webGAIN

Spin User's Guide
Beta Release Draft

Spin User's Guide

The software described in this book is furnished under a license agreement and may be used only in accordance with the terms of the agreement.

Copyright Notice

Trademarks

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

WebGain LICENSE AND WARRANTY

CONTENTS

Chapter 1: Basic Ideas

Introduction to Web Applications 1-1
Building Web Applications with Spin 1-2
Components 1-3
  JavaBeans 1-3
    Properties 1-4
    Events 1-4
    Methods 1-4
  Enterprise JavaBeans 1-4
Actors 1-5
Behaviors 1-5
  Activating Behaviors 1-6
  Kinds of Behaviors 1-6
    Action Behaviors 1-6
    Action Group Behaviors 1-7
    Conditional Behaviors 1-7
    Script Behaviors 1-7
    Timeline Behaviors 1-7
    Counter Behaviors 1-8
    User Behaviors 1-8
Data 1-8
Java Expressions 1-9
Capsules 1-9
  Kinds of Capsules 1-10
  Capsule Hierarchy 1-11
  Aliases 1-11
Editors and Views 1-13
  Kinds of Editors and Views 1-13
  Edit as You Run 1-16
Debugging 1-16

Chapter 2: Building a Web Application

What Web Applications Do 2-1
  URLs 2-1
  Requests 2-4
  Sessions 2-4
  Responses 2-5
Building a Simple Servlet 2-5
  Set Up for Testing 2-8

Producing a Web Page 2-9
Adding Dynamic Content 2-12
Adding a Simple Form 2-14
Using a Template 2-20

Chapter 3: Accessing a Database

Accessing the Built-in Database 3-2
Browsing Connections 3-2
Building SQL Statements 3-3
Checking Options 3-5
Checking Syntax 3-6
Building Simple Statements 3-7
Specify a Connection 3-7
Specify a Table....................3-8
Specify Columns 3-8
Specify Conditions 3-9
SqlSelect 3-11
Choose Tables 3-11
Specify Joins....................3-12
Specify Sorting....................3-13
Format Columns....................3-13
SqlRawStatement and SqlProcedureCall 3-15
Building SQL Statements - Additional Tools 3-18
Executing SQL Statements 3-18
Causing Execution 3-21
Retrieving Results 3-22
Passing Parameters 3-23
Connecting to a Database.................... 3-25
Defining Connection Pools 3-25
spinsql.properties....................3-26
Using Connection Pools 3-27
Using a Dedicated Connection 3-27
For More Information.................... 3-28

Chapter 4: Using the Spin JSP Tag Library

JSP Control Tags.................... 4-1
call tag 4-2
conditional tag 4-3
repeat tag 4-4
JSP User Interface Tags 4-4
Attributes for Individual User Interface Tags 4-8

Creating a JSP with Embedded Spin Capsule 4-13

Chapter 5: Working with EJBs in Spin

Configuring Your Environment 5-1
Conventions 5-1
Configuring the Server 5-1
Copy the EJBs to the jBoss\deploy folder 5-1
Copy other files 5-1
Edit the jBoss.properties file 5-2
Add Database Drivers 5-2
Edit the jBoss.conf file 5-2
Configuring Spin 5-2
Process EJBs for Spin 5-2
Enable the Web Server in Spin 5-3
Enable jBoss in Spin 5-3
Enable Pointbase in Spin 5-3
Restart Spin 5-3
Running the BankTransactions Example 5-3
Learning from the Bank Transactions Example 5-4

Appendix A: Events

Events Generated by Behaviors A-1
IfTest A-1
ActionGroup A-1
Timeline and Counter A-1
Events Generated by Capsules A-2
Applet Capsules A-2
Application Capsules A-2
Nonvisual Actor Capsules A-2
Visual Actor Capsules A-2
Servlet Capsules A-2
Java Server Page Bean Capsules A-3
Events Generated by Other Built-in Components A-3

Appendix B: User Interface Reference

Viewing JavaDocs B-1
Project Editor B-1
File Menu B-2
Edit Menu B-3
General B-3
Display B-3

ScriptOptions B-3
EnterpriseJavaBeans B-3
DebugServer B-4
Project Menu B-4
View Menu B-4
Capsule Outline Editor B-5
Drag and Drop in the Outline B-5
Using the Toolbar B-6
File Menu B-7
Edit Menu B-8
Insert Menu B-8
View Menu B-10
Outline Menu B-10
Capsule, Behavior, or Actor Menu B-12
Capsule Menu B-12
Behavior Menu B-12
Actor Menu B-12
Toybox Editor B-13
Editing Handles B-13
Drag and Drop or Copy Objects in the Toybox B-14
Toybox Editor Menus B-14
Running versus Editing B-14
Debugging Using an Error Window B-14
Debugging Using the Console Window B-15

CHAPTER 1

Basic Ideas

Spin is a tool for assembling applications out of software components. It is especially useful for building enterprise applications such as web applications. This chapter explains what a web application is and introduces you to the basic ideas behind Spin. It covers the following Spin topics:

- "Introduction to Web Applications (page 1)
- "Building Web Applications with Spin (page 2)
- "Components (page 3)
- "Actors (page 5)
- "Behaviors (page 5)
- "Capsules (page 9)
- "Editors and Views (page 13)
- "Debugging (page 16)

Introduction to Web Applications

Originally, all computer applications ran on a single computer. You are probably used to single-computer applications such as word processors or spreadsheets. These applications contain both the *user interface* and *logic layer* parts of the application. The user interface is the windows, buttons, menus, and other means by which you make your will known to the program. The logic layer manipulates the underlying data (text, numbers, formulas, or e-mail messages). The entire application — both the user interface and underlying logic — runs on a single computer and accesses data on that same computer.

Client-server business applications break this single-computer model of applications. They use a model that separates the user interface from the underlying logic. The user interface runs on the user's computer (often a PC) and makes requests as a *client* to a *server* computer that manipulates the underlying data. For example, an airline reservation system might have many thousands of client PCs, one for each reservation agent, all connected to a central computer that makes flight reservations and issues tickets.

Client-server applications, such as airline reservation systems and bank teller systems, are a kind of *distributed application*. Distributed applications run on more than one computer. The pieces of the application communicate over a network.

The *World Wide Web* is itself a distributed application that runs over the Internet. It consists of computers that function as *web servers* — computers that serve resources such as text, images, sounds, or other applications. These resources are accessed using a *URL* (Uniform Resource Locator), which functions as the address to the resource. A *web browser* client, such as Netscape Navigator or Microsoft Internet Explorer, uses URLs to access various resources over the Internet and display them. One such resource is a *web page*. Web pages are resources whose contents are encoded (tagged or marked up) with *HTML* (Hypertext Markup Language). HTML consists of sets of tags that mark the structure of a web page's content. This tells the web browser how to display the page. An important HTML tag is the *link* tag, a hypertext pointer that contains a URL that links web pages and other resources into a giant interconnected web of information.

The World Wide Web was originally designed to be a publishing medium that contained only static information that didn't change much over time. A web page would change only when someone, like a webmaster, would update it manually. However, people soon wanted to use the Web interactively, and for dynamic information. Dynamic information is information that changes automatically, such as a web page that contains a company's current stock price and gets updated without requiring someone to edit the page manually. A familiar example of interactive use of the Web is a search engine such as Yahoo!, which lets you specify a list of keywords and then responds with a list of web pages matching those keywords.

Dynamic and interactive web pages are implemented using *server-side programs*, programs that run on the web server. So in addition to fetching static resources, a web server can also collect data (such as keywords or other search specifications) from the web browser client and execute a program that creates a web page dynamically and returns it to the web browser.

Server-side programs are combined with static web pages and other resources to build *web applications*. Besides search engines, other examples of web applications range from simple hit counters, to web sites that provide stock prices and weather reports, to complex e-commerce and business-to-business applications. Another use for web applications is to give web access to older client-server applications. For example, a business with a client-server human resources system might want to build a simple web application that allows employees to see how much vacation time they have accrued. Or a more complex web application can give customers online web access to their banking systems or to an airline reservation system.

Although creating static web pages using HTML is relatively easy, creating server-side programs such as web applications can be rather difficult. Web applications can be expensive to create: they are typically created manually by programmers. Another problem with building web applications is a lack of standards. Good standards, like HTML and HTTP, have allowed the development of powerful tools to make it easier to create and serve static web pages. But because there are few standards for web applications, most of the available tools to help you build web applications use proprietary technology, which typically locks you into using proprietary servers and has other limitations.

Building Web Applications with Spin

Spin offers a different approach to building web applications. Spin is a new kind of tool, an *application assembly* tool. Spin leverages standards to allow you to build complex applications out of components.

The advantages of building applications out of components are well established: dramatically increased reliability, supportability, scalability, security, and reuse. The hard part of building applications from components is connecting the components together, traditionally a huge integration problem. Spin solves this integration problem by providing a powerful and comprehensible application assembly technology based on *behaviors*. Applications you develop in Spin are:

- **Standards based.** Spin assembles open, standards-based web applications from prebuilt standards-based components, which you can acquire from a number of sources. Conforming to standards when writing your own components vastly improves the ease of use and reusability of those components.
- **Portable.** Your applications will run on any Java-enabled platform, including all major web application servers. Spin itself is written in Java.
- **Powerful.** In various places, you can insert arbitrary Java expressions or even whole chunks of Java code, in case the predefined choices do not permit the required functionality. Anything you can code in Java, you can create in Spin, so Spin never limits you.
- **Flexible.** Applications built with Spin are easy to change. You can modify your Spin application as your needs change over time. It is also possible to build generic applications, such as a generic e-commerce application, that can be easily customized to meet specific and individual needs.
- **Reusable.** Much of the work you do in Spin can be reused in other applications. The process of using Spin inherently creates new components.

Components

Spin builds applications out of *components*. Components are reusable software building blocks that work as single, separate functional units. Components are available for various applications, and new components can be created easily. Spin comes with a variety of components and the WebGain web site lists suppliers and distributors of additional components.

Spin works with many types of components, including JavaBeans and Enterprise JavaBeans (EJBs). This document refers to components that are either JavaBeans or EJBs simply as *beans*. Beans implement an interface that allows any application to determine automatically the services and functions they provide. Applications built with JavaBeans can run on any platform that runs Java, which includes all major web servers. Applications built with EJBs require a suitable EJB application server (such as BEA WebLogic) or an EJB container. At this writing, there are more than fifty such products in the marketplace; some of them are free.

Spin's component architecture allows various tools and parts to work together. Standardized nut, bolt, and screw sizes allow wrenches and screwdrivers from any toolbox to be usable; likewise, you can use Spin with any tool that works with JavaBeans or EJBs. Any valid bean you create can be used with Spin, no matter how you created it, and any bean you create with Spin can be used with other tools that work with beans.

JavaBeans

A JavaBean is a packaged Java class with the following qualities:

- It has *properties* with values that can be set and retrieved.
- It can generate specified *events*.

- It can execute specified *methods*.
- It can be serialized for storage or for transmission over a network, including its current state (such as the value of all of its properties).
- It is packaged as a .jar file with supporting classes and resources, complete and ready for use in an arbitrary number of applications.

Spin comes with many JavaBeans. See the `Spin/beans` directory for some examples.

For more information on JavaBeans, see:

```
http://java.sun.com/products/javabeans
```

Properties

Properties are the qualities of a component that you can modify. Properties have values: the color of a background, the size or location of a window, the identification number or balance of a checking account.

Events

Many JavaBeans, particularly those that represent user interface components, generate events. An *event* is any occurrence of interest to an application, such as a mouse click, a key press, or the opening of a file. A timer can generate an event according to a schedule. A bean can generate no events, one event, or many events.

Events use the Java two-part naming convention: an event class followed by the name of the specific event, with a dot as a separator. For example, a mouse click is `Mouse.mouseClicked`. The event class is `Mouse` and the specific event is `mouseClicked`.

For the complete list of events defined for specific Spin components, see Appendix A.

Methods

All Javabeans have methods. A *method* is a procedure that executes when the method is invoked. Invoking a method on a component typically causes the component to do something.

The properties of a JavaBean are implemented using two methods, the property's get and set methods. For example, an animated object could use the two methods `setAnimationRate` and `getAnimationRate` to allow you to edit the current value of the property `animationRate`. Spin hides such methods from you and lets you view or edit a component's properties directly, rather than by calling methods. Spin also allows you to execute any method from a menu, as long as the method does not require an any arguments (additional data) to execute.

In Spin, executing a method that takes an argument requires a behavior. Since Spin is running your application while you are building it, it is easy to drop a behavior onto a bean and use it to execute an arbitrary method on that bean.

Enterprise JavaBeans

Enterprise JavaBeans provide portable access to enterprise services, including:

- databases
- distributed transactions
- security
- messaging
- mission-critical robustness and scalability

Enterprise JavaBeans make use of specific conventions so that applications containing EJB can access enterprise services, in a portable manner, from within any EJB-enabled web application server. See Chapter 5 for more information about working with EJBs in Spin.

For more information about Enterprise JavaBeans, see:

```
http://java.sun.com/products/ejb
```

Actors

Spin combines a component architecture with an authoring technology based on *behaviors*. With behavior-based authoring, you assign behaviors to *actors*. One of the unique features of Spin is that it allows you to use an arbitrary bean as an actor.

An actor can be visible on the display, such as a clock or an animation, or can be a nonvisible component, such as a component that reads a file or generates HTML. When an actor is visible on the display, it's called a *visual actor*. Spin comes with a set of predefined actors, some visual and some not.

There is sometimes confusion in the Java world because the base class of the visible user interface widgets in the Abstract Windowing Toolkit (AWT) is named "Component." Even though many visual components in Spin are derived from the AWT Component class, the concept of a JavaBean or EJB component in Spin should not be confused with the AWT Component class. Because an actor is a Spin component, actors also have properties, events they can generate, and methods they can execute.

Behaviors

A *behavior* causes an actor to do something, enables communicate between actors, or responds to user input. Every behavior has a stimulus and a response:

- The stimulus for a behavior is always an event. When the proper event arrives, it activates the behavior.
- How the behavior responds when activated depends on the kind of behavior. The behavior might call a method on an actor, set the value of a property, or generate another event.

For each field in a behavior editor dialog box, there is a menu that lists appropriate values for that field. This menu is activated by clicking on the arrowhead to the right of each field.

Spin comes with several different kinds of predefined behaviors. You can also assemble new behaviors out of existing behaviors and save them for reuse.

Activating Behaviors

When a behavior receives its stimulus and begins to run, we say it is *activated*. All behaviors are activated by an event. You can specify any number of events that activate a behavior. When you specify more than one stimulus, the behavior activates if any one of the specified events are received. For each event, you can also specify a condition to be evaluated. When the associated event is received, the behavior evaluates the condition and activates only if the result is true.

Certain kinds of behaviors, once activated, continue to execute for some period of time. These behaviors also allow you to specify an event, along with an optional condition to evaluate, that will cause the behavior to deactivate. When a behavior is *deactivated*, it stops executing. Two behaviors that can be activated and deactivated are the counter and the timeline behavior. (These two behaviors are rarely used in web applications, however.)

