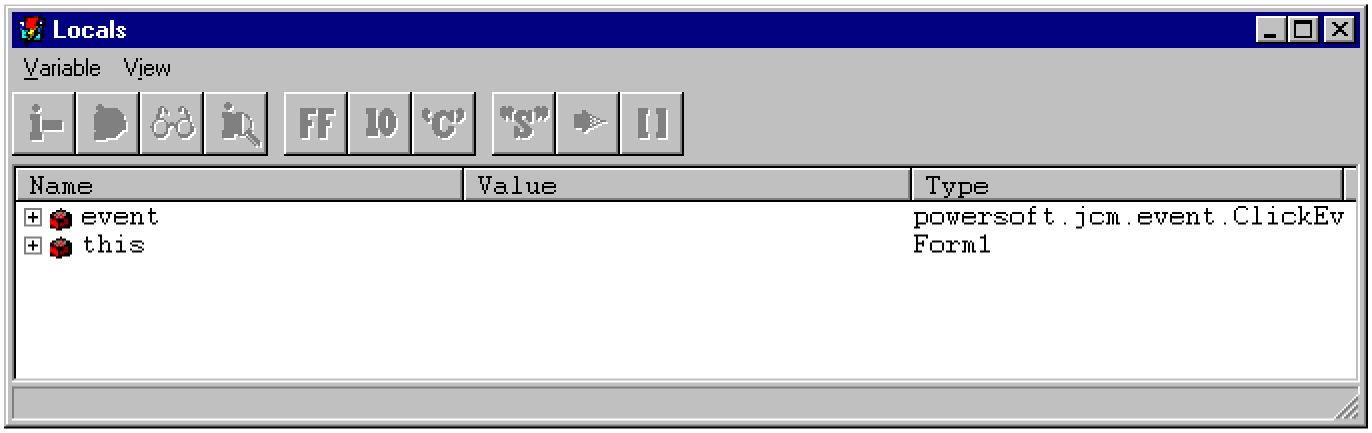
See the Locals view

When you are stopped at a breakpoint, you can examine the local variables of your program using the Locals view.

◆ **Open the Locals view:**

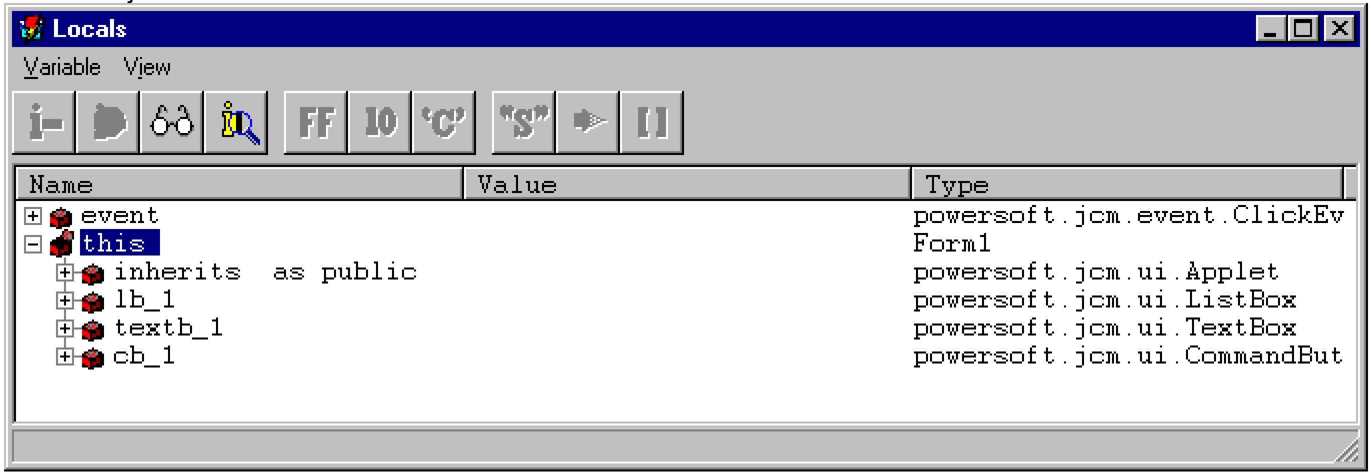
1. From the **Debug** menu of the code editor window, click **Locals**.

This displays the following window:



Name	Value	Type
⊞ event		powersoft.jcm.event.ClickEv
⊞ this		Form1

This shows two local variables defined at this point in the program: `event` referring to the argument of the **Click** event handler, and `this` referring to the form itself. If you click on the `+` sign to expand `this`, you see the objects on the form:



Name	Value	Type
⊞ event		powersoft.jcm.event.ClickEv
⊟ this		Form1
⊞ inherits as public		powersoft.jcm.ui.Applet
⊞ lb_1		powersoft.jcm.ui.ListBox
⊞ textb_1		powersoft.jcm.ui.TextBox
⊞ cb_1		powersoft.jcm.ui.CommandBut

By expanding other items in the Locals view, you can examine the contents of all the objects on the form. For example, you can see the text that is stored in the text box.

Step through the code

After stopping at a breakpoint, you can continue executing your program one statement at a time.

◆ **Step through the code:**

1. On the toolbar of the code editor window, click

You will see the yellow arrow move to the next statement in your program. This means that Starbuck has executed the statement at the breakpoint and has moved on to the next statement. If you click the same button again, execution moves on to the next statement. By repeatedly clicking the button, you can execute your program one statement at a time.

As you execute your statement, the Locals view changes to display changing values. For example, when a new value is assigned to `textLength`, the Locals view displays the new value.

Return to normal execution

You can step through your program as far as you want. When you are ready to resume normal execution again, the process is simple.

◆ **Return to normal execution:**

1. On the **Run** menu of the code editor window, click **Run**.

This starts your program running normally again, instead of the statement-by-statement operation used in the previous section.

The next time you click the button, your program executes the **Click** event handler and runs into the breakpoint again. The program will stop as before, giving you a chance to examine local data again.

Many other debugging features are available while you are stopped at a breakpoint. These are available through the **Debug** and **Run** menus of the code editor window. For more information, see Debugging.

Chapter 3. A simple database application

This chapter guides you through the creation of a simple database application in Starbuck, showing you how to use the Starbuck Form Wizard and Query Editor.

You will start a new project and create an application to view employee identifiers, first names, last names and departments from a sample employee database.

To use this tutorial, you must have Sybase SQL Anywhere 5.0 (or 5.5) and its sample database installed. This is installed when you install Starbuck, unless you choose to not install it.

Note: You cannot perform this tutorial with the preview version of Starbuck. See the ReadMe for more information on creating database applications with the preview version.

Before you begin

- Verify that Sybase SQL Anywhere 5.0 or 5.5 is installed, and that it includes the sample database. To check this, click **ODBC Administrator** in the `PowerSoft Starbuck` folder on your desktop. Make sure that "SQL Anywhere 5.0 Sample (Sybase SQL Anywhere 5.0)" is listed as a data source. Click the close button to terminate the **ODBC Administrator** program.
- Start Starbuck; if it is already running, start a new project.

Overview of creating a database application

In order to create this application, you use the Starbuck Form Wizard to create a database form. The Form Wizard automatically sets up objects on the form that will connect to a database, obtains information from that database, and displays that information. To do this, the Form Wizard asks you to enter various types of information: parameters for connecting to the database, the values you want to obtain from the database, and so on.

When you finish using the Form Wizard, you will have a form containing objects which the Form Wizard has created. Some of these objects will be visible to the user when the program runs:

- Text boxes used to display information obtained from the database.
- Labels for those text boxes.
- A *data navigator* which lets you move through the information that your program gets from the database.

Some of the objects will not be seen by the user:

- A *transaction object* providing information for connecting to the database.
- A *query object* specifying the SQL statement for fetching data from the database.

You could create your own database program by placing the same objects on a form and setting their properties appropriately. However, it is much easier to let the Form Wizard do this work for you.

The fundamental steps to creating a form connected to a database are:

1. Choose the type of form.
2. Specify the connection to the database.
3. Create a database query.
4. Choose the layout for the bound objects on the form.

The rest of this chapter guides you through this process.

Choose the type of form

You use the Form Wizard to create a new database form and specify the connection, query and layout of the form.

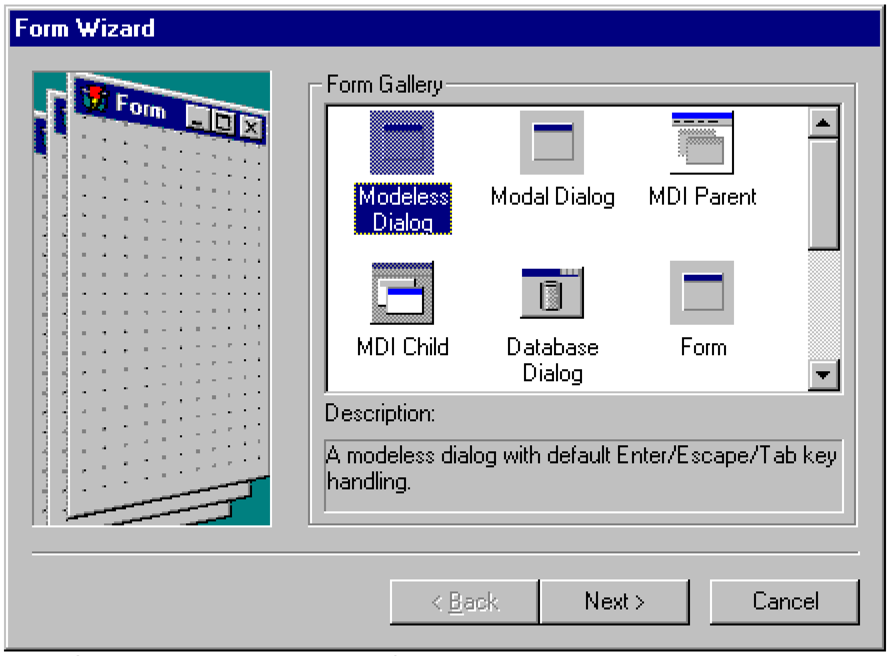
For this example, you must start with a new project.

◆ **Create a new form**

1. Click the **Form** button on the main Starbuck toolbar. The **Form** button has the icon:

Note: If you have just created a new project or started Starbuck, and you have not changed the original form, that form is deleted when you create a new form.