Kinds of Behaviors

Spin defines seven kinds of behaviors:

- action
- action group
- conditional
- script
- timeline
- counter
- user behaviors

Action Behaviors

An *action behavior* is the most common kind of behavior in Spin. When activated, an action behavior invokes a method on an actor. You specify this method as follows:

- You specify the actor on which to invoke the method in the `Send To:` field.
- You Specify The Specific Method To Execute On The Actor In The `Message:` Field.
- You specify any arguments (additional data) required by that method in the `Data:` field. A method may require no arguments; in this case the `Data:` field is not displayed.

A single method name may offer several ways of specifying additional data. For example, a method that requires a color as an argument might allow you to specify that color as an object of type `Color` using three integers that represent the red, green, and blue components of the color. Alternatively, you might be able to specify a string, such as "orange" or use a pulldown menu to select the color.

You can specify each required argument using either a constant or an evaluated expression. Spin lets you do this by toggling (click on whichever type is showing to show the other) between two types of argument:

- `With:` (an argument specified using a constant)

- `With=` (an argument specified as an evaluated expression)

One of the powerful features of Spin is that it allows you to specify an evaluated argument using an arbitrary Java expression. This expression can use values supplied by other actors, call arbitrary Java functions, and effectively do anything you can program in Java.

Action Group Behaviors

An *action group behavior* lets you group several behaviors so they can be moved or copied as a unit. When activated, the action group behavior responds by generating another event called `Behavior.activated`. The behaviors grouped in the action group can all receive this single event as their stimulus, responding according to their definitions. However, this is up to you — some or all of the behaviors in the group can be triggered by other events, and other behaviors not in the group can be triggered by that event.

Conditional Behaviors

Spin allows you to specify conditional activation for any behavior. Although this is useful in many cases, Spin also provides *conditional behaviors,* a more general-purpose mechanism.

A conditional behavior allows you to embed an explicit *if* test anywhere you can locate a behavior. When a conditional behavior receives its stimulus, it evaluates its associated expression. If the result is true, it generates an `ifTrue` event; if the result is false, it generates an `ifFalse` event. Other behaviors can use either of these events as a stimulus, causing them to execute under the appropriate circumstances.

Script Behaviors

A *script behavior* allows you to execute any arbitrary sequences of Java statements in response to an event.

As discussed earlier, building applications out of components presents a huge integration problem. Components from different source are not designed to work well together. Script behaviors allow you to using Java code to perform this integration. While this is a powerful feature of Spin, note that a heavy use of script behaviors is a sign that you should be purchasing (or writing) better components.

A script behavior can also be used when you cannot find or do not want to write a component to provide a desired functionality. Again, while this is a powerful feature that allows you to create any application in Spin, it is usually better to create a component with the desired functionality. New components can be created directly in Spin, or can be programmed using any Java programming environment (such as WebGain Visual Cafe).

Timeline Behaviors

A *timeline behavior* is one that modifies the properties of one or more actors over time, similar to an animation storyboard or a musical score.

A timeline can modify any property of any actor: you specify the values of a given property at certain points along the timeline, and these values change accordingly when the behavior is activated. For many types of properties (numeric values, colors) Spin can interpolate between values so that the value changes smoothly as the timeline progresses. This interpolation can be done linearly, or with an ease-in and ease-out feature.

A timeline behavior can be deactivated before it is finished executing.

Counter Behaviors

A *counter behavior* is another way to manipulate occurrences over time. A counter behavior counts time, starting when it is activated, and generates events at specified time intervals until it reaches the specified stopping point.

A counter behavior can be deactivated before it is finished executing.

User Behaviors

A *user behavior* allows you to reuse any behavior you have defined by explicitly saving it as a behavior. To save a complex behavior, you must define it as a set of behaviors grouped together under a single behavior (typically an action group behavior). You then select the root behavior and choose File>Save Behavior. The behavior then appears in the Insert>User Behavior menu item.

Data

Data variables allow you hold values of various types. For example, a variable can hold a name or the color of a background. Spin lets you specify the following types of variables:

- **int**, a 32-bit integer number
- **float**, a 32-bit floating-point number
- **double**, a 64-bit double precision floating point number
- **boolean**, a Boolean value (true or false)
- **string,** a string of characters
- **color**, a color
- **Hashtable**, a hash table (a key-value lookup table), also called a dictionary
- **Vector**, a vector (growable array of objects)
- **Point**, a x, y (2D) point
- **Rectangle**, a (2D) rectangle
- **Dimension**, a (2D) dimension (a width and height)
- **URL**, an HTTP URL
- **HttpCookie**, an HTTP cookie

The Spin data variables are all either Java primitive types, or types defined in standard Java libraries (such as java.util, java.awt, and java.net), with the exception of HttpCookie, which is a special Spin data type.

Java Expressions

Spin provides the ability to embed scripts written in the Java programming language in three ways:

- Java expressions can be evaluated to provide arguments to method calls in action behaviors.
- Boolean (true/false) expressions can be evaluated to control the activation and deactivation of any kind of behavior, or as the test condition in a conditional behavior.
- Sequences of Java statements can be executed in script behaviors.

Normally, compiled programming languages like Java are not used as scripting languages because of the enormous difficulties compiling and linking scripts dynamically. Instead, dynamic interpreted languages are used, but this invariably leads to performance problems. Spin solves the performance problems by using a compiled language — and Spin can take advantage of future advances in compilation techniques for Java, and so become even faster. In order to use a compiled language like Java as a scripting language, Spin takes advantage of several advances in dynamic compilation and linking techniques, so that scripts can be recompiled and relinked even while your application is running.

In addition, of the most powerful visual authoring systems that do provide scripting capabilities, most use a proprietary language, typically one that is unique to that system. This forces the advanced user who wants to use scripts to learn yet a new computer language, without the benefit of the many books and training materials available for a standard and popular language like Java. In addition, it is almost guaranteed that a proprietary language will be missing features and library functions found in a standard programming language. When you write a Java script for Spin, you can take advantage of all the features and vast numbers of library functions available in Java.

To make Java more suitable for writing scripts, Spin adds several upwardly compatible features to Java:

- Spin dynamically resolves names so that scripts can reference other objects (actors, behaviors, and data variables) and can set their values.
- Spin performs conversions between Java primitive types and their object counterparts. For example, Spin integer data variables can be passed, without error, to methods that expect either type `int` or type `Integer`. This vastly simplifies the Spin's use of values without sacrificing compatibility with the Java language.
- Spin automatically bundles scripts (both expressions and script behaviors) as automatically generated method calls on automatically generated objects, as required by Java. This relieves the overhead of programming for the Spin user.

The result is a powerful scripting language that is remarkably easy to use.

Capsules

Visual programming systems make it very easy to build small, toy applications, but make it virtually impossible to build real, complex applications. To solve this problem, Spin organizes applications into *capsules*. As indicated by the name, capsules are a visual metaphor for the encapsulation techniques proven by object-oriented languages.

Capsules can contain actors, behaviors, data variables, and other capsules. When you create a capsule, you are creating a new component (a JavaBean) that can be used, like any component, as an actor. The use of capsules has several advantages:

- Spin allows you to build complex applications hierarchically.
- A capsule, like any component, can be reused in other applications.
- Capsules organize your work, allowing you to focus on one piece at a time.
- A capsule has a well-defined interface that provides modularity. As long as the external interface and behavior of a capsule does not change, you can change the internal implementation of a capsule without affecting other parts of your application that use that capsule.
- Capsules help support collaboration. Multiple Spin users can work on the same application, as long as each user works on a separate capsule.

Kinds of Capsules

A capsule can represent different kinds of things:

- a stand-alone application
- an applet
- a servlet
- a JSP bean
- a visual actor (a component derived from the Java class `Component`)
- a nonvisual actor (a component with no visible run-time representation)

The last two kinds of capsules, visual and nonvisual actors, create new components that can be used as actors in other capsules.

JSP bean capsules create components that can be called from Java Server Pages (JSP), web pages that contain embedded Java calls to provide dynamic output. Otherwise, capsules represent top-level entities — either an application, applet, or servlet.

Application capsules can be used to create stand-alone, single-computer applications or the client portion of client-server applications. Application capsules usually have their own user interface, built using standard Java user interface widgets.

Applet capsules can be used to create applets that are downloaded over the Internet and run in a browser. Note that Java compatibility and security is still an issue in major web browsers, so applets are really usable only in intranets or other controlled environments.

Servlet capsules run on a web server and are the primary building blocks of web applications. Servlets typically implement a "thin-client" user interface in HTML, which executes on a browser. Servlet capsules can use Spin's built-in web server for testing and debugging.

Capsule Hierarchy

Items in a capsule are organized in a hierarchy, which you manipulate as an outline. The figure below shows the outline view of a capsule.

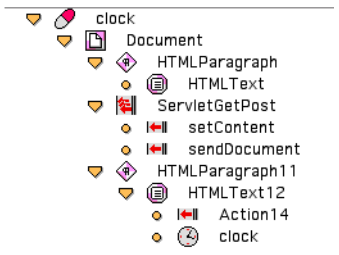


Figure 1-1: Capsule outline view

The capsule is at the top of the hierarchy. The remaining levels of the hierarchy contain the capsule's *children* — actors, behaviors, and data variables that make up guts of the capsule. Just as in life, the relationships parent and child are reciprocal; if Actor A, for example, is the child of the capsule, then that capsule is the *parent* of Actor A.

Actor A might in turn be the parent of other actors, or perhaps of several behaviors. It is common for an actor such as Actor A to have one or more behaviors as children — those behaviors associated with Actor A. These behaviors are in turn the children of Actor A, and Actor A is their parent.

The capsule does not enforce any specific hierarchical relationships among its constituents. Any item in the outline can be selected and moved up or down in the hierarchy. When you do so, its parent changes.

The capsule outline is also one means of defining the order in which things execute. As described previously, behaviors are activated by their triggering event. But if several behaviors have the same triggering event, then they execute from top to bottom, according to their order in the outline view.

Aliases

Normally, actors, behaviors, and data variables are referenced by name. This limits reusability, since the names would have to be changed manually to apply, for example, a behavior to a different actor. To increase reusability, Spin provides three *aliases* that can be used by behaviors to reference objects by their position in the capsule hierarchy. These aliases are available from various menus as you define behaviors. Their meanings are as follows:

parent	This alias refers to the immediate parent of the current behavior, which might be the capsule, an actor, or another behavior, of which this behavior is a child.
actor	This alias refers to the nearest direct or indirect ancestor that is either an actor or, if none is found, the capsule itself. If a behavior is the immediate child of an actor, this alias refers to the same entity as `parent`.
capsule	This alias refers to the capsule that immediately contains the behavior.

To illustrate these aliases, imagine a capsule that contains one actor — a button — that has two behaviors: one to implement an action when the button is pressed, and another to implement a rollover effect to highlight the button when the cursor passes over it. Figure 1-2 shows the capsule outline view for this example.

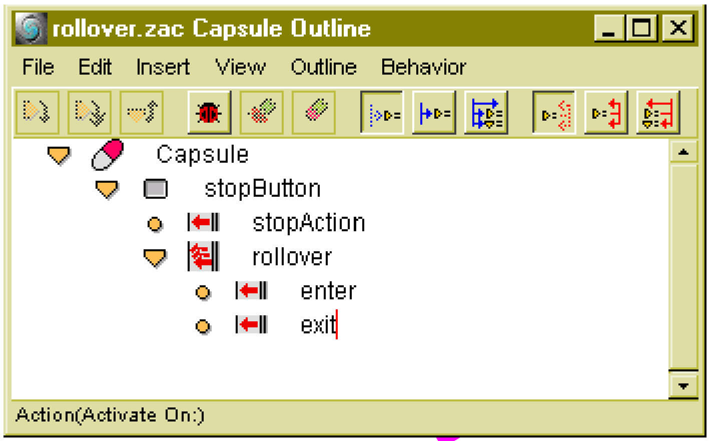


Figure 1-2: Capsule outline view for button with children behaviors

Note that the rollover behavior is a complex behavior whose root is an action group, which itself has two children action behaviors: one to execute when the mouse enters the button and one for when it exits. In the exit behavior, the alias `parent` refers to the rollover behavior, the alias `actor` refers to the `stopButton` actor, and the alias `capsule` refers to the capsule named `Capsule`.

In the `rollover` behavior, the aliases `parent` and `actor` both refer to the button named `stopButton`. Using aliases in behaviors, rather than referring to entities by name, makes it easier to reuse behaviors. For example, by using aliases you could move the `rollover` behavior onto another button (say, a new one named `startButton`), without having to change any names inside the behaviors.

Note that when you use a behavior to connect two actors, the behavior can be the child of only one of the two actors. For example, a behavior can be used to connect the button to an animated object, such as a juggler, so that the object starts moving when the button is pressed. In this case, the behavior can be the child of either the button or of the juggler. If the behavior is a child of the button, then the alias `actor` refers to the button, but the juggler must be named explicitly. If the behavior is a child of the juggler, then the alias `actor` refers to the juggler, but the button must be named explicitly. How you organize your capsules is a design issue. Since it is likely that several buttons will be used to control the juggler, it is probably better to make the behavior a child of the button so that you can reuse the behavior on multiple buttons.

Editors and Views

Editors are used to edit the contents of capsules — to add, delete, or modify the items they contain. We have already seen the outline editor of a capsule, but the outline editor is only one of several possible editors (or views). Different editors and views provide different functionality, and make it easier to build applications using Spin.

Kinds of Editors and Views

The views and editors in Spin are:

◆ The **project editor**, shown in Figure 1-3, allows you to organize the many files (capsules, HTML and JSP web pages, images, and so on.) that make up an application. It lets you set preferences and keep track of database connections. These preferences allow you to specify a directory (in addition to Spin's *docs* directory) that the Get Info command can use to find documentation. You can also specify parameters with which you can test the servlets you build.

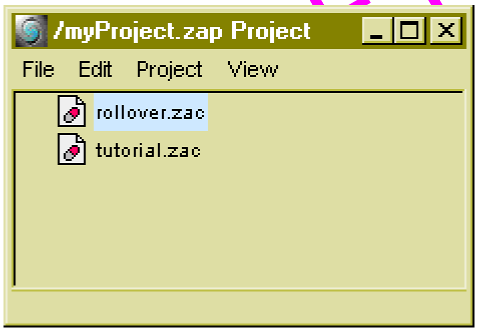


Figure 1-3: Project editor

- The **capsule outline editor**, shown in Figure 1-4, shows the hierarchical view of a capsule. It allows you to add or remove components from a capsule, change their position in the hierarchy, and edit individual actors, behaviors, and data variables.

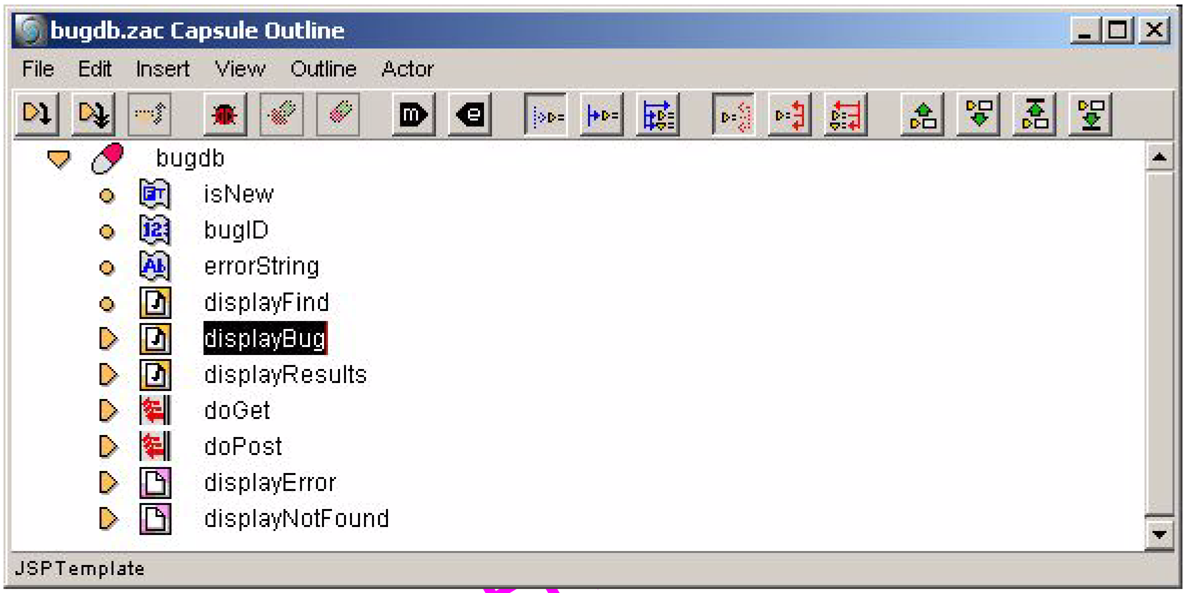


Figure 1-4: Capsule outline editor

- The **toybox editor**, shown in Figure 1-5, is primarily useful for applications with a graphical user interface, such as capsules containing visual actors. It is not used for servlets. The toybox editor allows you to edit your application while it runs. You can use this editor to change components' properties, to add, delete, or modify behaviors, and to see the results of your work instantly.

  Each component in the toybox view appears with an editing handle that you can use to move or resize the component, or to select it for some other operation. These editing handles also reveal helpful status information, such as the location of a component you are moving, or the behaviors that are the immediate children of the component. You can double-click on these behaviors to edit them from the toybox, instead of the outline editor.

  You can hide the editing handles and the toybox grid to see what your application will look like without them. You do this using the View>Hide/Show Editor command in the toybox window.

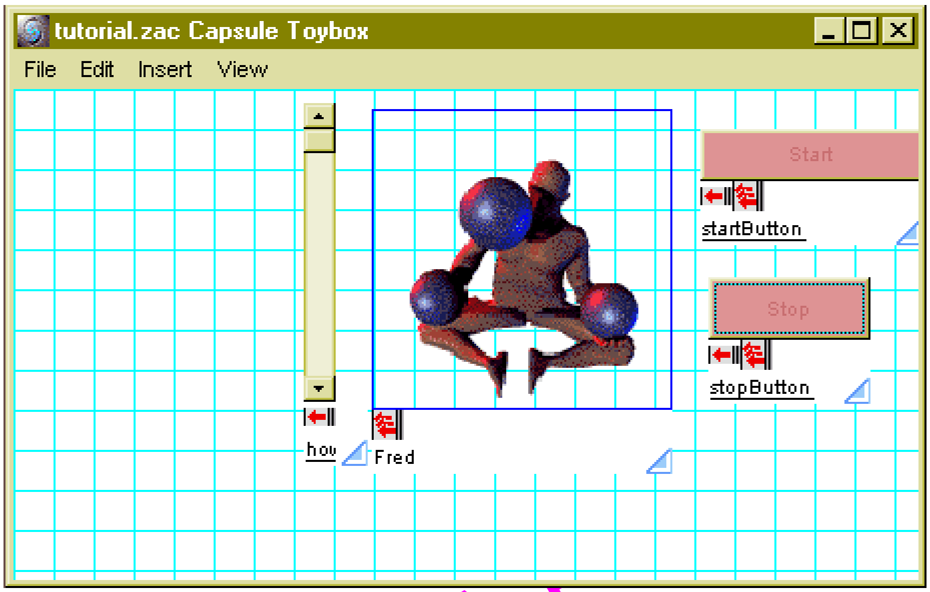


Figure 1-5: Toybox editor

◆ The **run view**, shown in Figure 1-6, compiles and runs an application capsule exactly as a user will see it.

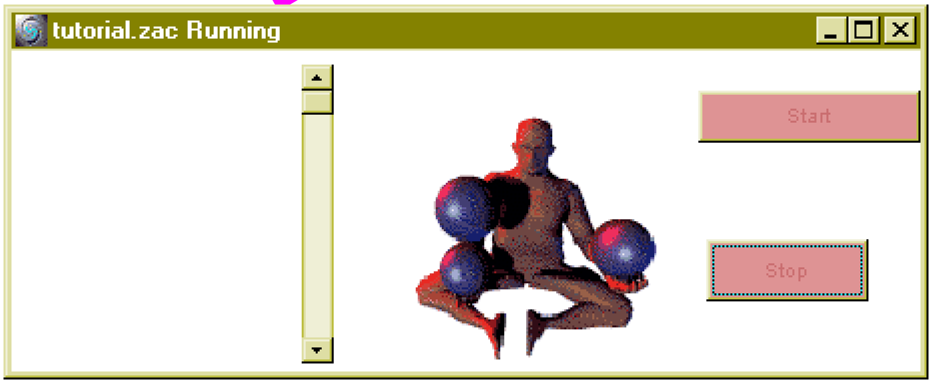


Figure 1-6: Run view

◆ The **layout editor**, shown in Figure 1-7, allows you to view and edit the layout of scenes or other visible components, such as windows, that are subclasses of a container. A scene is a view with one or more components ordered from back to front that change according to time or user input. Scenes are useful for creating animations or laying out user interfaces.

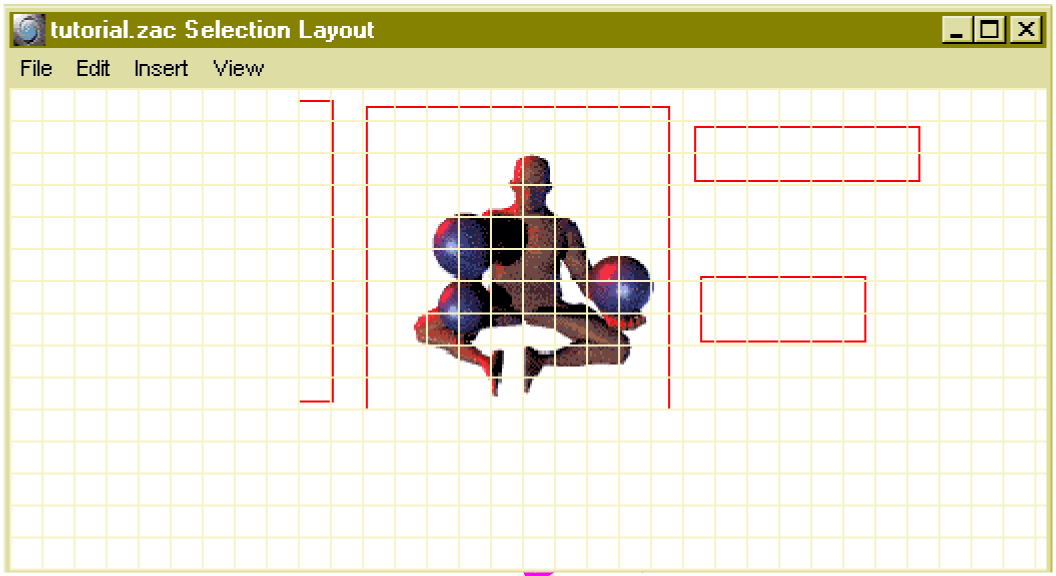


Figure 1-7: Layout editor

- The **browser view** of a servlet, generated by Spin's built-in web server, allows you to view the HTML output from a servlet capsule in a standard web browser as you build it.

In addition, Spin provides editors for all actors (components) and data types.

Edit as You Run

Spin runs your application continuously as you build it. There are no separate edit, compile, and run steps to see the results of your work. When you have an idea, nothing prevents you from seeing immediately whether it works — and if it does not work, you can play with it until it does.

The ability to work with a running application is one of Spin's most powerful features; it can feel like the difference between being able to hold an item directly in your hands versus having to work with robot arms.

The toybox view is used to run an application capsule while you edit it. For servlet capsules, you use the debug web server to view the HTML output from your servlet while you edit it.

Debugging

Because Spin runs your application as you work on it, Spin encourages you to experiment. Sometimes that leads to mistakes. In addition, errors can occur in the Java expressions you add to behaviors in various places, or in the blocks of Java code you add as script behaviors. Spin provides the following debugging tools:

- When Spin encounters a bug in your application, one or more error messages appear in a pop-up error window. Click on an error message, and Spin opens the editor for the behavior that contains the error.
- Spin has a console window to which it writes certain kinds of error messages. You can also use this console for displaying debugging statements. For example, if you want to know when a certain script is executing, you can add a line to the script such as the following:

  ```
  System.out.println("now executing Script A");
  ```

  Leave the console window open while you run, and you can determine whether the compiler reaches your debugging statement.
- The Spin debugger allows you to set breakpoints, single-step through your application one behavior at a time, and view the values of data items as they change. An arrow in the outline view shows you where the compiler has stopped, and a debug window allows you to inspect the values of any data variables you have asked to watch. You can see when the data variables change, and what they hold at any time.

CHAPTER 2

Building a Web Application

Spin allows you to create applications whose functionality can be distributed across any of several pieces: servlets running on a web server, applets running on a client browser, or stand-alone applications. This chapter describes how to use Spin to create servlets: applications that run on a web server and respond to requests from remote client browsers.

This chapter covers the following topics:

- What Web Applications Do
- A Simple Servlet
- Additional Form-handling

What Web Applications Do

As explained in Chapter 1, web applications allow users to interact with web servers in ways that are much more powerful than displaying static, HTML-coded information. Web applications provide the difference between masses of linked, but static, information and customized, real-time data generated in response to the actions of a specific user.

Web applications provide the truly interactive experience we have come to expect from the World Wide Web. Search engines, online storefronts, and web sites that provide weather forecasts are all examples of web applications. The applications you design determine a web user's experience when that user explores your data, product line, or services.

Your challenge is to build a web application that responds and adapts to broad ranges of queries, offers richly detailed information, behaves flexibly, and allows users to meet their needs. The ideal system responds with unique pages for each request. A *servlet* constructs appropriate responses to requests as they stream in. This chapter explains how to use Spin to build servlets.

URLs

The main mechanism by which browsers find web servers is the URL. Consider the following minimal example:

```
http://www.webgain.com/
```

A URL consists of at least:

- the protocol to use for the connection (in this case, `http`), and
- the name of the web server (in this case, `www.webgain.com`).

These are separated by a colon and two forward slashes `://` and ended with a forward slash `/`

protocol	A request for a web page uses *HTTP*, Hypertext Transfer Protocol. Other possible protocols are FTP for direct file transfers, or MAIL for e-mail.
web server	The name of the Internet host computer that functions as the web server. A web server is not actually identified by name, but by IP address — a set of four numbers separated with decimal points, such as 192.168.24.55. The Domain Naming Service (DNS), a global Internet facility, translates web server names into IP addresses for you.

In addition, URLs can specify a particular web page:

```
http://www.webgain.com/login.html
```

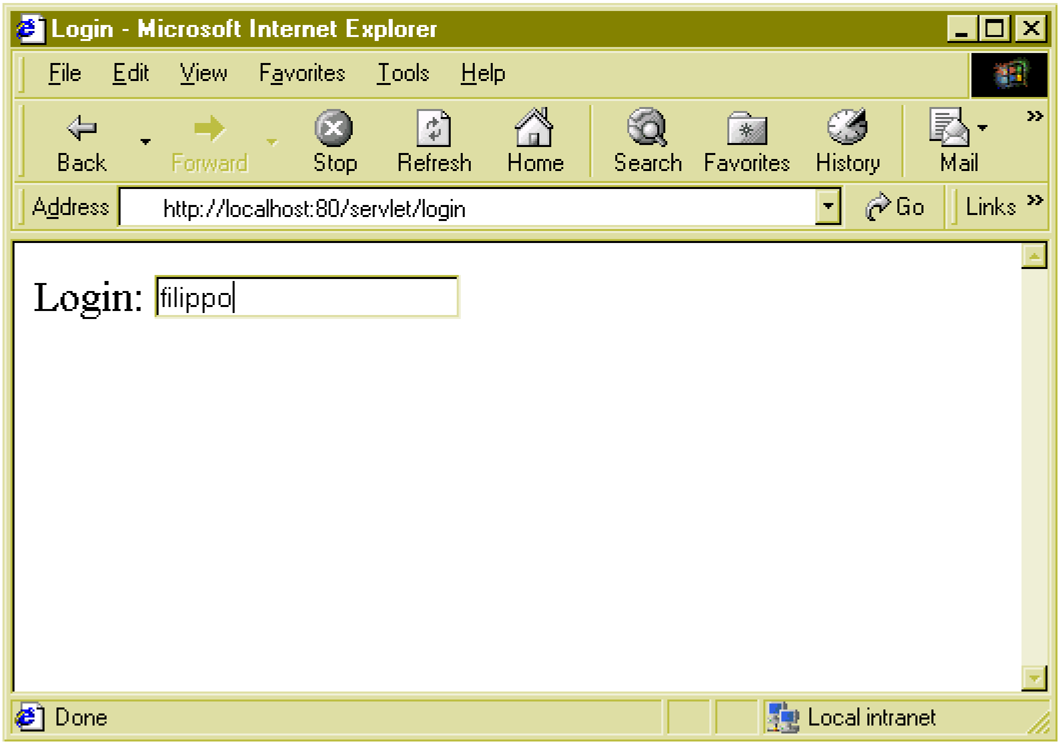
In the URL above, `login.html` specifies the specific page requested. A single forward slash `/` separates the server name from the page name. If no specific page is requested, the default page — usually a page named `index.html` or `default.html` is served.

URLs can also specify a port:

```
http://www.webgain.com:81/login.html
```

On any given machine, different protocols listen at different ports. A port number identifies a specific service on a machine, sending requests to the protocol for which they are intended. By default, a web server listens for HTTP requests at port 80. If a web server is configured to listen at another port, the URL must include a colon after the server name, followed by the port number.

All of the above merely sends a web page: the problem is to build them dynamically. In order to respond to queries from the user, you have to be able to get queries from the user. Users tell you what they want to know more about by filling out forms:


Login - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail
Address http://localhost:80/servlet/login Go Links
Done
Local intranet

Login: filippo

Each field in the form is a parameter: a name for the field, and a value (whatever the user enters in it). All these names — as well as other fields in other forms your servlet may specify — define the set of named parameters for your servlet.

The above login page constructs the URL below:

```
http://www.webgain.com/login.html?user=filippo
```

parameters	Separated from the page by a question mark, the parameter name (`user`) is followed by an equals sign, then the value (`filippo`).