The Form Wizard opens and displays a gallery of forms that you can create:

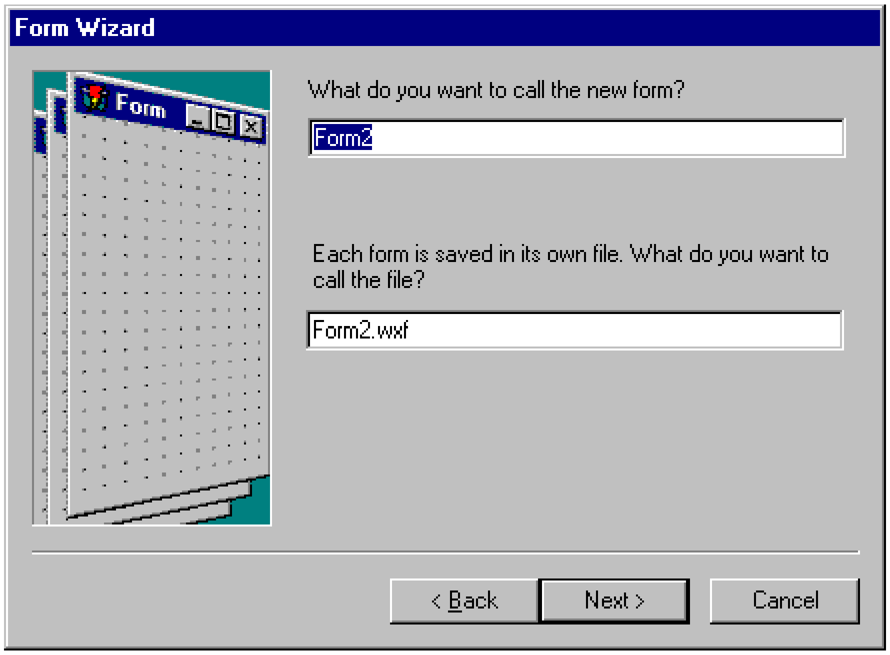


A modeless dialog with default Enter/Escape/Tab key handling.

◆ **Create a database dialog form:**

1. Click **Database Dialog** in the Form Gallery.
2. Click **Next**.

This advances the Form Wizard to the next page.


Form Wizard

What do you want to call the new form?

Form2

Each form is saved in its own file. What do you want to call the file?

Form2.wxf

The Form Wizard suggests defaults for the name of the form and the source file that will be created for it.

◆ **Name the form and its file:**

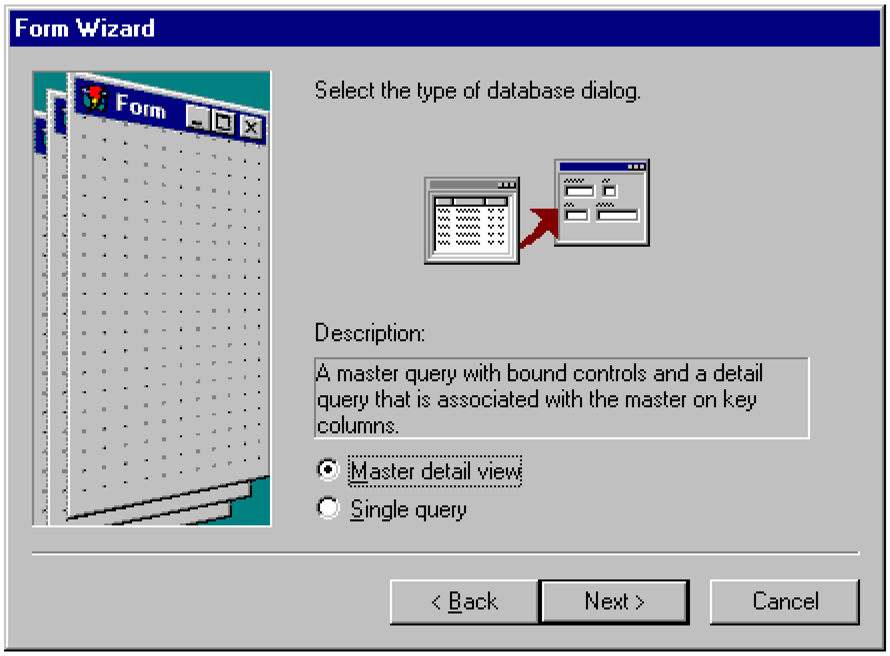
1. Click **Next** to accept the default names, or type in names of your choice.

For projects with multiple forms, use descriptive names for forms to help keep track of them.

❑ Choosing the type of database dialog

Choosing the type of database dialog

The Form Wizard now displays its next page:



Select the type of database dialog.

Description:

A master query with bound controls and a detail query that is associated with the master on key columns.

[✓ Master detail view] Single query

This offers you two types of database dialog:

- *Master detail view*: This type of form uses two queries: a master query and a detail query. For example, the master query might obtain information about a company department, and the detail query might obtain information about employees in that department. For more information, see Master detail views.
- *Single query*: This type of form makes use of a single query to obtain information from the database.

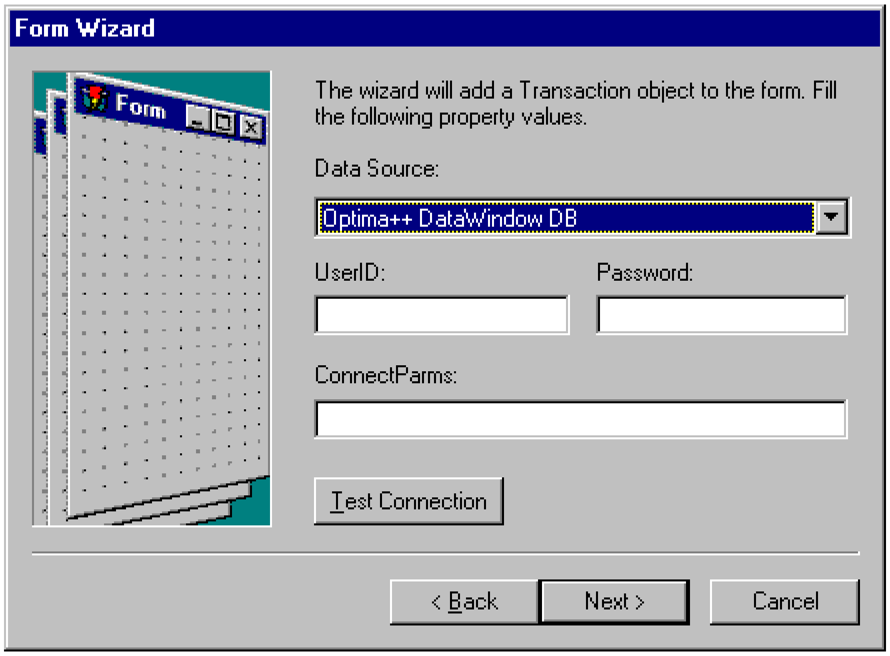
For simplicity, use a single query here.

◆ **Select the query type:**

1. Click **Single query**.
2. Click **Next**.

Enter connection information

On this page of the Form Wizard, you provide information about connecting to the database:



The wizard will add a Transaction object to the form. Fill the following property values.

Data Source: Optima++ DataWindow DB

UserID:

Password:

ConnectParms:

This information is used to specify properties for the transaction object that will appear on the final form.

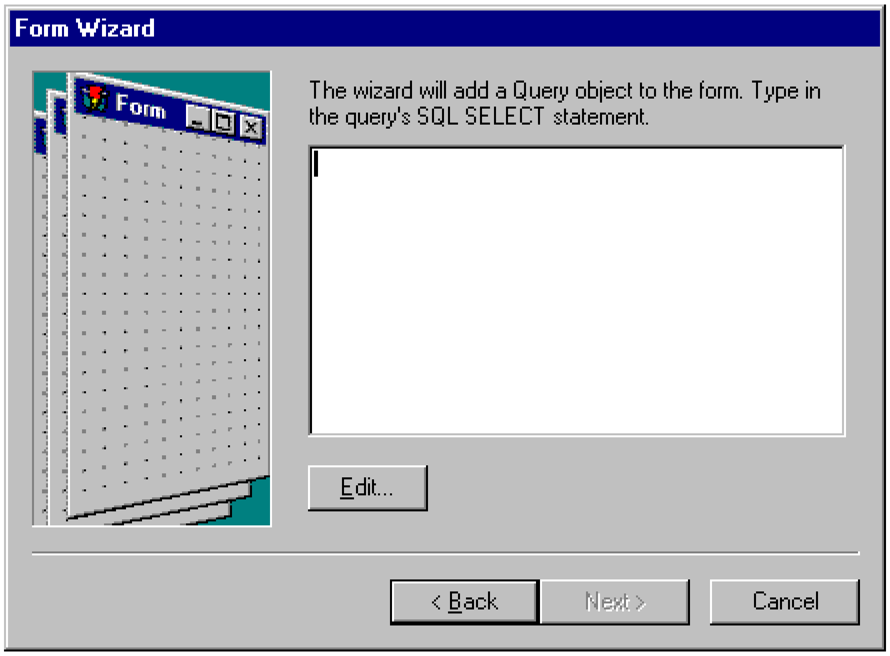
◆ **Enter connection information:**

1. Click on `SQL Anywhere 5.0 Sample` in the **Data Sources** list.
2. Type `DBA` for the **UserID**.
3. Type `SQL` for the **Password**. The Form Wizard displays `*` characters instead of the password itself, so that the password cannot be read by others.
4. Click **Next**.

If you do not specify user information at design time, you may supply this information during execution by setting the appropriate properties. If your program attempts to connect to the database without specifying a userid or password, the database may prompt the user for this information (depending on the properties you set up for the transaction and the default behavior of the database).

Create a query

On this page you enter your SQL statement to obtain information from the database:



The wizard will add a Query object to the form. Type in the query's SQL SELECT statement.

The Form Wizard lets you type the SQL statement directly, but you can also use the query editing facilities of Starbuck to create the statement.

◆ **Start creating the query:**

1. Click **Edit** to open the Query Editor.

In the process of starting the Query Editor, Starbuck connects to the database and examines its contents. Therefore, you may have to wait a while after clicking **Edit** before Starbuck is ready to proceed.

Tip: You can reduce delays if you start the database before editing your query and leave the database running while you are using Starbuck. To start the SQL Anywhere 5.0 Sample, click on **Standalone Sample Database** in the **Powersoft Starbuck** folder under the **Programs** folder of the Windows **Start** menu.

Select the tables you want to examine

The Query Editor has different pages for creating and editing a SQL statement.