You can string more parameters together with ampersands, like this:

```
http://www.webgain.com/login.html?user=filippo&password=*****
```

URLs thus do a lot of work. They can specify:

- the protocol
- the web server
- a special port
- a specific page
- the name-value pairs of any relevant parameters

Requests

Hypertext Transfer Protocol (*HTTP*) defines various types of requests to allow a range of interaction between web servers and browsers. The request type is encoded not in the URL, but elsewhere, and is therefore invisible to users. In most interactions with servlets, however, the browser's request is either of type GET or POST. Both GET and POST can originate from any kind of web page, including a form. These two requests are very similar:

- GET sends a URL; parameters, if any, are appended to the end of the URL, where they are visible to users.
- POST sends a URL; parameters are sent in a separate part of the request, where they are not visible to users.

The URL has a length limit. The specific limit depends on the web server, but 256 characters is typical. Therefore, there is a limit to the number of parameter name-value pairs you can send in one GET request. Also, if one of the parameters is, for example, a password, you might not wish its value to be displayed in plain text in the URL of a web browser. These are the main reasons to use POST instead of GET.

The web server responds to either request by sending the requested document and, if necessary, updating parameters.

Sessions

Most web applications involve more than one servlet. For example, e-commerce applications often have one or more servlets to allow customers to select items and place them in their shopping carts, and other servlets to pay for the items and ship them. If Filippo and Amaia are both shopping at the same time, the servlets must be able to keep their interactions separate.

Servlets use *sessions* to identify which user is sending a request. To a web server, each session is identified in some unique way, usually with an identification number. The session is typically implemented as a hash table that contains the data (the name and value of each parameter) for the session. For example, the session for an e-commerce application could contain the customer's shopping basket, shipping address, and other user-specific information. Various servlets in an application can access the session and store information in it.

When a servlet receives a GET or POST request, it must be able to identify the session to which the request belongs. The servlet can do this in either of two ways:

- The servlet can rewrite each URL on each HTML page it generates, appending a parameter containing the session identifier.

  Or:
- The servlet can send a *cookie* containing the session identifier to the browser and receive it back again with each request. A cookie is a text file, sent by the web server, that a web browser saves and sends back with each request. It can contain a session identifier, as in this example, or any other information the server may need to store on the client.

Thus, the application is able to retrieve the session identifier, either as a parameter or from the cookie, and access the current session's hash table. Common web servers allow you to specify which mechanism to use to identify sessions.

Note: Some older browsers do not support cookies, and browser users are able to disable cookies if they wish.

Depending how your web server is set up, and whether cookies are allowed by the user's web browser, Spin automatically takes care of rewriting URLs or storing and retrieving cookies so it can retrieve the correct session for you. All you need to do is to define session parameters and use them to store information that must be associated with a particular user.

Responses

Just as the first part of a URL specifies the protocol — the rules used to interpret the rest of the URL — when a web server responds with a document, the first piece specifies the rules used to interpret the document's content type. Content type is also called *MIME* (Multipurpose Internet Mail Extension) type. After sending the MIME type, the web server sends the contents. After sending a MIME type of HTML text, for example, the web server would send an HTML document.

Building a Simple Servlet

When you use Spin to build servlets, you typically build HTML documents using Spin's HTML components. These are described in detail below, but familiarity with HTML can help a great deal. For an elementary introduction to HTML, see:

```
http://www.gettingstarted.net
```

Yahoo's index of all HTML tutorials can be found at:

```
http://yahoo.com/Computers_and_Internet/Data_Formats/HTML/
  Guides_and_Tutorials
```

Also helpful are HTML references at:

```
http://yahoo.com/Computers_and_Internet/Data_Formats/HTML/Reference
```

Spin can construct HTML documents using the HTML components described in Table 2-1.

Table 2-1: Spin HTML Components

	Spin HTML Component	Icon	Corresponding HTML Tag(s)
Documents	HTMLDocument		`<HTML> </HTML> <TITLE> </TITLE> <HEAD> </HEAD> <BODY> </BODY>`
Lists	HTMLList		`<UL> </UL> <OL> </OL> <DL> </DL>`
	HTMLListItem		`<LI> </LI> <DD> </DD> <DT> </DT>`

Table 2-1: Spin HTML Components

	Spin HTML Component	Icon	Corresponding HTML Tag(s)
Tables	HTMLTable		`<TABLE> </TABLE>`
	HTMLTableRow		`<TR> </TR>`
	HTMLTableCell		`<TD> </TD> <TH> </TH>`
Forms	HTMLForm		`<FORM> </FORM>`
	HTMLFormInput		`<INPUT>`
	HTMLFormSelect		`<SELECT> </SELECT>`
	HTMLFormSelectOpt		`<OPTION> </OPTION>`
	HTMLFormTextArea		`<TEXTAREA> </TEXTAREA>`
Other containers	HTMLHeading		`<H1> </H1> ... <H6> </H6>`
	HTMLParagraph		`<P> </P>`
	HTMLAnchor		`<A> </A>`
	HTMLBlock		`<BLOCKQUOTE> </BLOCKQUOTE>` `<DIV> </DIV>`
	HTMLFontStyle		`<FONT> </FONT>`
	HTMLTextStyle		`<BIG> </BIG> <B> </B> <CIT> </CIT> <CODE> </CODE> <DFN> </DFN> <EM> </EM> <I> </I> <KBD> </KBD> <SAMP> </SAMP> <SMALL> </SMALL> <STRIKE> </STRIKE> <STRONG> </STRONG> <SUB> </SUB> <SUP> </SUP> <TT> </TT> <U> </U> <VAR> </VAR>`
Other noncontainers	HTMLHorizontalRule		`<HR>`
	HTMLImage		`<IMG>`
	HTMLLineBreak		`<BR>`
	HTMLText		A string of characters
Execution control	HTMLDoMultiple		Outputs a chunk of HTML any number of times, as specified by the behaviors.
	HTMLRepeater		Repeats output of all child components the specified number of times.

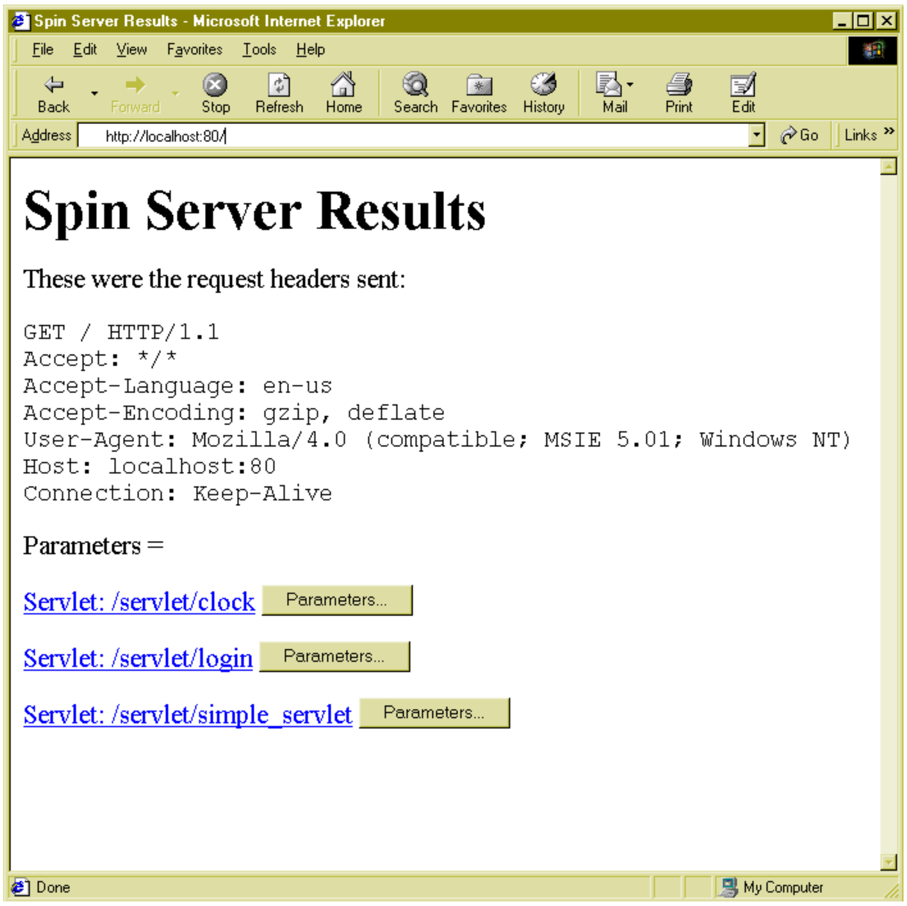
When you install Spin, it places the `SpinHTML.jar` file in the `beans` subdirectory of the `Spin` directory. These beans implement the Spin HTML components, as well as supporting classes and resources, especially component editors allowing you to set and change bean properties.

Spin's HTML components open a text stream and write the corresponding tag, or pairs of tags, in that part of the document corresponding to their place in the execution order of the servlet. HTML nesting corresponds to the child-parent hierarchy of the Spin components: for example, Spin nests a list item tag <LI> between a list start <UL> and list end </UL> tag when the HTMLListItem is a child of the HTMLList. You must ensure that the Spin hierarchy follows the rules of a proper HTML hierarchy; for example, all HTML components except HTMLDocument must be either direct children of HTMLDocument, or else more distant descendants.

In the editor for every HTML component, there is an HTMLElement tab. The settings in the editor provide another mechanism to change the flow of control in your application dynamically. By default, the tag for an HTML element is enabled, and so are the tags, if any, of its child elements. To suppress the output of the tag produced by an HTML component, set TagEnabled to false. To suppress the processing of any children of this HTML component, set TreeEnabled to false. Coupled with a conditional statement, these settings can be useful for producing different HTML under different conditions.

Set Up for Testing

For testing and debugging your servlet, as well as to run the examples in this chapter, Spin has a built-in web server. Its initial page provides a link to every Spin servlet currently open. Figure 2-2 shows Spin's built-in web server.


Spin Server Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address http://localhost:80/
Go Links
Done
My Computer

Spin Server Results

These were the request headers sent:

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT)
Host: localhost:80
Connection: Keep-Alive
```

Parameters =

Servlet: /servlet/clock Parameters...

Servlet: /servlet/login Parameters...

Servlet: /servlet/simple_servlet Parameters...

Figure 2-2: Spin's Built-in Web Server

To use the built-in web server, you need to set certain preferences. To enable Spin's built-in web server, follow these steps:

1. Edit > Preferences
2. Click the DebugServer tab.
3. Set WebServerEnabled to true.
4. By default, ServerPort is set to 80, the default port for web servers. If a web server is already running on your machine, use another value, such as 90 (for Windows or Macintosh) or 8080 (for UNIX).

Note: Do not use a port number that is being used by some other protocol.

5 In order to construct the initial page of links, Spin's web server needs to know where to look for running servlets. By default, `ServletAccessDirectory` is set to a virtual directory called `servlet`. This does not refer to any actual directory on the local file system; instead, it refers to only the name you type in the browser in order to open the initial web page. If you wish to run servlets from another location instead, specify its relative pathname here.

6 Spin also has to know where to look for images or other resources. By default, `URLOrDirectoryForNonServletRequests` is set to a directory called `public_html`, inside the Spin installation directory. If you wish to store such resources in another location instead, specify its URL or its full or relative pathname here. Relative pathnames are interpreted relative to the Spin installation directory. If you use a URL, it must start with `http://`.

7 For debugging purposes, Spin lets you pass parameters from the servlet to the browser. `NunberOfPostParameters` lets you specify the number presented to you through Spin's built-in web server parameters page. To run the examples, you do not need to change this.

8 To view any servlets you're working on, open your favorite web browser to this page:

```
http://localhost/
```

If you've changed the port number in step , add a colon and the correct port number. For example:

```
http://localhost:90/
```

Note: Some browsers misinterpret the name `localhost` and try to open `www.localhost.com`. If you encounter this behavior, use the IP address instead of the name; open your browser to 127.0.0.1

9 Click on the link to the servlet of interest to view the page that it is currently sending.

If you wish, you can run Spin's debugging web server on another computer. To do so in this case, open the browser to `http://`*`machineName`*`:80`, substituting the correct name for *`machineName`*.

The following sections show you how to build a simple servlet by:

- "Producing a Web Page"
- "Adding Dynamic Content"
- "Adding a Simple Form"
- "Using a Template"

Producing a Web Page

To produce a simple web page:

1 File > New Capsule

2 Capsule > Edit Properties

3 Under Type, select Java Servlet.

4 Insert > com.webgain.spin.html > HTMLDocument

5 Double-click the HTMLDocument icon to invoke its editor. The fields you see map to tags, generally those associated with HTML <BODY> tags. Navigate to the background color editor and click the radio button labeled Value, then double-click the background color to invoke the color editor. Choose a pleasant background shade. (Make sure it is light so that black text is visible on it, or else click on the text color and change that, too).

   In the Title: field, type `Hello Servlet` Click OK to dismiss the editor when you're done.

6 Insert > com.webgain.spin.html > HTMLParagraph. Make sure it is a child of the document.

7 Insert > com.webgain.spin.html > HTMLText. Make sure it is a child of the paragraph.

8 Double-click the HTMLText icon to invoke its editor. You'll see a field labeled Text:. Enter `Hello, world!`

   You have now built the nested structure of an HTML document, but have not yet caused anything to happen.

9 Select the HTMLDocument component.

10 Insert > User Behavior > ServletGet. Make sure it is a child of the document.

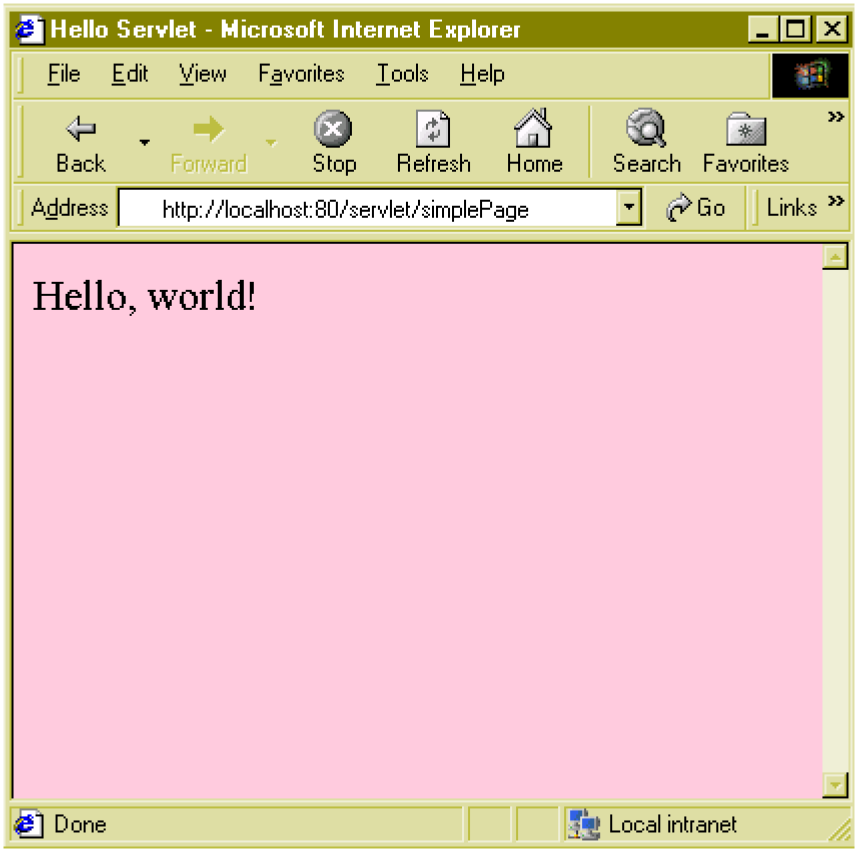
11 In response to a GET request, servlets must set their content type, then send their contents. When you expand the user behavior ServletGet, you see it consists of two actions to do these things. ServletGet is provided as a convenience, and an example of programming reusable components by using the `actor` and `capsule` aliases: you can add ServletGet without modification to any servlet, as long as you make the user behavior a child of the document.

   If it is not a child of the document, all you need to change is the action `sendDocument` — send its message to the specific document instead of to the alias `actor`.

12 Open the editor for ServletGet and examine its stimulus: it responds when the capsule generates the event `ServletRequest.doGet`. This event represents the servlet receiving a request of type GET from a browser. The capsule knows how to generate this event because we made it of type Java servlet.

13 Now is a good time to save your file. Choose File > Save As.... Name it `simplePage.zac`.

14 Now open a browser on `localhost`, and click the link for `simplePage` to see the document that contains one paragraph saying Hello, world! and the background color you chose.


Hello Servlet - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:80/servlet/simplePage
Go Links
Done
Local intranet

Hello, world!

Here's what happened:

1 You loaded the page in your browser.

2 The browser sent a GET request to the server, in this case, `localhost`, Spin's built-in web server.

3 The specified capsule `simplePage.zac` responded by generating the event `ServletRequest.doGet`.

4 The user behavior ServletGet triggered, generating the event `Behavior.activated`.

5 `SetContent` and `SendDocument`, its two children actions, both triggered on `parent.Behavior.activated`.

Because they both trigger on the same event, the capsule hierarchy's other flow of control mechanism comes into play: the action above executes first, then the one below it.

a `SetContent` tells the capsule (the servlet) that the response type is HTML text.

b `SendDocument` sends the document (actor) to the servlet's (the capsule's) output stream (a `printWriter` in Java terms).

To see the document you built, return to your web browser and view the page source:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<HTML>
<HEAD>
<TITLE>Hello Servlet</TITLE>
</HEAD>
<BODY BGCOLOR="#f9cadf"> (your color will vary)
<P>Hello, world!</P>
</BODY>
</HTML>
```