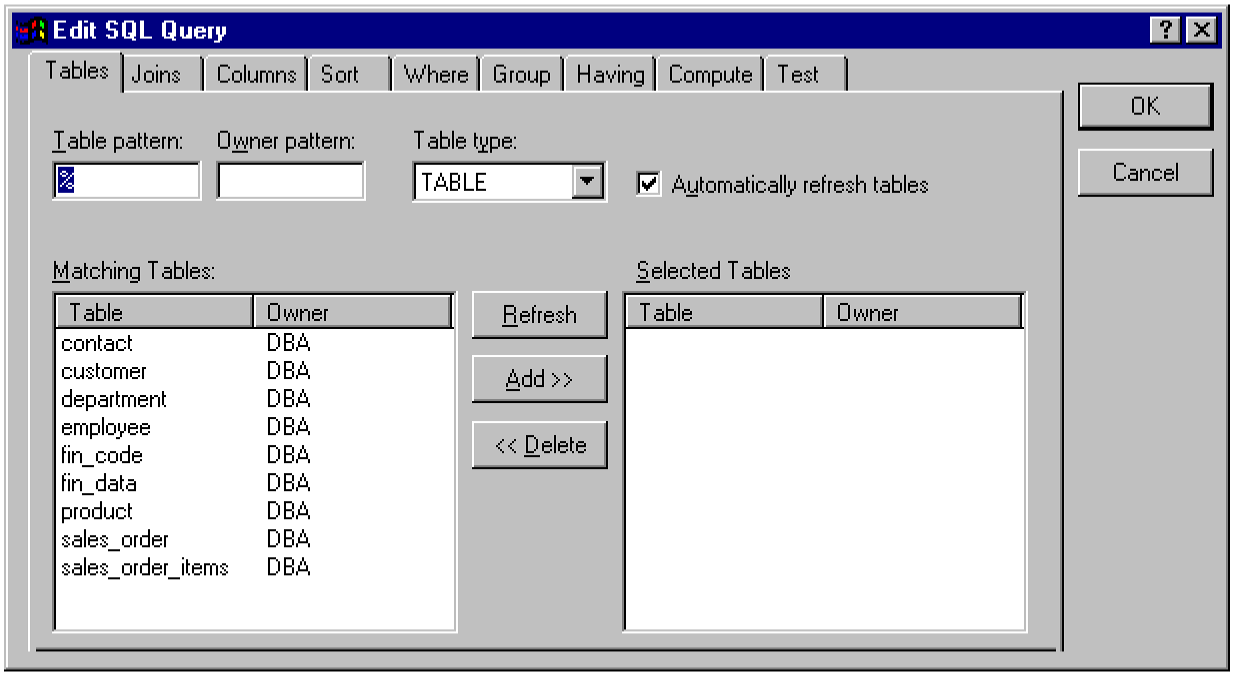


Table pattern: %

Owner pattern:

Table type: TABLE

[✓] Automatically refresh tables

Matching Tables:

Table	Owner
contact	DBA
customer	DBA
department	DBA
employee	DBA
fin_code	DBA
fin_data	DBA
product	DBA
sales_order	DBA
sales_order_items	DBA

Selected Tables

Table	Owner

In order to retrieve the employee identifier, first name, last name and department name for each employee, you need to obtain data from the **employee** table and **department** table.

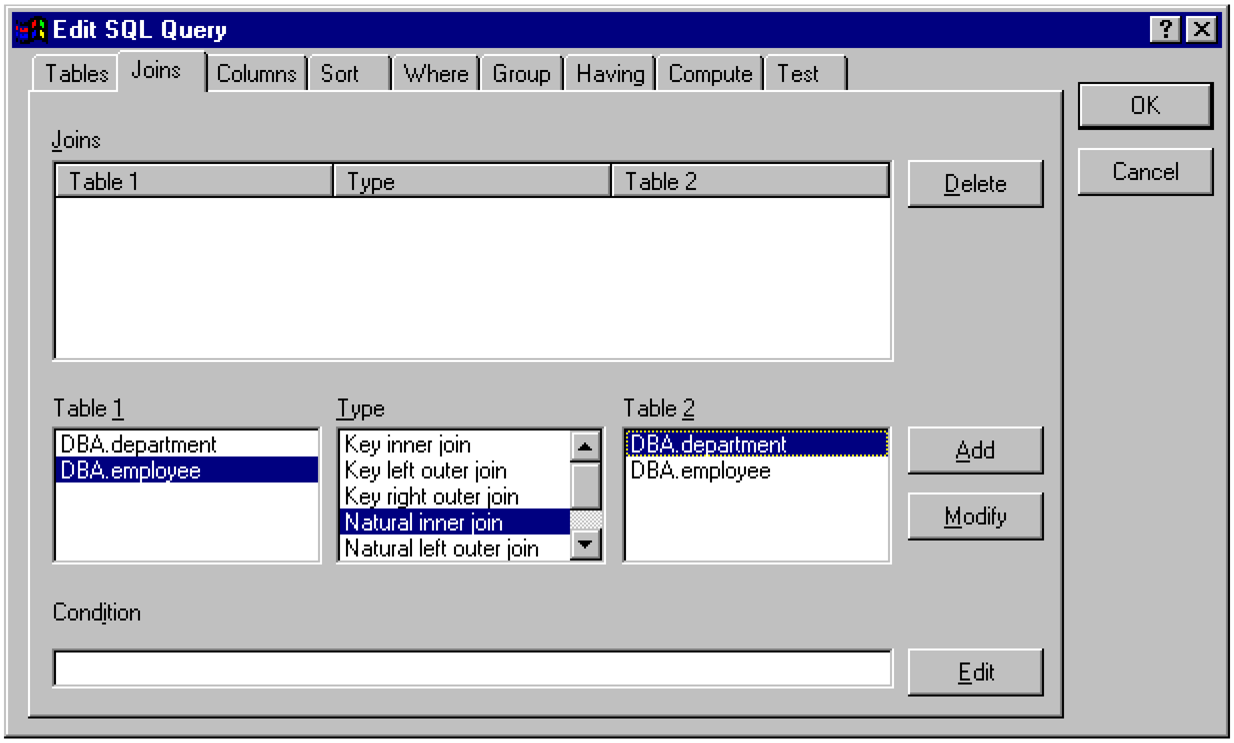
◆ **Select the tables:**

1. If the **Tables** page is not already displayed, click the **Tables** tab.
2. In the Matching Tables list, click **department**, then click **Add**. This adds **department** to the Selected Tables.
3. In the Matching Tables list, double-click **employee**. This adds **employee** to the Selected Tables.

Note: This demonstrates two different ways performing the same kind of action: selecting a table. Many of the sections that follow take the same approach, giving several ways of performing the same operation.

Specify the joins

The **Joins** page of the Query Editor controls how to combine the information from the tables you just selected:



Joins

Table 1	Type	Table 2

Table 1: DBA.department, DBA.employee (selected)

Type: Key inner join, Key left outer join, Key right outer join, Natural inner join (selected), Natural left outer join

Table 2: DBA.department (selected), DBA.employee

Condition

You need to join the information for an employee to information about the employee's department.

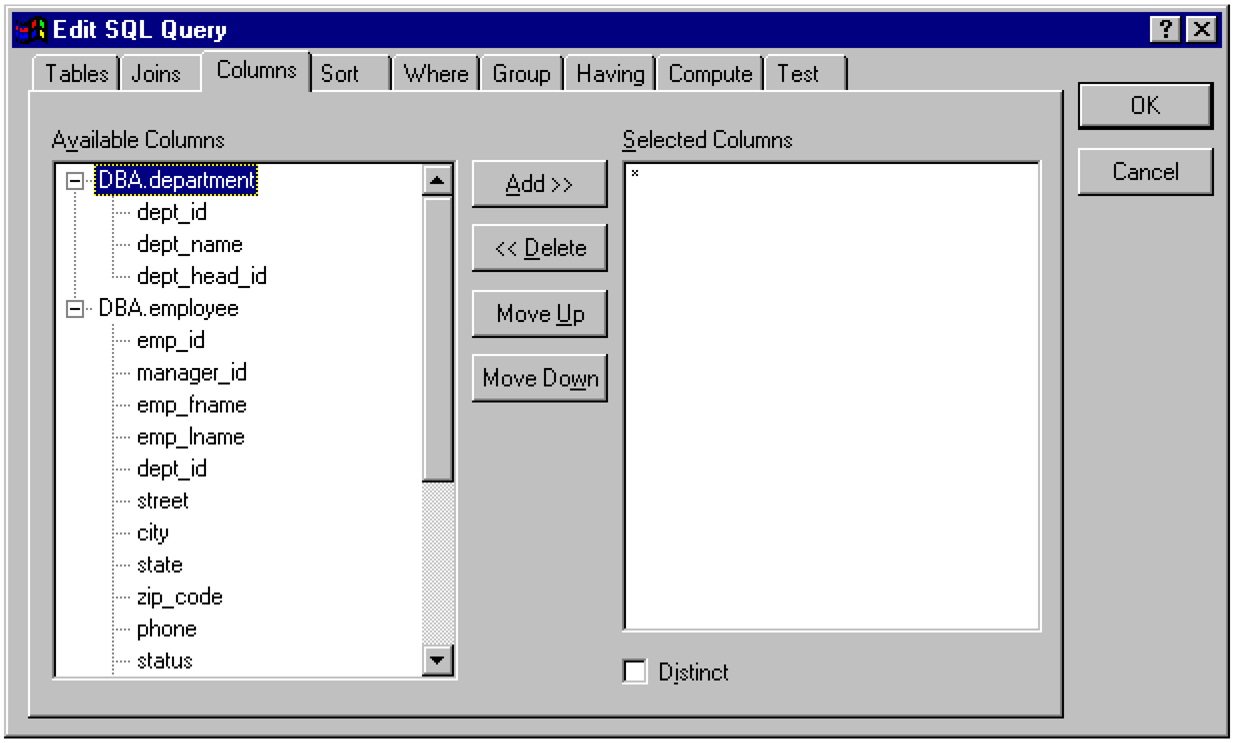
◆ **Specify the joins:**

1. Click the **Joins** tab.
2. In Table 1 of the **Joins** page, click **DBA.employee**.
3. In the Type list, click **Natural inner join**.
4. In Table 2, click **DBA.department**.
5. Click **Add**.

A *natural inner join* merges the two tables. This joins two tables based on common column names, and only includes rows where both tables have values in the matching columns. In this case, both tables have a column called dept_id, so all rows with a value for this column will be merged.

Specify the columns

The **Columns** page of the Query Editor controls which data values will actually be obtained from the database:



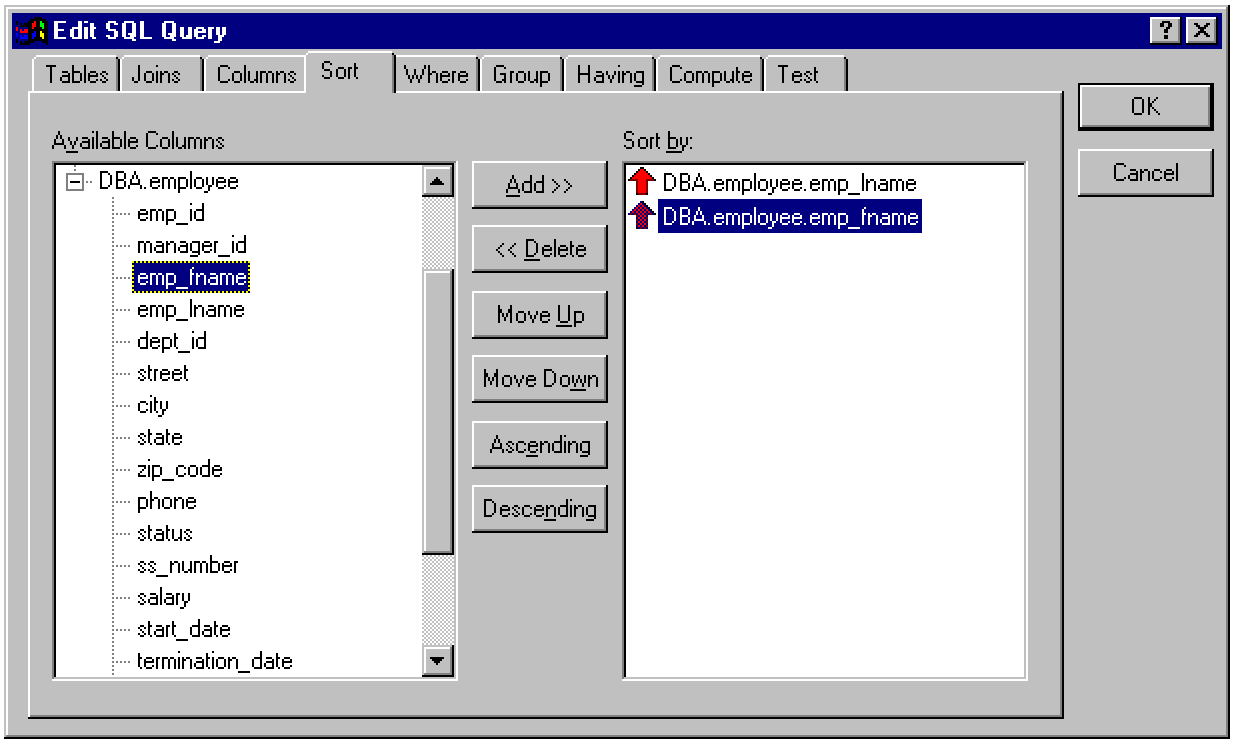
For each employee, the sample program needs an ID number, first name, last name, and department name.

- **Specify the columns:**

1. Click the **Columns** tab.
2. Expand **DBA.employee**, click **emp_id**, and then click **Add.** This creates an entry in the Selected Columns list.
3. Double-click **emp_fname** and **emp_lname**.
4. Expand **DBA.department**, and drag **dept_name** to the Selected Columns list.

Specify the sorting order

The **Sort** page of the Query Editor controls how the results of the query are sorted:



Available Columns:
- DBA.employee
  - emp_id
  - manager_id
  - emp_fname
  - emp_lname
  - dept_id
  - street
  - city
  - state
  - zip_code
  - phone
  - status
  - ss_number
  - salary
  - start_date
  - termination_date

Sort by:
- DBA.employee.emp_lname
- DBA.employee.emp_fname

The sample program will sort by employee last name, and by first name when last names are the same.

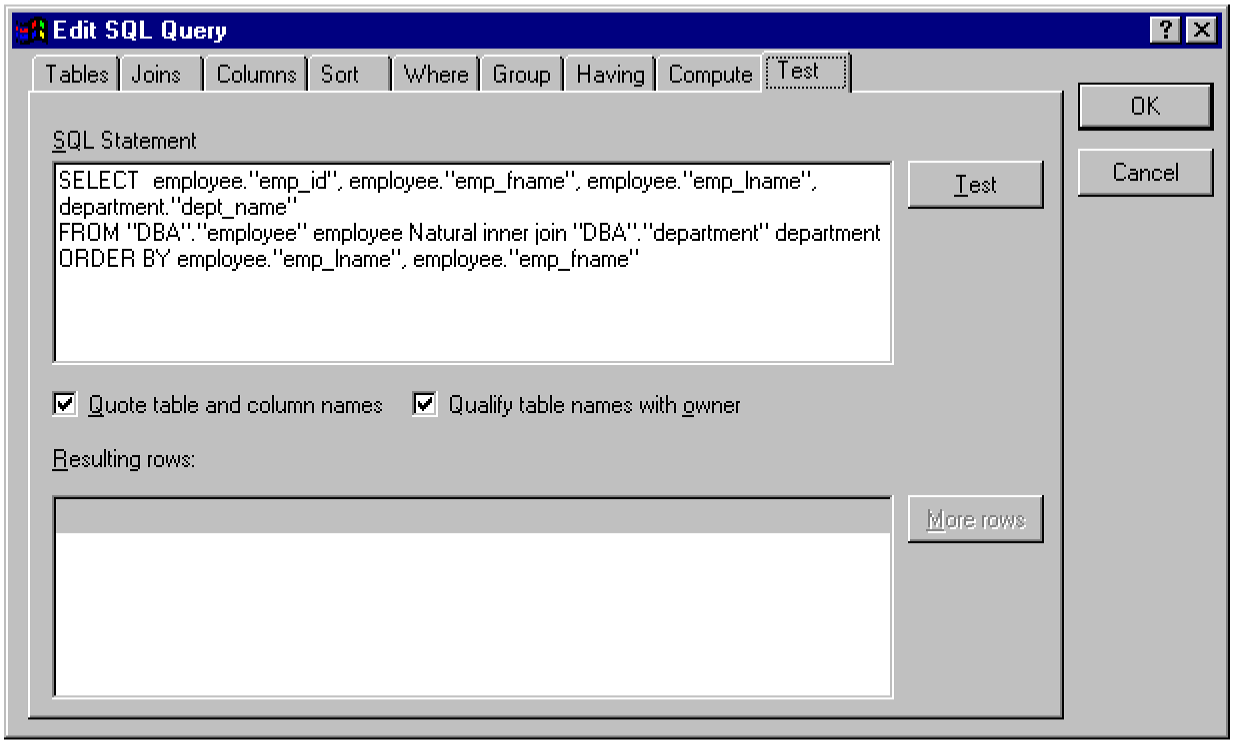
◆ **Specify the sorting order:**

1. Click the **Sort** tab.
2. Expand **DBA.employee**, and double-click **emp_lname**. This creates an entry in the "Sort by" list.
3. Click **emp_fname**, then click **Add**.

Tip: To change the sorting order for a column, double click on the arrow icon next to the column name.

Test the query

The **Test** page of the Query Editor lets you test the query you have constructed:

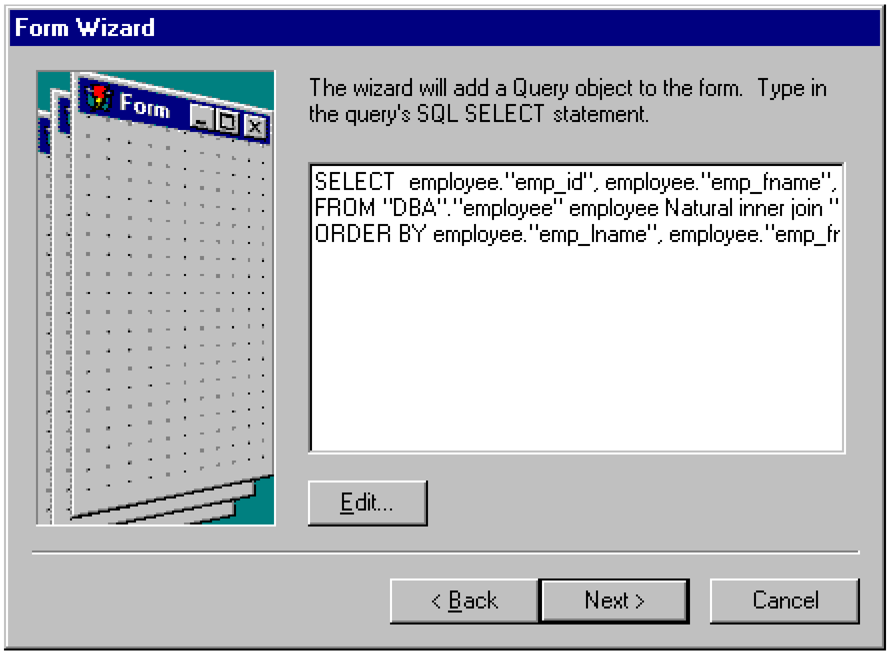


```
SELECT  employee."emp_id", employee."emp_fname", employee."emp_lname",
department."dept_name"
FROM "DBA"."employee" employee Natural inner join "DBA"."department" department
ORDER BY employee."emp_lname", employee."emp_fname"
```

☑ Quote table and column names ☑ Qualify table names with owner

◆ **Test the query:**

1. Click the **Test** tab.
2. Click the **Test** button. This executes the query to make sure it is valid. If you want to retrieve more rows, you can click **More rows**.
3. Click **OK** to return to the Form Wizard.