As you can see, it's built from the HTMLDocument, HTMLParagraph, and HTMLText components you inserted. Nice background, nice sentiment—but you could build the same page with a text editor. Now let's add some dynamic content.

Adding Dynamic Content

We can easily add content that changes:

1. Open a capsule view on `simplePage.zac` if it isn't already open.
2. Select the document component.
3. Insert > com.webgain.spin.html > HTMLParagraph. Make sure it is a child of the document.
4. Insert > com.webgain.spin.html > HTMLText. Make sure it is a child of the new paragraph.
5. Select the capsule.
6. Insert > com.webgain.spin.misc > clock. Make sure it is a child of the capsule. A clock is a component that responds with appropriate output to the message `getCurrentTime()`.
7. Rename it `clock`.
8. Reselect the text component.
9. Insert > Behavior > Action. Make sure it is a child of the text.
10. Edit the action as shown:

    a Set the stimulus:

    ```
    ActivateOn actor.WhenOutput.whenOutput
    ```

    The action's actor is the text component. This action triggers when the text is output.

    b Set the response:

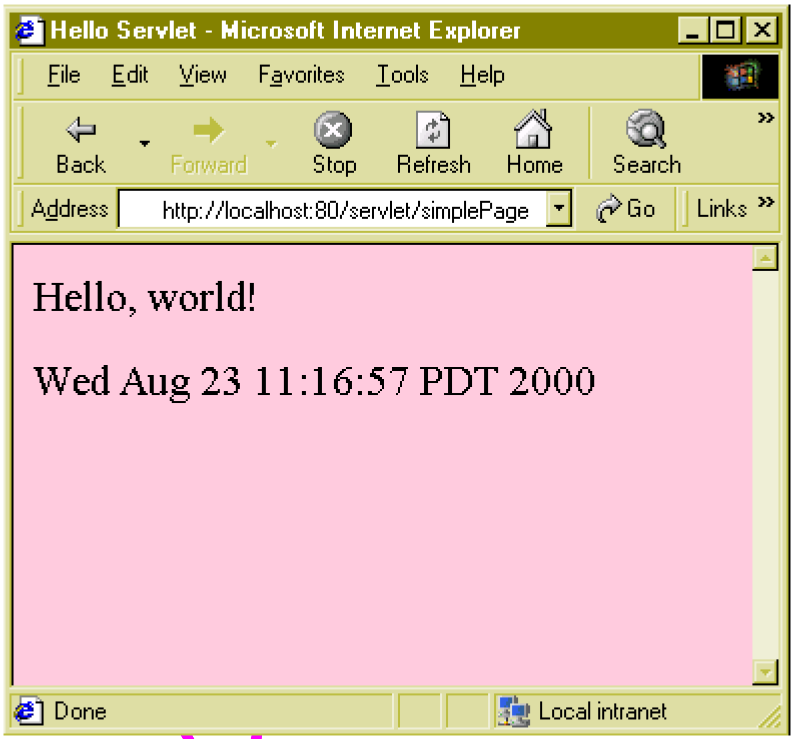
    ```
    Send To: actor
    Message: setText
    Data:(String)
    With=clock.getCurrentTime()
    ```

    Click on the button labeled **`With:`** to change it to **`With=`** and enable the popup menu with the appropriate messages.

    The action now sets its text to be whatever the clock outputs when it receives `getCurrentTime()`.

11. Save your work.

12 Now reload your browser to see:


Hello Servlet - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost:80/servlet/simplePage
Go Links
Done
Local intranet

Hello, world!

Wed Aug 23 11:16:57 PDT 2000

This time, the page source includes a second paragraph:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<HTML>
<HEAD>
<TITLE>Hello Servlet</TITLE>
</HEAD>
<BODY BGCOLOR="#f9cadf">
<P>Hello, world!</P>
<P>Wed Aug 23 11:16:57 PDT 2000</P>  (your time will vary)
</BODY>
</HTML>
```

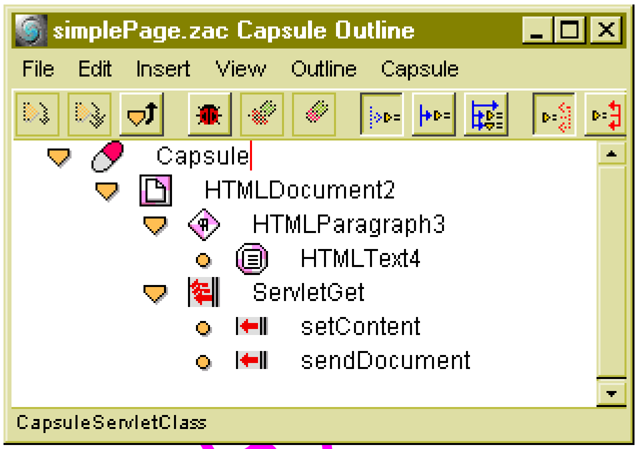
13 Reload the browser again to see the clock change. The content is now dynamic, but it’s still quite simple. Let’s enable some interaction with a simple form.

Adding a Simple Form

Next we'll add a simple form to the web page: one input field and a Submit **button.**

1 Open a capsule view on `simplePage.zac`, if one is not already open.

2 Delete the paragraph that is a parent of the clock, and all its children. Choose Actor > Delete. Your outline now appears as shown below:



```
Capsule
  HTMLDocument2
    HTMLParagraph3
      HTMLText4
    ServletGet
      setContent
      sendDocument
```

CapsuleServletClass

3 Select the document component.

4 Insert > com.webgain.spin.html > HTMLParagraph. Make sure it is a child of the document.

5 Insert > com.webgain.spin.html > HTMLText. Make sure it is a child of the paragraph you just inserted.

6 Select the document again. Insert > com.webgain.spin.html > HTMLHorizontalRule. Make sure it is a child of the document.

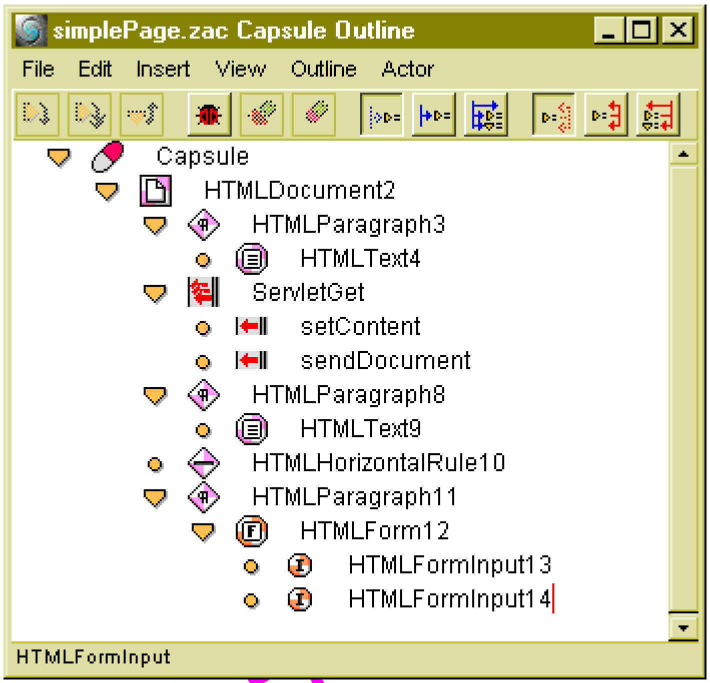
7 Select the document again. Insert > com.webgain.spin.html > HTMLParagraph. Make sure it is a child of the document.

8 Insert > com.webgain.spin.html > HTMLForm. Make sure it is a child of the paragraph you just inserted.

9 Insert > com.webgain.spin.html > HTMLFormInput. Make sure it is a child of the form you just inserted.

10 Select the form again. Insert > com.webgain.spin.html > HTMLFormInput. Make sure it is a child of the form.

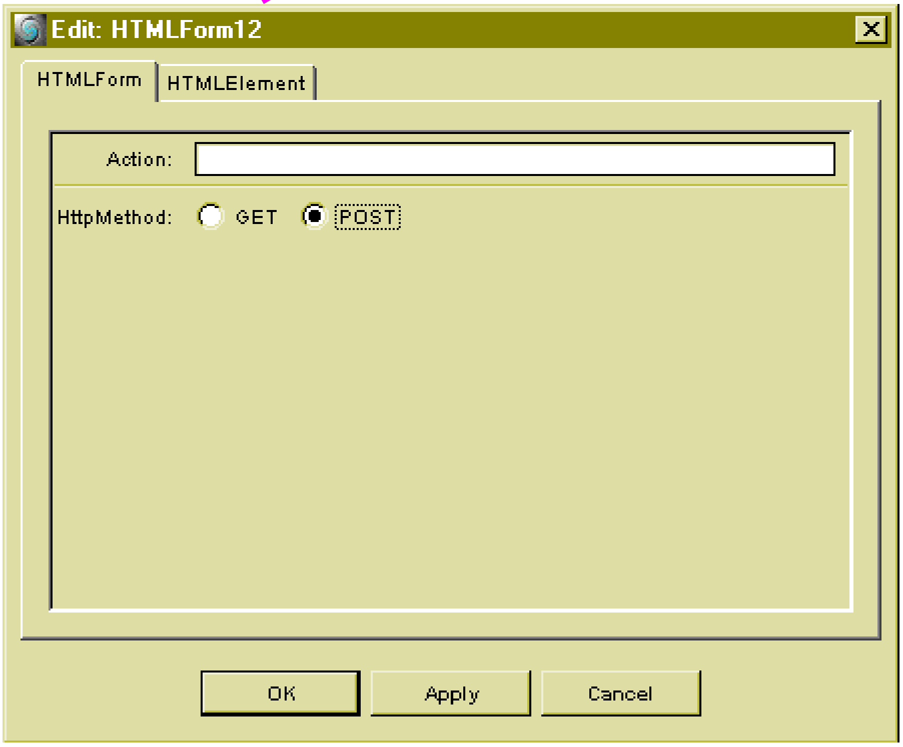
You've now added all the elements we'll need, and the outline appears as shown below:


simplePage.zac Capsule Outline
File Edit Insert View Outline Actor

```
Capsule
  HTMLDocument2
    HTMLParagraph3
      HTMLText4
    ServletGet
      setContent
      sendDocument
    HTMLParagraph8
      HTMLText9
    HTMLHorizontalRule10
    HTMLParagraph11
      HTMLForm12
        HTMLFormInput13
        HTMLFormInput14
```

HTMLFormInput

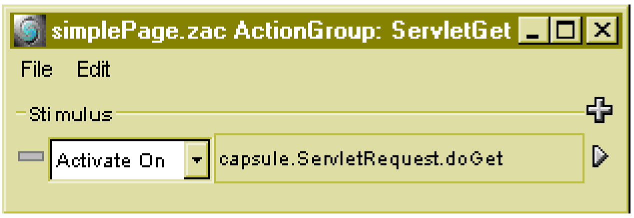
11 Double-click the HTMLForm icon to invoke its editor. It appears open to the HTMLForm tab. Check the radio button to change the HTTP method to POST:



Action:

HttpMethod: GET [✓ POST]

12 The form updates its page using the HTTP method POST, but our ServletGet behavior only produces output in response to a GET request type. We must correct this. Double-click the icon representing the ServletGet user behavior to invoke its editor.

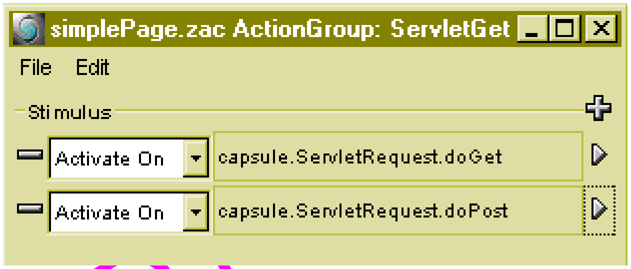


Activate On capsule.ServletRequest.doGet

13 Click the plus sign in the upper right corner to add another stimulus.

14 From the drop-down menu, add the stimulus:

```
ActivateOn capsule.ServletRequest.doPost
```



Activate On capsule.ServletRequest.doGet

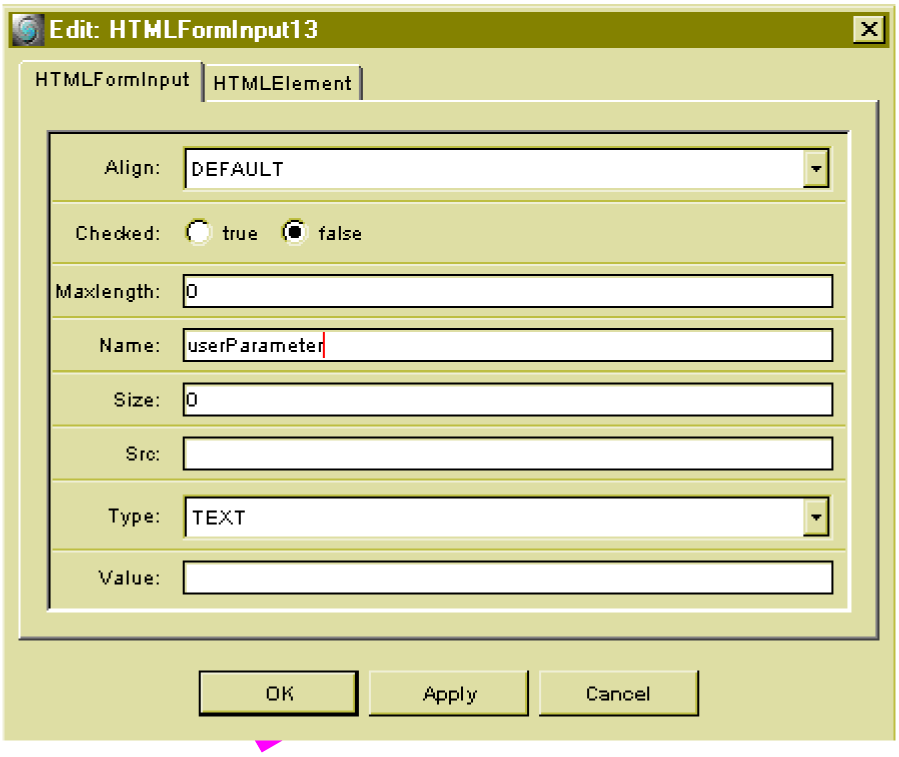
Activate On capsule.ServletRequest.doPost

15 Now the behavior responds to either a GET or a POST request type. Close the editor and rename ServletGet to ServletGetAndPost so its name reflects its new behavior. The two child actions do not need to change.

16 Now edit the topmost of the two form inputs you added. This one represents the text field that allows the user to enter some text.

- **a** Leave **Align** set to DEFAULT to specify a default alignment for the form field.
- **b** Leave **Checked** to false—the form field won't be checked.
- **c** Leave the **Maxlength** set to 0, specifying that there's no maximum length to the text that the user can enter.
- **d** The **Name** is the handle by which we'll be able to refer to the value that the user types in. Enter `userParameter`.
- **e** Leave the **Size** also set to 0, specifying the default length for the field.
- **f** Leave the **Src** empty as well. (If we were displaying, for example, an image in this form, here's where we'd specify the path to it.)
- **g** If necessary, set the **Type** to TEXT to make it a text field.

h Leave the **Value** empty, indicating no default text will appear in the field. The editor now appears as follows:



Align:	DEFAULT
Checked:	○ true [✓ ● false]
Maxlength:	0
Name:	userParameter
Size:	0
Src:	
Type:	TEXT
Value:	

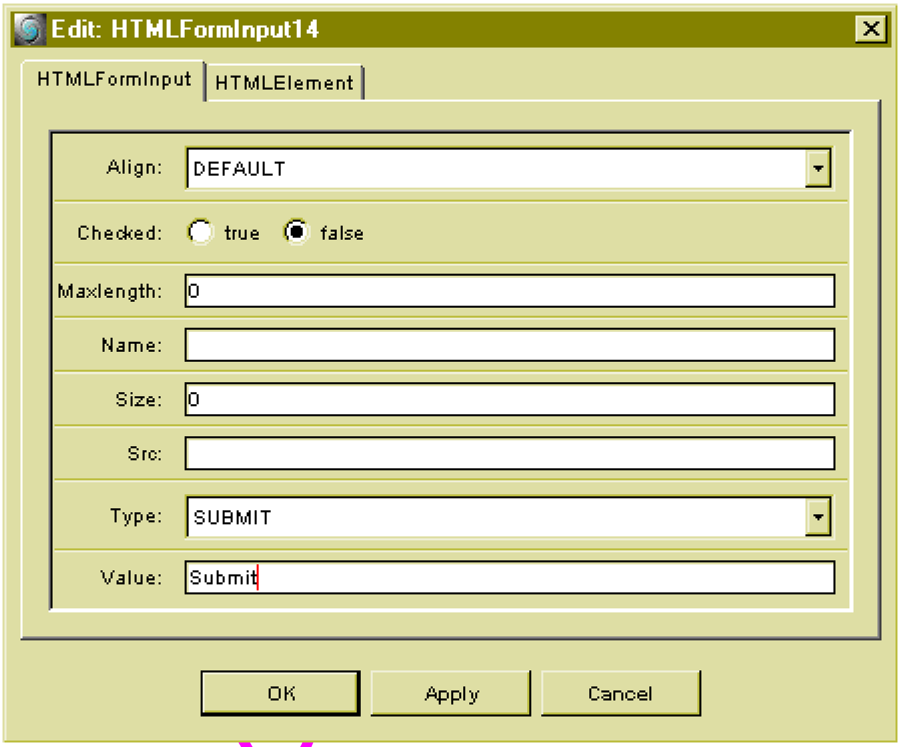
i Click OK to dismiss it.

17 Now edit the second of the two form inputs. This one represents the Submit button.

a Set the **Type** to SUBMIT.

b Set the value to Submit: this represents the button label.

c Leave all the other fields at their defaults. The editor now looks like this:



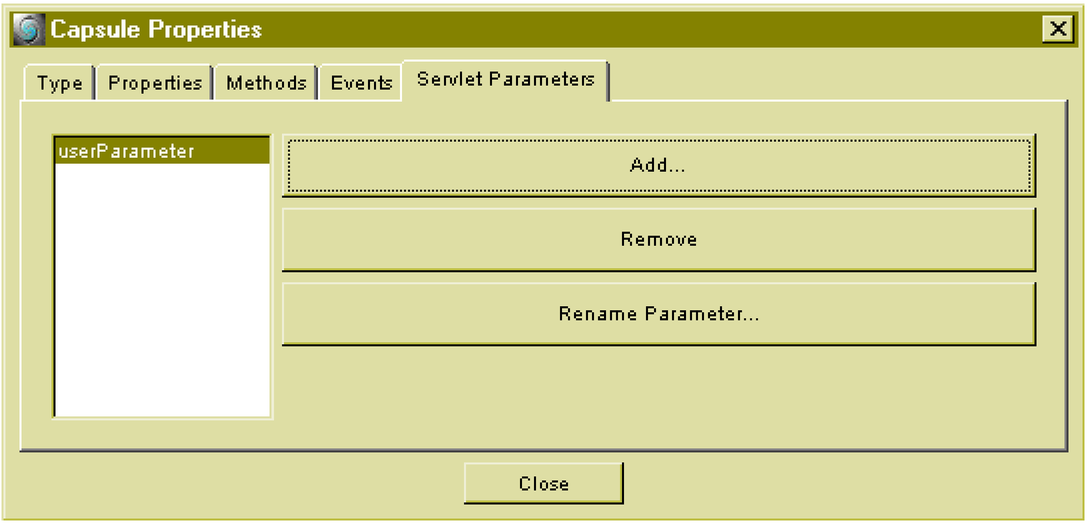
Align: DEFAULT
Checked: ( ) true [✓ false]
Maxlength: 0
Name:
Size: 0
Src:
Type: SUBMIT
Value: Submit

18 The form is ready; we now have to tell the servlet to expect a parameter. Select the capsule to get a Capsule menu, then choose Capsule > Edit Properties.

19 Click the tab labeled Servlet Parameters.

20 Click Add....

21 In the resulting dialog, enter the name you entered for the text field: `userParameter`. Make sure the two names match exactly, in both spelling and capitalization.



userParameter

22 Close both the dialog and the editor.

23 Only one final task remains, to get the second paragraph to print what the user entered. Select the text you inserted previously.

24 Insert > Behavior > Action. Make sure it is a child of the text component.

25 Edit the action so it appears as shown:

a Set the stimulus:

```
ActivateOn parent.WhenOutput.whenOutput
```

The action's parent is the text component. This action triggers when the text is output.

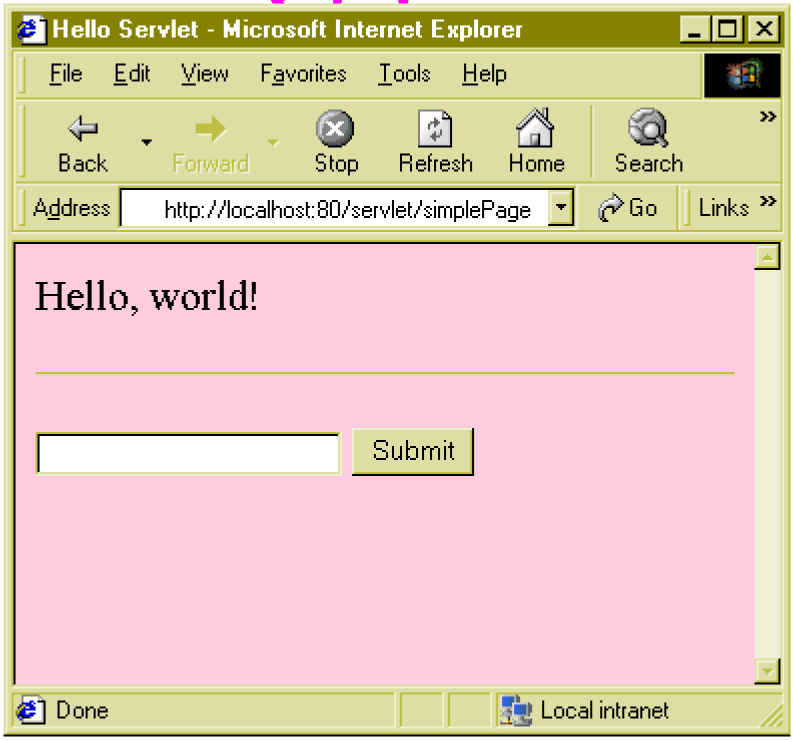
b Set the response:

```
Send To: actor
Message: setText
Data:(String)
With=capsule.getParameter("userParameter")
```

The action's actor is also the text component; this sets its text to be whatever the user enters into the text field.

26 Save your work.

27 Open a browser on `localhost`, if necessary, and click `simplePage`. (Or click the reload button, if it's already open.) You'll see:


Hello Servlet - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost:80/servlet/simplePage
Go Links
Hello, world!
Submit
Done
Local intranet

28 Enter some text into the text field and click Submit. The text you entered appears in the second paragraph, above the horizontal rule.

View the page source if you wish to see the document your capsule has now produced:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<HTML>
<HEAD>
<TITLE>Hello Servlet</TITLE>
</HEAD>
<BODY BGCOLOR="#f9cadf">
<P>Hello, world!</P>
<P>This is the forest primeval.</P> (your text will vary)
<HR WIDTH="100%">
<P><FORM METHOD="POST" ACTION="simple_servlet">
<INPUT TYPE="TEXT" NAME="userParameter">
<INPUT VALUE="Submit" TYPE="SUBMIT">
</FORM>
</P>
</BODY>
</HTML>
```

Using a Template

The above application has the virtue of simplicity, but it also has a significant drawback — it mixes page design with application logic. For such a small and simple application, this may not be a problem, but for larger and more complex applications, it is less confusing to separate presentation from application logic.

Even if a page design is relatively simple, a web site with dozens or even hundreds of pages is not. If most or all of these pages use just one or a few of the same basic designs, it is far more efficient to implement presentation in a template that you can reuse as needed.

CHAPTER 3

Accessing a Database

Data is only useful when people can access it. Web applications in particular typically gain great utility if they provide access to a database. For example, your employer probably maintains a Human Resources database which holds, among other data, the number of vacation hours you have accrued. To find out how many vacation hours you currently have, you could call the company HR department and ask. Depending on how busy they are, this could involve a time-consuming round of phone tag; even if they're not busy, this solution still requires a human being to look up the data and answer you. How much more efficient it is to log into the company Intranet and look up the data yourself.

Spin allows you to build applications that can access any database having a Java DataBase Connectivity (JDBC) driver. Spin also includes the JDBC-ODBC bridge, which allows Spin to access any database having an Open DataBase Connectivity (ODBC) driver. With both JDBC and ODBC, you can access nearly all existing databases.

To build a Spin application with database access:

1 Define database connections or pools of connections.
2 Build SQL statements with SQL components.
3 Build behaviors to execute the SQL statements and handle the results.

For your convenience in development and testing, Spin ships with a single-user copy of the PointBase relational database and example data. For documentation and examples, see the directory named `pointbase` in the Spin installation directory.

This chapter covers the following topics:

- Accessing the Built-in Database
- Building SQL Statements
- Executing SQL Statements
- Connecting to a Database
- For More Information

Accessing the Built-in Database

Spin comes bundled with an evaluation copy of the PointBase relational database server and associated client Java software for you to use while developing and testing database-enabled applications. This evaluation database includes example data from both PointBase and Spin.

Note: You must enable PointBase in order to run the database examples.

To use the PointBase server while working in Spin:

1. Edit > Preferences.
2. Click the General tab, if necessary.
3. Set the property `PointBaseServerLoadsAtStart` to `true.`
4. Restart Spin. The console displays the following message:

```
Server started, listening on port 9092, display level: 0 ...
```

To use the PointBase Server as a separate application:

1. Double-click the Pointbase Server icon:

   On Windows platforms, this icon is located in the `pointbase` folder inside the Spin installation folder, as well as in the program group you specified when you installed Spin (accessible from the Start menu).

   On the Macintosh, this icon is located in the `pointbase` folder inside the Spin installation folder, as well as in the Apple menu if you chose to have aliases placed there.

   On UNIX platforms, the script `PointBase Server.sh` is installed in the Spin installation directory. You must edit this script to conform with your Java installation before you can run the PointBase Server as a separate application.

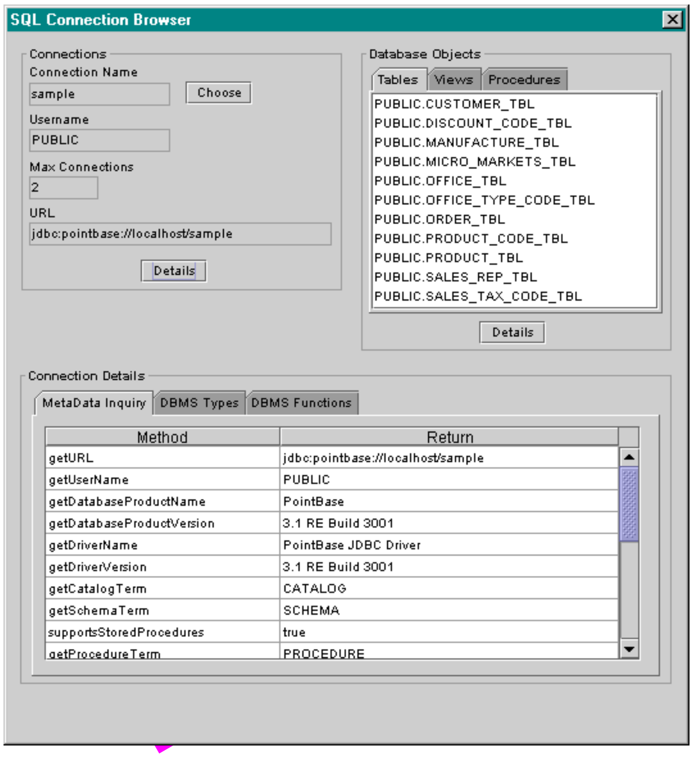
Note: To deploy a PointBase server in a released application, you must purchase a fully licensed version from PointBase.