The wizard will add a Query object to the form. Type in the query's SQL SELECT statement.

```
SELECT  employee."emp_id", employee."emp_fname",
FROM "DBA"."employee" employee Natural inner join "
ORDER BY employee."emp_lname", employee."emp_fr
```

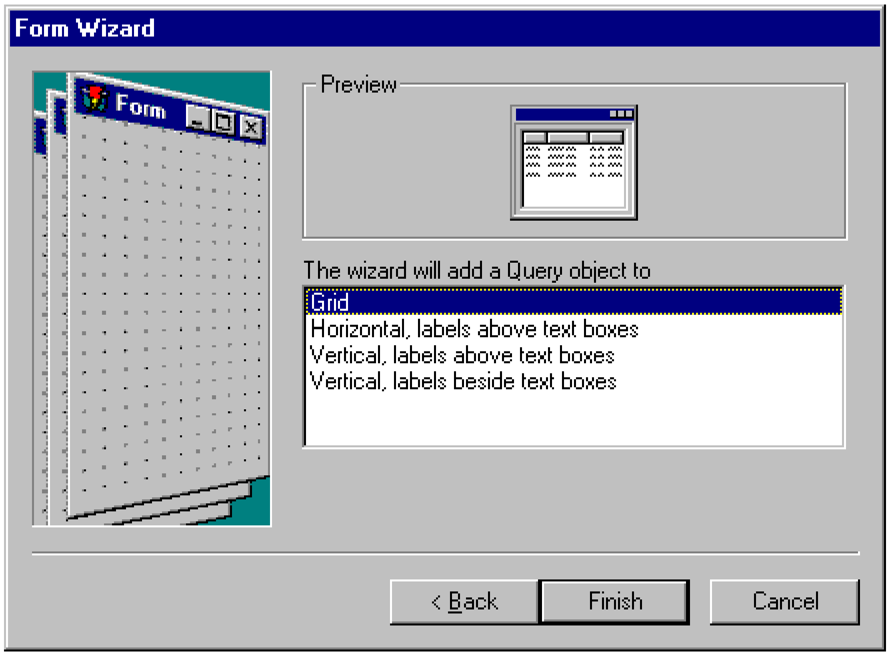
◆ **Finish the query:**

1. Click **Next**.

This advances you to the next Form Wizard page.

Select a layout for the bound objects

The last step in creating the database is choosing how to display the information you retrieve from the database:



The wizard will add a Query object to

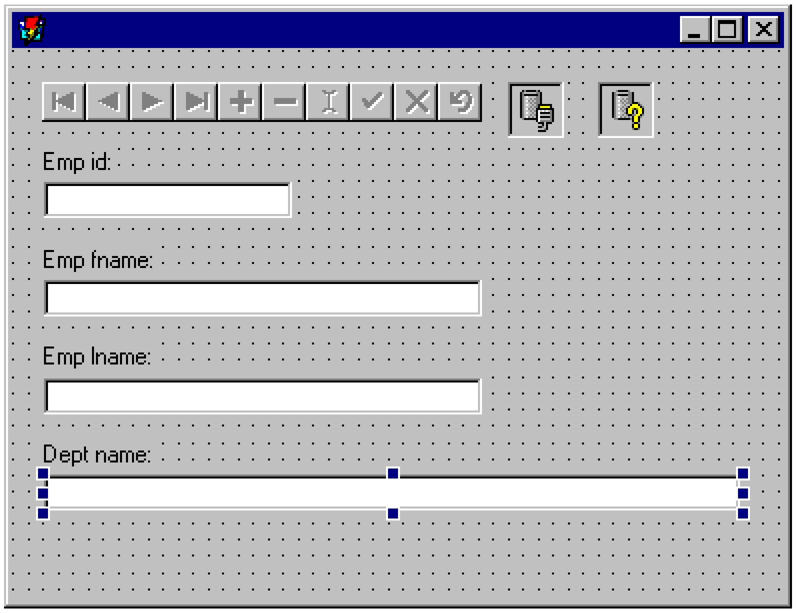
- Grid
- Horizontal, labels above text boxes
- Vertical, labels above text boxes
- Vertical, labels beside text boxes

Bound objects are objects (for example, text boxes) which display information from the database. The Form Wizard asks you to select a layout for the bound objects it creates. You can choose any format that appeals to you, but for the sake of example, we will choose a vertical arrangement, with labels above text boxes.

◆ **Select a layout for the bound objects:**

1. Click **Vertical, labels above text boxes**.
2. Click **Finish**.

The Form Wizard produces a form based on the information you have provided.



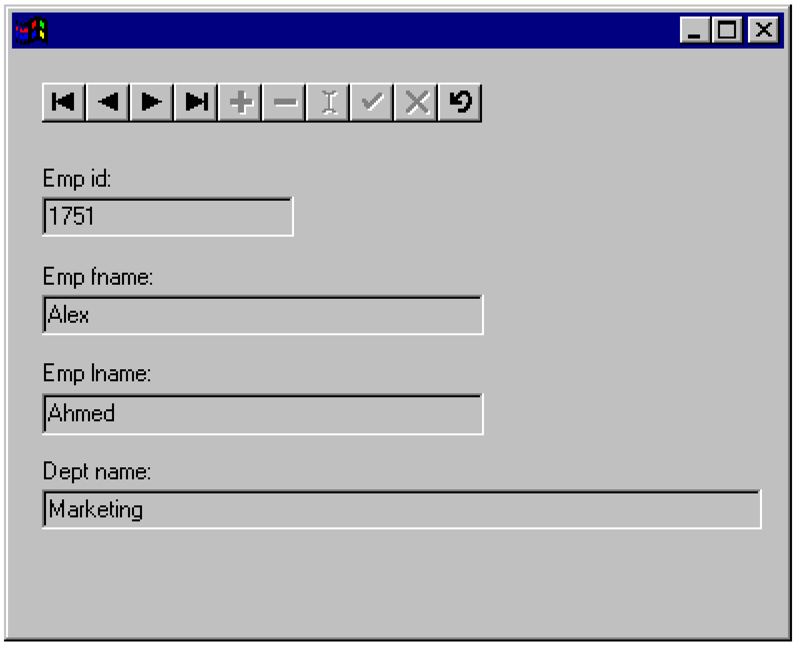
The object in the top left corner is a data navigator, used for moving forward and backward in the list of entries obtained from the database. Beside the data navigator is a transaction object containing information for connecting to the database. Beside the transaction object is a query object, specifying the SQL statement that will obtain information from the database. Below those objects are labeled text boxes that will display the information.

Run the program

At this point, you could modify the form to make it more useful to the user. For example, you could change the label **emp_fname** into **Employee's First Name** to make it easier to understand. You could turn off any unneeded data navigator buttons by using the property sheet for the data navigator. You could also add a title to the form, rearrange the layout, adjust the size or position of the text boxes, and so on. However, the program is ready to run as it is.

◆ **Run the program:**

1. On the **Run** menu, click **Run**. The program displays its database form:



Emp id: 1751

Emp fname: Alex

Emp lname: Ahmed

Dept name: Marketing

You can use the arrow buttons of the data navigator to move from one result row to the next. Click the close button on the caption bar to end the program.

Chapter 4. Tools and concepts

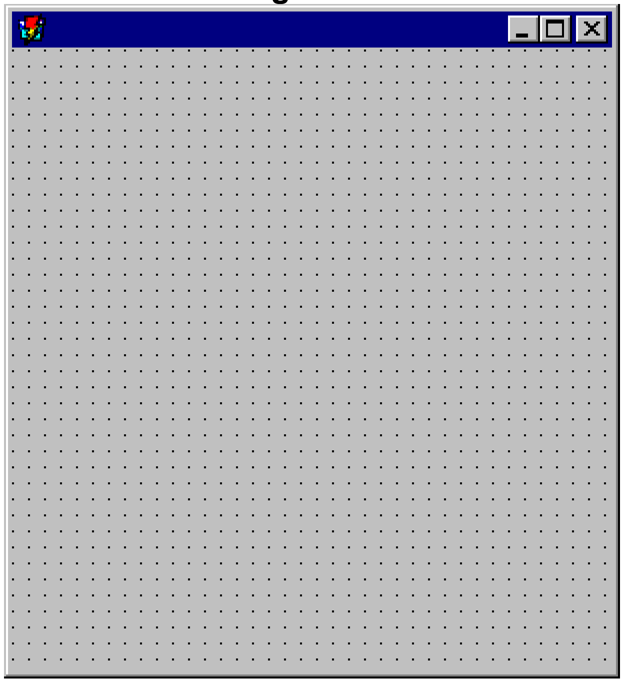
This chapter introduces you to the Starbuck programming environment and provides an overview of how Starbuck operates.

Note: There are keyboard shortcuts which provide quick access to many of the features of Starbuck. For a complete list, see “Keyboard shortcuts” in the online help for Starbuck.

Parts of the screen

When Starbuck starts, it displays a number of windows which help you design a user interface for your application.

Form design window



A *form* is a window which contains objects like command buttons and text boxes. You use forms to create your program’s main window, secondary windows, dialog boxes, and so on. A *form design window* lets you arrange objects on the form and adjust the form’s size, as you design the appearance of your program.

The grid of dots on the form design window help you judge the size and position of objects on the form. The dots do not appear when you run your program.

Main menu bar

The menu bar of the main Starbuck window offers menus whose commands help you create and test your application. When you point to a menu item, the status bar at the bottom of the Starbuck window offers a brief explanation of the item.

Toolbar

The *toolbar* appears below the main Starbuck menu bar. It offers quick mouse access to many Starbuck commands and features. You can use the **Toolbars** item of the **View** menu to control whether the toolbar is displayed or not, and to control the size of the toolbar buttons.

You can see the name of any toolbar button by pointing to the button and waiting a moment. The name of the corresponding button will appear in a small box called a *tooltip*.

You can customize the toolbar by adding or deleting buttons. Click **Toolbars** in the **View** menu, then click the **Customize** button. A dialog box appears to let you choose what buttons appear on the toolbar. If you make changes, you will see the changes in all future Starbuck sessions.

Java component palette

The *Java component palette* is a set of buttons which correspond to the objects you can place on a form.
When you first start using Starbuck, the Java component palette appears below the Starbuck toolbar.
You can see the name of any component by pointing to the button on the palette. After a moment, the name of the component will appear in a tooltip.
To add an object to a form, click a component button, then click on the form design window. This places an object of the selected type on the form.

When you place an object on a form in this way, the object is given a default size. For an explanation of how to change the object to a different size, see Sizing an object.

Status bar

The status bar appears at the bottom of the main Starbuck window.

When the mouse points to the form design window, the left part of this bar shows:

- The current position of the cursor in dialog units.
- The name of the object that the cursor is over.
- The class of the object that the cursor is over.

The right part of the status bar displays information about other actions that Starbuck performs. For example, it tells you when Starbuck initializes files for a new project.

View windows

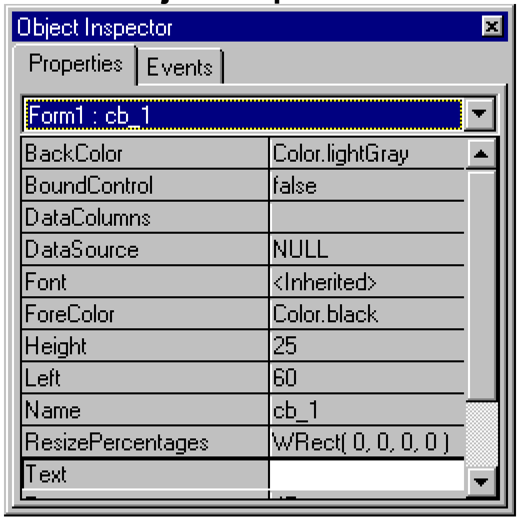
Starbuck offers a number of *view windows* to display information about your current project. For example, the *Files window* shows the files associated with your project, while the *Objects window* shows the forms, buttons, boxes, and other objects that make up your project.

You can display a view window by selecting the window's name from the main Starbuck **View** menu. When you quit your Starbuck session, Starbuck remembers which view windows you have active. The next time you return to the project, Starbuck opens the same view windows in the same positions so you can pick up where you left off.

Note: This chapter only provides an introduction to the view windows, explaining in general terms what the windows can do. The Starbuck Programmer's Guide provides specific instructions for performing the actions mentioned in these overviews.

In most view windows, if you use the right mouse button to click on an item, a menu appears giving actions that can be performed on that item. The exception to this is the Object Inspector.

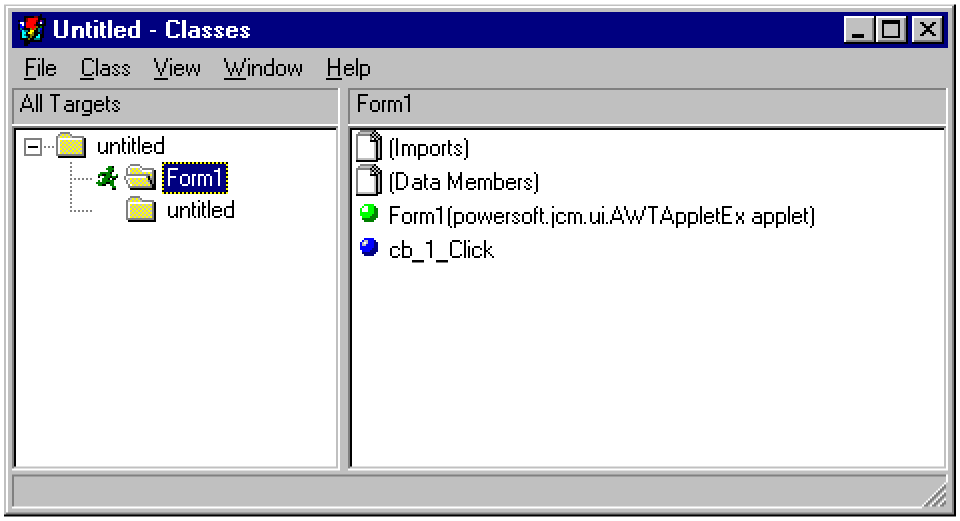
Object Inspector



BackColor	Color.lightGray
BoundControl	false
DataColumns	
DataSource	NULL
Font	<Inherited>
ForeColor	Color.black
Height	25
Left	60
Name	cb_1
ResizePercentages	WRect(0, 0, 0, 0)
Text	

The *Object Inspector* lets you view and change properties and events.
When you are viewing the **Properties** page of the Object Inspector, it shows all properties for that object. You can change the value of a property by clicking on it and then changing the value in the right pane.
The **Events** page of the Object Inspector lists all events applicable to the selected object. You can double-click an event to edit the code that will be invoked when that event occurs. If there is no event handling code for an event, the right pane for that event is blank. If there is already an event handler defined for a particular event, the Object Inspector does *not* let you specify a different function to be the event handler.
You can change the size of the columns in the Object Inspector by dragging the column separator. Double-clicking the column separator gets a default size for the left column, just big enough to show all the names.

Classes window

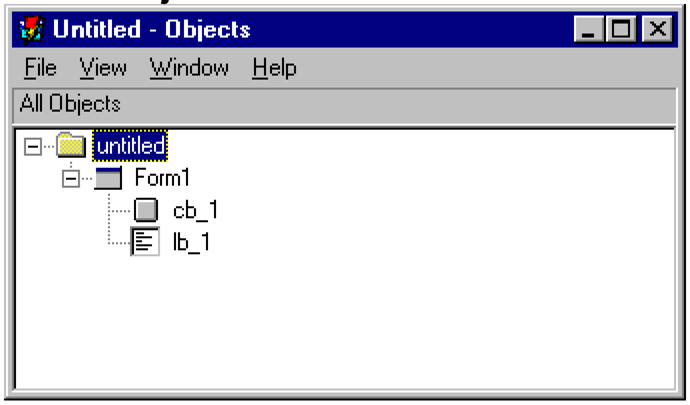


The *Classes window* lists the contents of each class that defines a form and each class that you have specifically asked to control through the Classes window. The window displays all the member functions, as well as any data objects you have defined for the form and any `import` statements required by the form's source code. For some classes, the window also displays files used in building the class. If you double-click any name in the list, Starbuck displays a code editor window where you can edit the selected function or data declarations.

The Classes window lets you add new member functions to a class, and change or delete existing functions. For more information, see Working with classes.

If you use the Classes window to create new Java classes for your target, you can examine and edit them in the same ways as for the generated form classes. Classes that you create and maintain in this way are called *managed classes*. For more information, see Adding classes to a target.

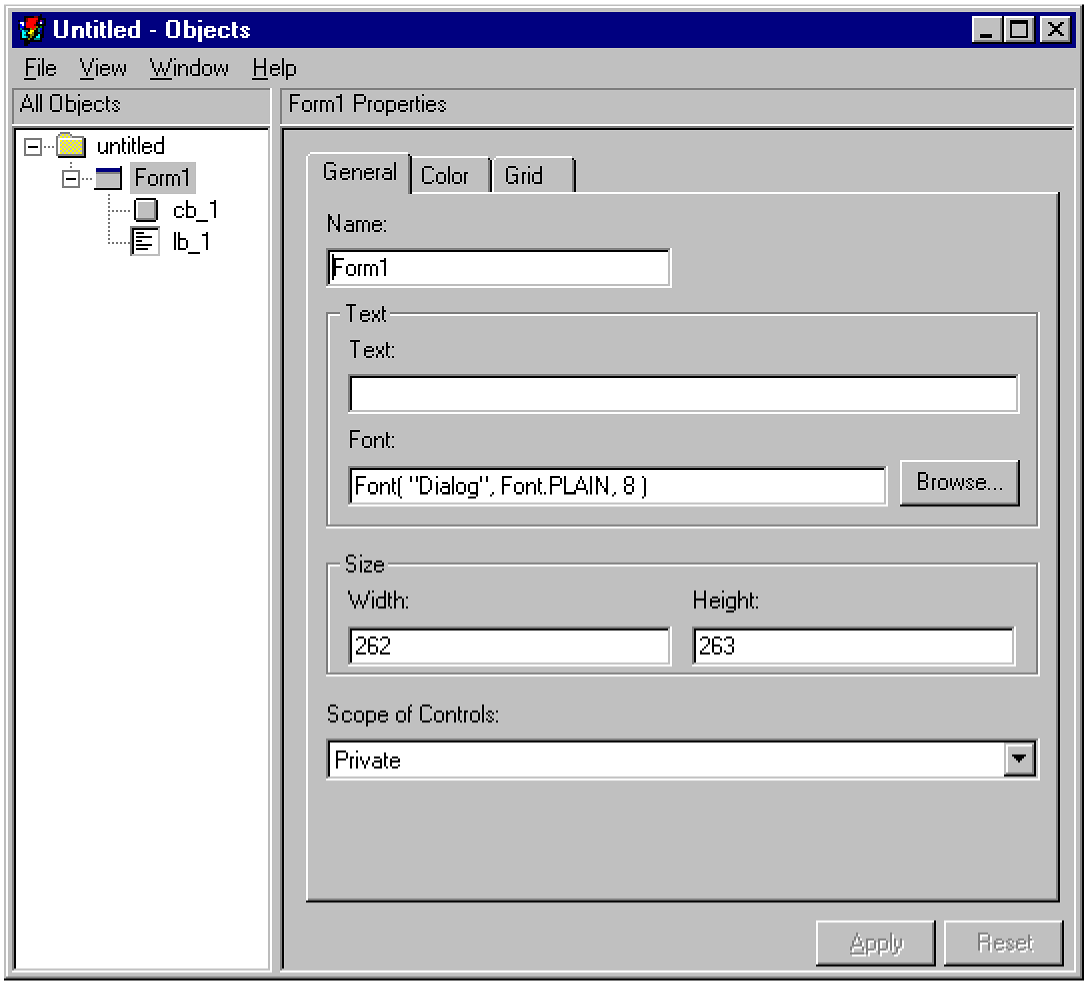
Objects window



The *Objects window* lists the objects on each form of your project, using a hierarchical display format. Targets are shown at the top of the hierarchy. You can expand the tree at a form icon to show all the objects that belong to the form.

Clicking an object with the right mouse button opens a context menu for manipulating the object. For example, this menu lets you delete an object from a form.

The Objects window has a second format, obtained by turning on **Show Property Sheet** in the Objects window **View** menu.



Name: Form1

Text

Text:

Font: Font("Dialog", Font.PLAIN, 8)

Size

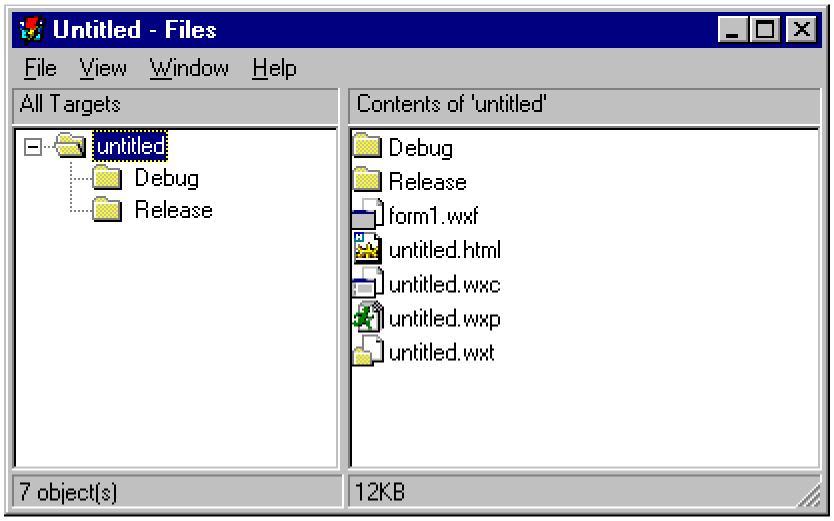
Width: 262

Height: 263

Scope of Controls: Private

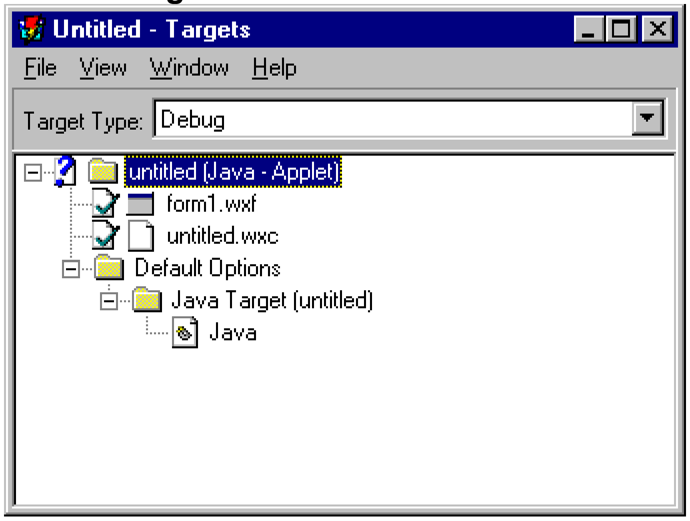
This displays a property sheet for the currently selected object. This property sheet is one way to examine or change the object's properties. For a description of other ways to work with an object's properties, see Changing an object's properties.