Browsing Connections

Spin comes with several predefined connections to the built-in database, as well as certain example data. To view the properties of these predefined connections and the example data:

1. File > SQL Connections... . The Connection Browser appears.
2. Choose a connection a user name, and a URL.

3 In order for you to view connection properties and data, the Connection Browser connects to the database when you click the button labeled Details.



Connection Name: sample
Username: PUBLIC
Max Connections: 2
URL: jdbc:pointbase://localhost/sample

PUBLIC.CUSTOMER_TBL
PUBLIC.DISCOUNT_CODE_TBL
PUBLIC.MANUFACTURE_TBL
PUBLIC.MICRO_MARKETS_TBL
PUBLIC.OFFICE_TBL
PUBLIC.OFFICE_TYPE_CODE_TBL
PUBLIC.ORDER_TBL
PUBLIC.PRODUCT_CODE_TBL
PUBLIC.PRODUCT_TBL
PUBLIC.SALES_REP_TBL
PUBLIC.SALES_TAX_CODE_TBL

Method	Return
getURL	jdbc:pointbase://localhost/sample
getUserName	PUBLIC
getDatabaseProductName	PointBase
getDatabaseProductVersion	3.1 RE Build 3001
getDriverName	PointBase JDBC Driver
getDriverVersion	3.1 RE Build 3001
getCatalogTerm	CATALOG
getSchemaTerm	SCHEMA
supportsStoredProcedures	true
getProcedureTerm	PROCEDURE

Note: The Connection Browser is a convenience for viewing database properties. Your application will connect as specified in the Spin SQL component that calls the `connect()` method.

Building SQL Statements

Each of Spin's database components except SqlConnection builds a specific kind of SQL statement, which executes when a Spin behavior invoke its `execute()` method.

SqlConnection allows you to define a dedicated connection for circumstances in which connection-pooling is inappropriate. Table 3-1 lists the Spin database components. For more information, see "Using a Dedicated Connection" on page 27.

Table 3-1: Spin Database Components

Component	Icon	SQL Equivalent	Purpose
SqlDelete		DELETE	Deletes a row.
SqlInsert		INSERT	Inserts a row.
SqlSelect		SELECT	Searches the database.
SqlUpdate		UPDATE	Changes the data in existing row(s).
SqlRawStatement		Any of INSERT, DELETE, UPDATE, SELECT or CALL	Build statements with complex, unimplemented syntax.
SqlProcedureCall		CALL	Call a procedure stored in the database.
SqlConnection		Passes user name and password as part of connecting.	Encapsulates a connection to override default connection pooling.

When you install Spin, it places the `SpinSQL.jar` file in the beans directory in the Spin directory. These beans implement the Spin database components, as well as supporting classes and resources, especially component editors allowing you to set and change bean properties.

Spin's SQL components are intended to be lightweight and simple. Therefore, the delete, insert, select, and update components do not allow for all legal SQL syntax. To compensate, Spin provides two general-purpose components:

◆ The SqlRawStatement component allows you to directly enter any SQL statement that uses one of the operators INSERT, DELETE, UPDATE, SELECT or CALL.

◆ The SqlProcedureCall component allows you to call any procedure stored in the database, thus permitting any arbitrary computation.

Note: If you wish to use SqlProcedureCall, make sure your driver supports execution of stored procedure calls.

You build SQL statements by adding these database components to your application, editing each as necessary. Building a simple SQL statement includes some or all of the following steps:

1. Specify a connection.
2. Specify a table.
3. Specify columns if appropriate.
4. Specify conditions if necessary.
5. Check the syntax of the statement you're building.
6. Change component options, if appropriate.

The SELECT statements differ somewhat, and you might also specify joins, a sorting order for results, and column formats. SqlRawStatement and SqlProcedureCall differ considerably.

To add an SQL component to your application:

1 Open the capsule outline editor on your application, if you are not already using it.
2 Insert > com.webgain.spin.sql > Sql*Component*.

To open an editor on an SQL component:

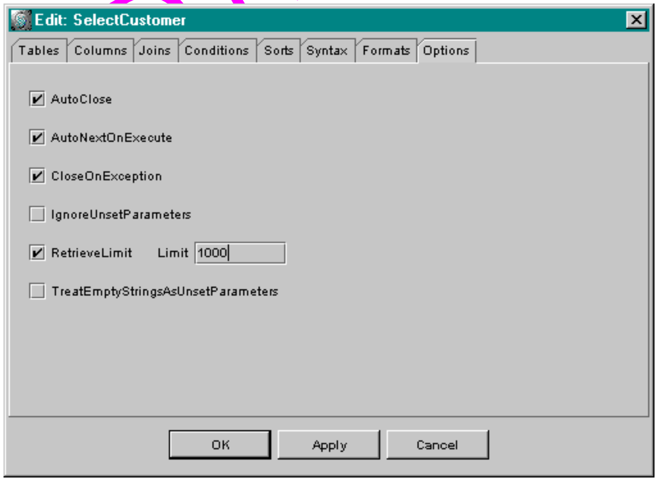
1 Open the capsule outline editor on your application, if you are not already using it.
2 Double-click the icon of the SQL component you wish to edit.

(You can also right-click to bring up a menu of more specific editing options for that component.)

Checking Options

Each component that builds an SQL statement supports from four to six options, described below. In order to be useful under the widest possible set of circumstances, default values vary by component and are discussed with the specific component.

1 Double-click the icon of the SQL component you wish to edit.
2 Click the Options tab. The Options editor appears.



[✓] AutoClose
[✓] AutoNextOnExecute
[✓] CloseOnException
[ ] IgnoreUnsetParameters
[✓] RetrieveLimit Limit 1000
[ ] TreatEmptyStringsAsUnsetParameters

`AutoClose` The application need not explicitly close the connection after an SQL component executes; Spin closes the connection transparently.

`AutoNextOnExecute`
The component automatically calls `next()` after `execute()`, repeatedly, until there's no more data or the retrieve limit is reached. Every row fetched generates an event before the component automatically calls `next()` again.

`CloseOnException`
If the statement encounters an exception, it closes the database connection and then

rethrows the exception. If this is not checked and an exception occurs, you must trap the exception and close the connection yourself.

`IgnoreUnsetParameters`

Rebuilds the SQL statement, omitting parameters with NULL values, useful if you want your application to allow incomplete input.

`RetrieveLimit`

Specifies whether you want SELECT statements to pause after they've retrieved the specified number of data items, or to continue until they've retrieved every item that meets the specified criteria.

`TreatEmptyStringsAsUnsetParameters`

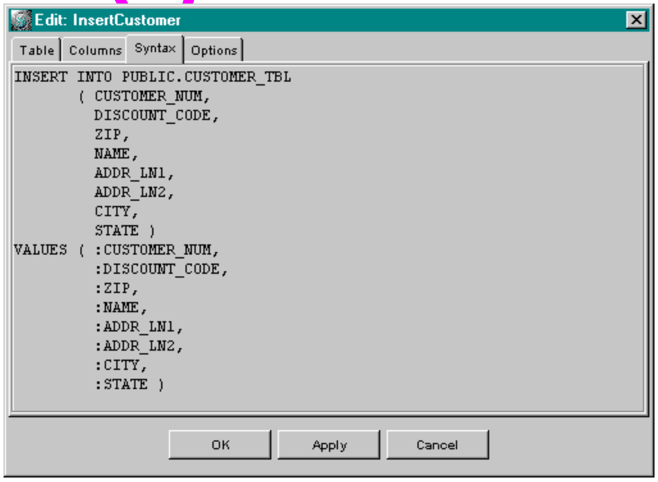
Transforms empty strings `" "` to NULL values, useful for validating input that comes, for example, from HTML forms.

Checking Syntax

As you edit an SQL component, you're building an SQL statement. You can check your progress at any time.

To view the statement you are building:

1. Double-click the icon of the SQL component you wish to edit.
2. Click the Syntax tab. Your component appears something like:



```
INSERT INTO PUBLIC.CUSTOMER_TBL
       ( CUSTOMER_NUM,
         DISCOUNT_CODE,
         ZIP,
         NAME,
         ADDR_LN1,
         ADDR_LN2,
         CITY,
         STATE )
VALUES ( :CUSTOMER_NUM,
         :DISCOUNT_CODE,
         :ZIP,
         :NAME,
         :ADDR_LN1,
         :ADDR_LN2,
         :CITY,
         :STATE )
```

Items with colons (:) in front are named parameters to be replaced with actual values when Spin sets them.

The section below walks you through building a simple SQL statement with SqlInsert, SqlDelete, or SqlUpdate. Next, SqlSelect statements are described, followed by SqlRawStatement and SqlProcedureCall.

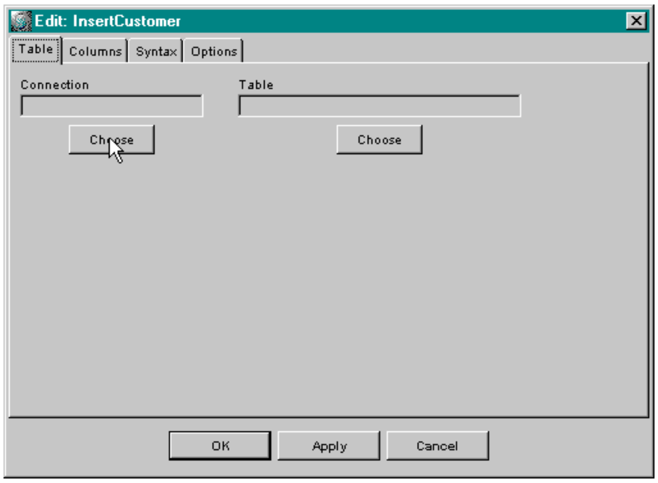
SqlConnection is discussed in "Using a Dedicated Connection" on page 27.

Building Simple Statements

The SqlDelete, SqlInsert, and SqlUpdate are the simplest components; after you build them, invoke their `execute()` method and you're done.

Note: If you have unchecked the options `AutoClose` or `CloseOnException` (checked by default), you'll also have to invoke the component's `close()` method, in the first case after normal operation, and in the second after catching an exception.

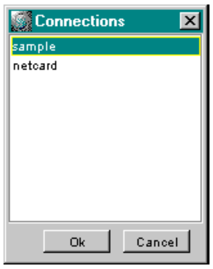
Double-click on the icon for SqlInsert and you'll see an editor like the one below:



Specify a Connection

First, you must specify the data source your application will use. The editor opens on the tab you'll need to do so, marked Tables.

a Click Choose under the Connection field. A dialog appears, containing a list of the defined connections. (These are actually pools of connections, defined in the `spinsql.properties` file described on page 26.)



b Highlight the desired connection and click OK.

The connection you selected appears in the Connection field.

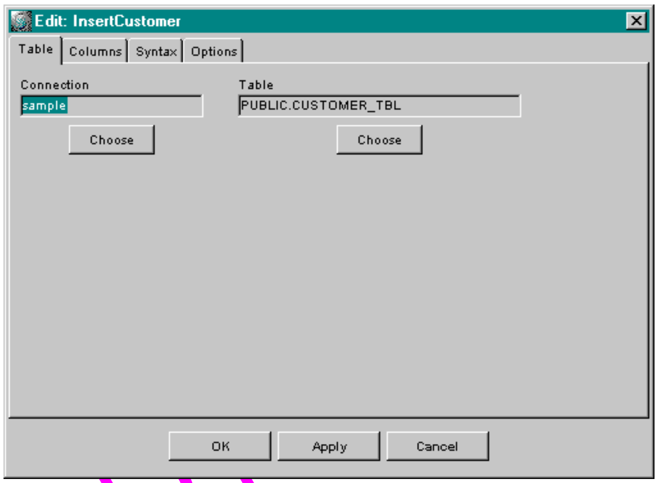
Specify a Table

Next, you must specify the table that holds the data this statement must access. In the Tables tab:

a Click Choose under the Table field. A dialog appears, containing a list of the tables defined in the specified database.

b Highlight the desired table and click OK.

The table you selected now appears in the Table field, and your editor now looks something like:



Connection: sample

Table: PUBLIC.CUSTOMER_TBL

Specify Columns

INSERT and UPDATE statements need columns to which to write their data; SELECT statements need columns to retrieve. To specify the column(s):

1 Click on the Columns tab.

2 Select a column in the Available Columns list, and click the rightward-pointing arrow to move it into the list of columns affected by the SQL statement.

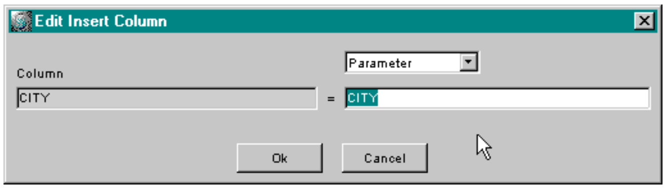
3 Repeat until you have moved all the columns that you require. If you move one by mistake, click the leftward-pointing arrow to move it back.

For SqlInsert and SqlUpdate to perform any useful work, you must provide column values. You can set the value of a column to any of three kinds of things:

- ❖ the value of another column (for SqlUpdate only),
- ❖ any arbitrary value, such as a string, as expressed in SQL, or
- ❖ a parameter passed by a Spin behavior.

By default, a Spin application uses this last mechanism to set a column value. Again by default, each column is assigned a parameter name equal to the column name. Your running application can then set the column's value when a Spin behavior executes a method named `setParameterXX()`, substituting your actual parameter name for `ParameterXX`. (This is discussed in greater detail in "Passing Parameters" on page 23.)

4 If you wish to use another mechanism to set column values, select the appropriate column and click Edit to invoke the following dialog:



Note: Use the default mechanism to set the column value if your SqlUpdate component will include a condition that refers to this parameter.

5 If you wish to change the parameter name, select it and type the desired name.

Otherwise, access the pulldown menu to change Parameter to SQL Syntax or (if you're editing an SqlUpdate component) Column.

a If you've chosen SQL Syntax, enter the SQL text in the edit field.

If you've chosen Column, select the column name from the drop-down menu.

b Click OK when you're done.