Files window



The *Files window* lists the files associated with your project. The window's format is similar to the Windows Explorer.
The **File** menu of this window lets you add items to your project or delete them. For example, you can import files of source code that were written for other projects.
The **View** menu changes the way that information is displayed in the window.
When you are working with a project that has many source files, you will likely use the Files window extensively. For example, you can use the Files window to add library files to your project. You can add new files to the Files window by dragging them from the Windows Explorer.

Targets window



A *target* is an application, applet, or library that you create with Starbuck. The *Targets window* displays the *target* files and the files that Starbuck will use to create them. For more about targets, see Targets.

The main purpose of the Targets window is to let you specify options for preparing Starbuck targets (for example, controlling where Starbuck searches for libraries to link with your code).

Coding tools

Starbuck includes several elements that let you create and edit Java code:

- The *Code Editor* is primarily a text editor, with automatic indenting and syntax coloring to make Java code easier to write and understand.
- The *Reference Card* provides quick information on the most important Starbuck classes and member functions.
- The *Parameter Wizard* guides you through the construction of function calls. This is particularly useful for complicated functions or for functions with which you are unfamiliar.

Drag-and-drop code generation

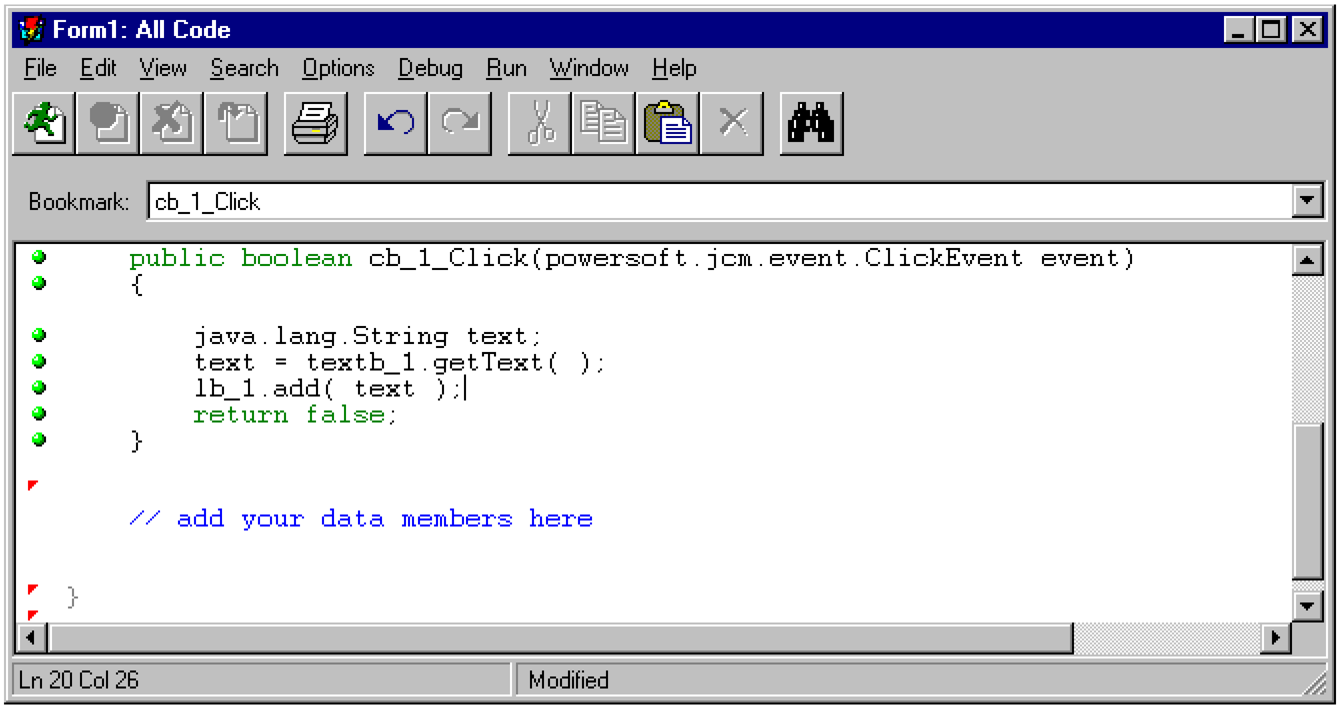
One of the most powerful features of Starbuck is drag-and-drop code generation: simply drag the cursor from an object on the form or the Objects Window to the code editor window. This opens the Reference Card, which lists all the actions you can perform on that object. If you choose an action, the Parameter Wizard helps you set the parameters for the operation. The actions provided through the Reference Card can set properties, retrieve their values, or execute member functions associated with an object.

For a step-by-step introduction to drag-and-drop programming, see Drag-and-drop programming. For a more detailed description, see Using drag-and-drop programming.

In many cases, setting up an operation with the Parameter Wizard only takes one step. Then, the wizard inserts the generated code in the code editor window, including declarations of any local variables required for the action.

When the Reference Card is already open, you can generate code by dragging the cursor directly from the Reference Card to the code editor.

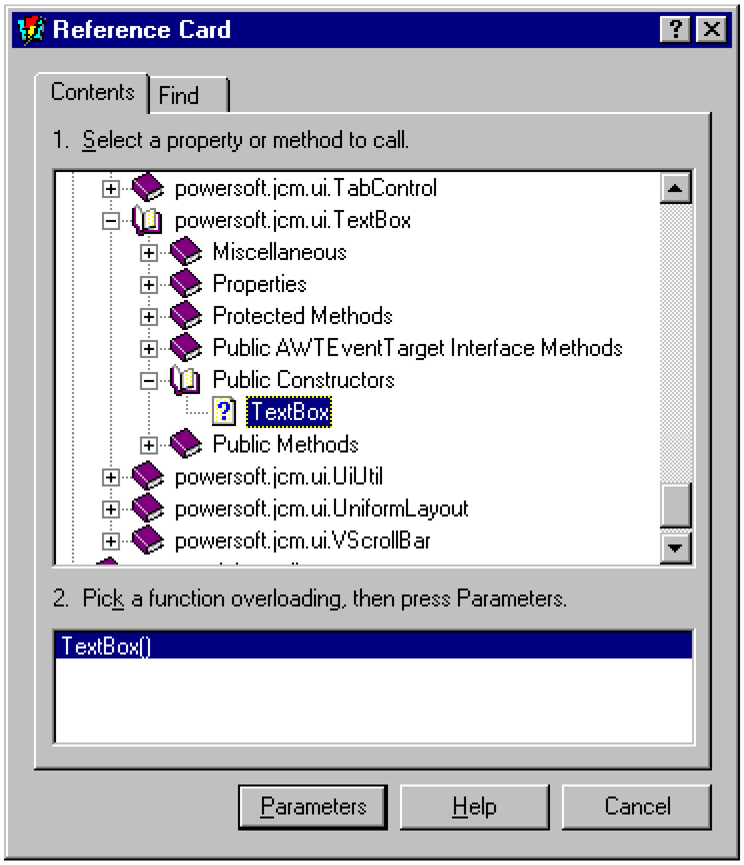
Code editor



```
    public boolean cb_1_Click(powersoft.jcm.event.ClickEvent event)
    {

        java.lang.String text;
        text = textb_1.getText( );
        lb_1.add( text );
        return false;
    }


    // add your data members here


}
```

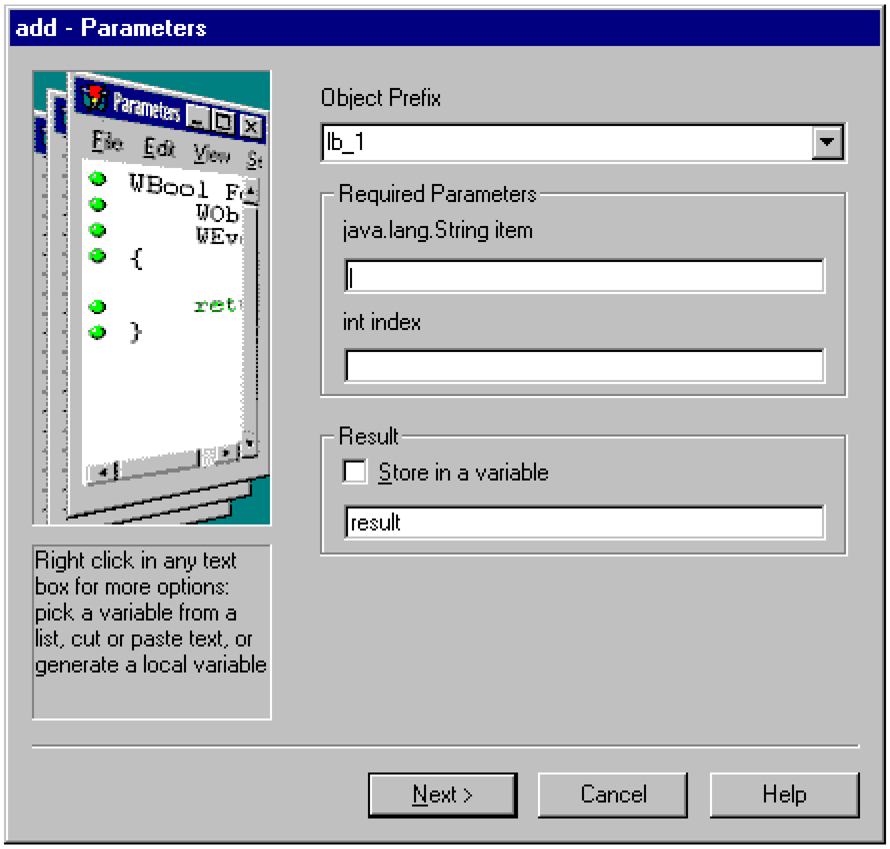
The Starbuck code editor is a full-featured editor that can edit the Java code for your Starbuck project. The code editor window is often used in drag-and-drop code editing. It also provides access to powerful facilities for debugging your program.
The code editor includes powerful features such as color syntax highlighting, auto indenting, unlimited undo, find and replace using pattern-matching, and debugging control. For more information about the code editor, see The Starbuck code editor.

Reference Card


Reference Card
Contents
Find
1. Select a property or method to call.
powersoft.jcm.ui.TabControl
powersoft.jcm.ui.TextBox
Miscellaneous
Properties
Protected Methods
Public AWTEventTarget Interface Methods
Public Constructors
TextBox
Public Methods
powersoft.jcm.ui.UiUtil
powersoft.jcm.ui.UniformLayout
powersoft.jcm.ui.VScrollBar
2. Pick a function overloading, then press Parameters.
TextBox()
Parameters
Help
Cancel

During your Starbuck session, you may use the Reference Card to select an action that you want to perform. You can search through a hierarchy of actions or perform a powerful text search to locate the desired action, then open the Parameter Wizard to create the Java code required to perform that action. The Reference Card also provides help on all the items listed – click **Help** to obtain the Starbuck Component Library Reference entry for a selected item.

Parameter Wizard



Object Prefix: lb_1

Required Parameters

java.lang.String item

int index

Result

[ ] Store in a variable

result

Right click in any text box for more options: pick a variable from a list, cut or paste text, or generate a local variable

The Parameter Wizard helps you construct a function call by showing what argument values are required. The Parameter Wizard can also store function results in a specified variable.

The text boxes in the wizard window let you enter variable names, constants, or expressions. You can use the right mouse button to click the text box and get a list of variables that may be entered. Click on a name from this list to place that name into the text box. You can also type in entries directly.

When you click **Finish**, the wizard pastes the constructed function call into the code editor window. The wizard also inserts declarations for any local variables that the method requires.

Basic concepts of Starbuck

The rest of this chapter discusses how Starbuck uses objects and forms to create an application.

The following sections provide an overview of how Starbuck works. The Starbuck Programmer’s Guide provides step-by-step instructions for performing the operations described here.

Forms

A *form* is a window that can be displayed on the user's system. When you use Starbuck to build a Java project, you usually create one or more forms. To create a form, you:

- Design the form by laying out objects (buttons, boxes, and so on).
- Specify properties for each object.
- Write Java source code to handle the events that might happen to each object (for example, when the user clicks a button or types text into a text box).

One form can open another form. For example, if the user clicks a button on one form, the associated source code may open a second form to obtain additional information from the user. There is no limit to the number of forms that may be associated with a project.

Forms in Java

In Starbuck, each form is defined as a Java *class*. This has several consequences:

1. A form is a data type, not a data object. In order to make a form appear on the user's screen, you must create an object of the form type.

   One of your form types is designated the *main form*. Your program automatically creates an object of this type when it starts up, to serve as the *initial window* that the user sees.
2. You can have multiple objects of the same form type. For example, suppose you design a form that displays the contents of a document. Your program can create multiple copies of this form, letting the user examine several documents at once.
3. Each form class has associated *properties*. Some properties affect the appearance of the form (for example, its color and its size). Other properties affect the behavior of the form (for example, whether the size of the form can be changed).

   When you design a form, you may specify initial values for the form's properties. These initial values are used whenever your program creates an object of the form class; however, you can change many of these properties later in program execution.
4. Each form class has a set of associated *methods*. A method is a function that lets you perform an action using the form. For example, a form has methods to examine or change the form's properties.
5. Using the Classes window, you can add your own methods to a form class. This is useful when you want to define a routine that can be used by other functions within the class, or when you want to provide controlled access to the class for objects outside the class.

The name given to a form is the name of the associated Java class. For example, if you name a form `Form1`, the associated Java class will also be called `Form1`.

Note: Starbuck lets you create many different types of forms. For example, a *dialog box* is a type of form which typically is opened to obtain information from the user and is closed once the user has filled in that information. When you create a new form, we suggest that you create a *modeless dialog* unless you have a reason for choosing some other form type. A modeless dialog supports more operations than the plain "form" type.

Objects

The first stage of using Starbuck is to place objects on a form. To do this, you select a component type from the Java component palette, then click on the form design window to specify the position of the object. In this way, you place buttons, text boxes, etc. on the form to design what the user will see when the form appears.

Object names

Starbuck associates a name with each object on the form. For example, Starbuck chooses names like `cb_1`, `lb_2`, and `Menu_1`. It is often a good idea to change these into more descriptive names: `cb_OK`, `fileList`, `menuSave`, and so on.

The name associated with an object is used as a variable name for referring to the object in Java code. For example, when you place a command button on a form, Starbuck might create the following declaration in the form class's source code:

```
CommandButton cb_1 = new CommandButton( );
```

This indicates that `cb_1` refers to a CommandButton object. Starbuck also generates code in the form class to initialize the CommandButton object. For example, this code sets various properties controlling the button's size and position. The variable `cb_1` is declared as a private member of the class for the form.

Since objects on the form are referenced by private variables within the form class, they cannot be referenced directly by entities outside the form. If `Form1` needs to affect an object on `Form2`, `Form1` must communicate with `Form2` and have `Form2` do the actual work.

Notation: Other parts of this guide use the notation *name_N* to describe the format of default object names. The *N* stands for an integer value, with values beginning at `1`. For example, label objects are given names of the form *label_N*. This means that the first label on the form is named `label_1`, the next is `label_2`, and so on.

Object methods

Each type of object has a set of associated methods. For example, a list box object has functions for adding new items to the list, deleting existing items, and so on.

```
lb_1.delete( 3 );
```

uses the **delete** method of a list box object to delete item 3 from `lb_1`. (The "." is the standard Java operator for invoking a method of an object.)

Some methods return a status value indicating whether they succeeded or failed. In many cases, this is a boolean value: `true` means the method succeeded and `false` means it failed. When the purpose of a method is to return a value, the function may return a special value to indicate failure. For example, if the purpose of a particular method is to return a string, the function will return a null string if the proper value cannot be determined.

Object properties

The properties of an object control the object's appearance and behavior. Starbuck lets you set the properties of an object while you are designing the form. Some properties can also be changed as your program executes. For example, you might disable the use of a command button at times when clicking that button is not appropriate.

Properties can be changed at run time using appropriate function calls. For example,

```
cb_1->setText( "New text" );
```

changes the text of `cb_1` to the given string. Methods that change the value of a property have names beginning with **set**; methods that return the current value of a property have names beginning with **get**. Every property has a corresponding **get** method and every property that can be changed at run time has a corresponding **set** method.

The Starbuck Component Library Reference documents the **get** and **set** functions for a property in the entry for the property. For example, **getText** and **setText** are explained in the entry for the **Text** property.

Events

An *event* is a message received by a form or an object on a form. Events may be generated directly by the user; for example, when the user clicks on a button, a **Click** event is triggered for that button. Events may also be triggered by the code of your program; for example, if `Form1` creates a `Form2` window, the action generates a **Create** event for the `Form2` object.

There may be many ways to cause the same event. For example, the **Select** event means that an item has been selected in a list box. This may happen because the user clicked a specific item, or because the user is using the arrow keys to walk through the existing list of items.

Writing Starbuck programs is mainly a matter of writing Java routines to handle events. For example, you do *not* have to write a mainline for the program; that is handled automatically by Starbuck. Instead you write a routine for what happens when the user clicks on one button, what happens when the user double-clicks an item in a list box, and so on. These routines are called *event handlers*.

A typical event handler has a name like

```
cb_1_Click
```

This incorporates the name of the object that receives the event (`cb_1`) and the name of the event itself (**Click**).

An event handler is a method belonging to the form class that contains the object receiving the event. This means that the handler has access to all the private members of the form object. In particular, it can work with all the objects defined on the form. For example, the **Click** event handler for `cb_1` can change the contents of a list box elsewhere on the form.

Event handler calling sequence

Every event handler is called with one argument: a block of information describing the event. Often, your event handlers will ignore this argument; the information is not necessary. In some cases, your event handler must place data in the information block to return a response to the caller.

Event handlers return `true` to indicate they handled the event completely or `false` if the default event handling of Starbuck should finish dealing with the event.

See Standard events for details about the way information is passed to an event handler, and how event handlers use their return values.

Starbuck and AWT

The *Abstract Windows Toolkit* (AWT) is a standard library of Java class definitions which define the common elements of graphical user interfaces: push buttons, list boxes, scroll bars, and so on. AWT was designed by Sun Microsystems to provide the basic building blocks for creating applets and other Java applications. For example, AWT specifies operations that can be performed on elements (for example, adding or deleting items in a list box) and events that can happen (for example, the user clicking a command button or entering text in a text box). AWT also defines other useful data classes (such as a String class for character strings and an Image class for "pictures").

The *Starbuck component library* is built on top of AWT. For example, one of the data members of a Starbuck CommandButton object is an AWT Button object. Similarly, many of the methods that can be performed on a CommandButton object correspond to AWT Button methods.

Most events that can be received by a CommandButton object correspond one-to-one with the events that can be received by an AWT Button. On the other hand, some events recognized for Starbuck objects do not correspond directly to AWT events, and vice versa.

All of this means that the component library is a Java *wrapper* around the basic data structures and operations of AWT. The components of the library parallel the facilities of AWT itself.

However, the library provides more than a simple wrapping: it gives you a complete structure for creating programs quickly and easily. Complex operations can be performed in a single instruction, avoiding tiresome set-up and tear-down operations. At the same time, the library gives you full access to low-level AWT features, for those rare occasions when you need to program directly with AWT classes.

Learning to create programs with Starbuck is primarily a matter of learning to use the design environment and the Starbuck library.

The rest of this guide outlines the major features of the library and presents simple examples using the library functions. The goal is to introduce the most common classes and methods within the library, to help you get results fast. For full details, however, you should consult the Starbuck Component Library Reference.

Important: With Starbuck, you do not have to use sophisticated features of Java. The Starbuck library helps you create efficient and effective programs, even if you do not have extensive knowledge of Java or AWT. Most of the user interface code that you write will consist of simple calls to library functions.