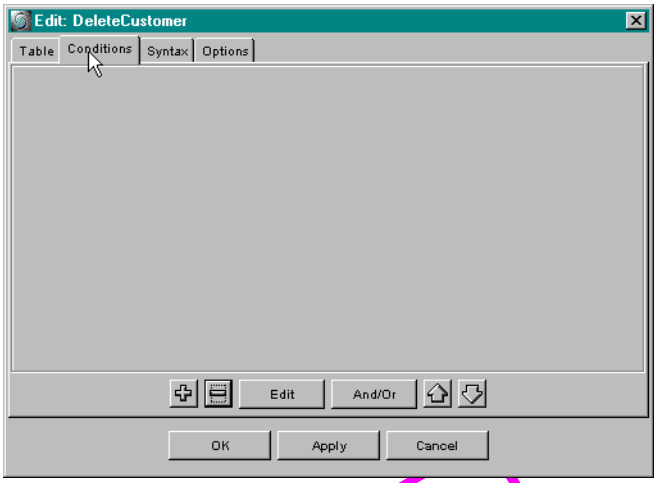
Specify Conditions

SqlDelete, SqlUpdate, and SqlSelect can build statements that include conditions—WHERE clauses, in SQL terms—that you can use to control the scope of delete, update, or select operations.

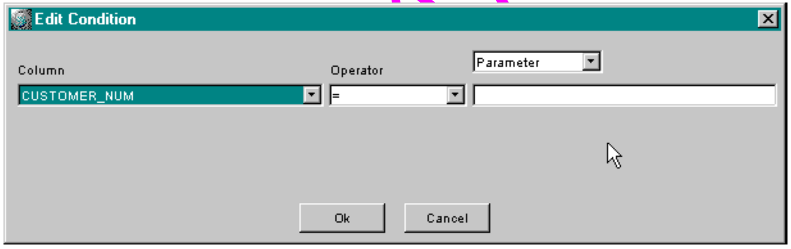
Note: If you build a condition that refers to named parameters, set the value of those parameters using the default mechanism of calling `setParameterXX()` methods; set the corresponding column values using parameters whose names are identical to the column names.

To specify a condition:

1 Click on the Conditions tab. The following appears:



2 Click the add button to invoke the condition editor and add a condition:



Column: CUSTOMER_NUM
Operator: =

A conditional expression consists of a left-hand operand, an SQL operator, and a right-hand operand. The left-hand operand is always a column. The right-hand operand can be either a named parameter, another column, or arbitrary SQL text. The following operators are supported:

`=`	equals
`<>`	does not equal
`<`	less than
`<=`	less than or equal to
`>`	greater than
`>=`	greater than or equal to
`LIKE`	matches string
`NOT LIKE`	does not match
`IN`	is in a set of values
`NOT IN`	is not in a set of values
`IS NULL`	column contains no value
`IS NOT NULL`	column contains a value

3 In the leftmost field, select a column from the pulldown menu.

4 In the middle field, specify an operator.

5 If you wish to use a parameter, type the desired parameter name in the rightmost field.

Otherwise, access the pulldown menu above that field to change Parameter to SQL Syntax or Column.

- **a** If you've chosen SQL Syntax, enter the SQL text in the edit field.

  If you've chosen Column, select the column name from the drop-down menu.
- **b** Click OK when you're done.

6. Click the add button again if you wish to add another expression to the WHERE clause you're creating.
7. Repeat steps as necessary.
8. If you have added more than one conditional expression, select the topmost. By default, it is joined with the next by the word AND. Click And/Or to toggle this to an OR if you wish.
9. Repeat the previous step until all the conditional expressions are strung together with either AND or OR, as required.

Table 3-2: SqlInsert, SqlDelete, and SqlUpdate Options

Option	Default
AutoClose	false
CloseOnException	true
IgnoreUnsetParameters	false
TreatEmptyStringsAsUnsetParameters	false

SqlSelect

SqlSelect components return data to the Spin capsule one row at a time, for each row generating a `rowRetrieved` event. Add behaviors that are activated by `rowRetrieved` when your application needs to act on this data.

After all behaviors triggered by the `rowRetrieved` event have executed, if AutoNextOnExecute is enabled, the SqlSelect component processes the next row of data and generates a new `rowRetrieved` event. After it retrieves the last row, the component generates a `noMoreRows` event.

If the RetrieveLimit option is set, then when the limit is reached, SqlSelect generates a `retrieveLimitReached` event.

SqlSelect allows you to specify more than one table to use for joins, the joins themselves, a sorting order for the results, and the format of columns.

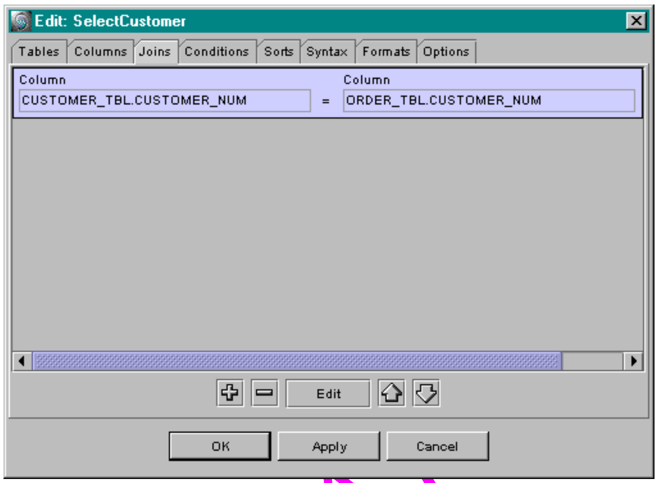
Choose Tables

To specify the tables, follow the same procedure described in "Specify a Table" on page 8, clicking the add button ⊞ to add an additional table, or the delete button ⊟ to remove one.

Specify Joins

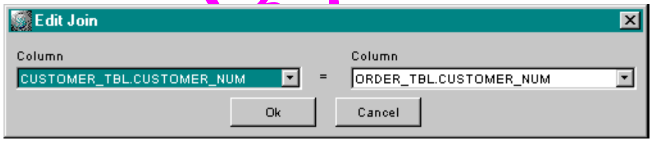
To specify a join:

1 Click on the Joins tab. The following appears:



Column		Column
CUSTOMER_TBL.CUSTOMER_NUM	=	ORDER_TBL.CUSTOMER_NUM

2 Click the add button to invoke the joins editor and add a join:



Column		Column
CUSTOMER_TBL.CUSTOMER_NUM	=	ORDER_TBL.CUSTOMER_NUM

3 Use the pulldown menus to select the two columns to join. When you've made the join you require, click OK.

4 Repeat the previous two steps until you've made all the required joins.

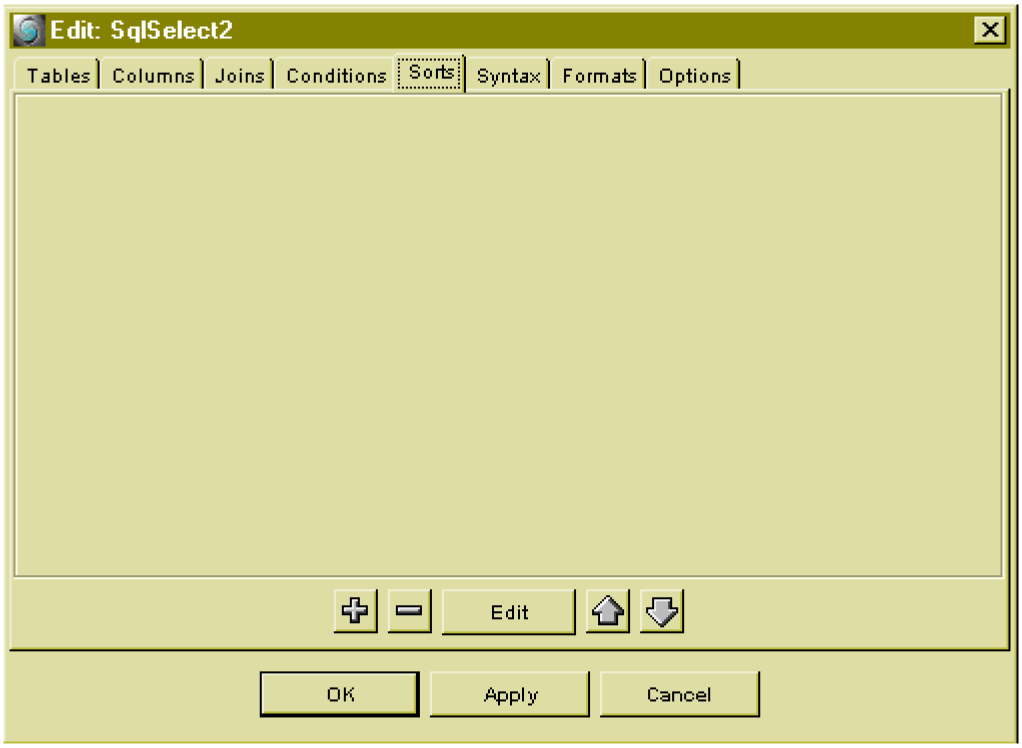
If an existing join is unsatisfactory, select it and click Edit to change it.

Click the delete button to remove one.

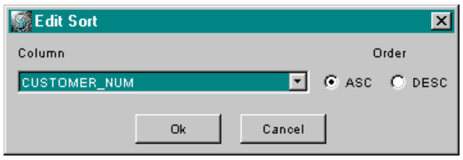
Specify Sorting

To specify a sorting order in which to return results:

1 Click on the Sorts tab. The following appears:



2 Click the add button ⊕ to invoke the sort editor and add a sort:



Column: CUSTOMER_NUM
Order: [✓ ASC] DESC

3 Use the pulldown menu to choose the column you wish to sort.

4 Click ASC for an ascending order or DESC for a descending order. Data is sorted according to the specific database implementation. Click OK when you're done.

5 To change an existing sort, click Edit.

Format Columns

The format of a column consists of two things:

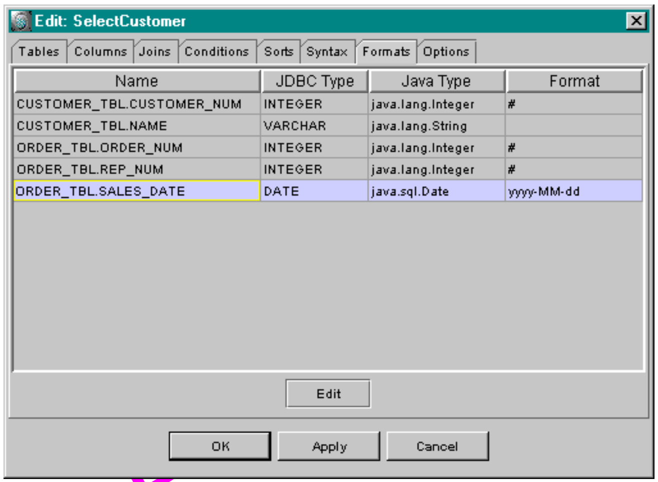
- the Java type that the component returns, and
- the form of a string, if any results are converted to a string.

The JDBC driver understands each datum in the database as a specific JDBC type. Spin's SQL components automatically convert the JDBC type to a Java type. You can change this to a different type, if your application requires it.

If a column result is converted to a string, either when it is retrieved, or later by calling a method such as `getColumnString()` (available in SqlSelect, SqlRawStatement, and SqlProcedureCall), you can also specify the format of the resulting string.

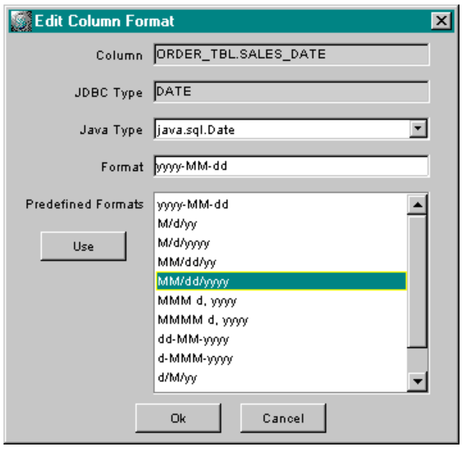
To do either of these things:

1 Click the Formats tab. The following appears, showing all the columns that return results:



Name	JDBC Type	Java Type	Format
CUSTOMER_TBL.CUSTOMER_NUM	INTEGER	java.lang.Integer	#
CUSTOMER_TBL.NAME	VARCHAR	java.lang.String	
ORDER_TBL.ORDER_NUM	INTEGER	java.lang.Integer	#
ORDER_TBL.REP_NUM	INTEGER	java.lang.Integer	#
ORDER_TBL.SALES_DATE	DATE	java.sql.Date	yyyy-MM-dd

2 Select the column you wish to change, and click Edit. The format editor appears:



Column	ORDER_TBL.SALES_DATE
JDBC Type	DATE
Java Type	java.sql.Date
Format	yyyy-MM-dd
Predefined Formats	yyyy-MM-dd M/d/yy M/d/yyyy MM/dd/yy MM/dd/yyyy MMM d, yyyy MMMM d, yyyy dd-MM-yyyy d-MMM-yyyy d/M/yy

3 Use the pulldown menu to change the Java type, if appropriate.

Note: The default Java type is appropriate under most circumstances.

Select a predefined string format from the bottom list and click Use, or enter your desired format directly in the format field using standard Java notation.

Table 3-3: SqlSelect Options

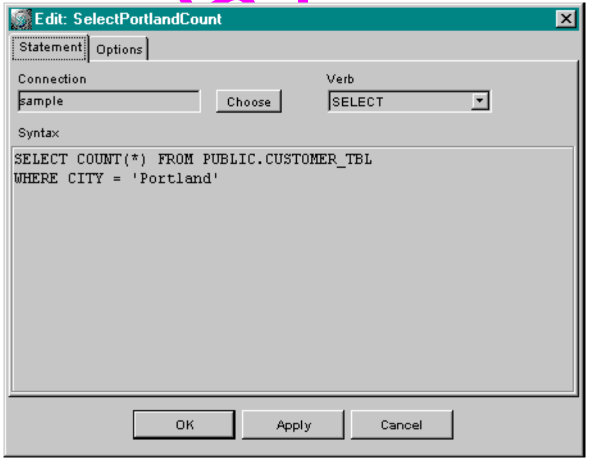
Option	Default
AutoClose	true
AutoNextOnExecute	true
AutoCloseOnException	true
IgnoreUnsetParameters	false
RetrieveLimit	false
TreatEmptyStringsAsUnsetParameters	false

SqlRawStatement and SqlProcedureCall

As mentioned earlier, SqlInsert, SqlDelete, SqlUpdate, and SqlSelect focus on efficiency and ease of use, and therefore do not implement all possible SQL syntax. When you need functionality they do not provide, use SqlRawStatement to enter any arbitrary SQL statement, or SqlProcedureCall to call any arbitrary stored procedure call.

To enter any SQL statement you can type:

1 Double-click the SqlRawStatement icon. The editor appears, open to the Statement tab:



Connection: sample

Verb: SELECT

Syntax

```
SELECT COUNT(*) FROM PUBLIC.CUSTOMER_TBL
WHERE CITY = 'Portland'
```

2 See “Specify a Connection” on page 7.

3 Choose the required verb from the pulldown menu, one of INSERT, DELETE, UPDATE, SELECT or CALL. The verb appears in the edit field below.

4 In the edit field, type the rest of your statement.

Table 3-4: SqlRawStatement Options

Option	Default
AutoClose	false
CloseOnException	true
IgnoreUnsetParameters	false
TreatEmptyStringsAsUnsetParameters	false

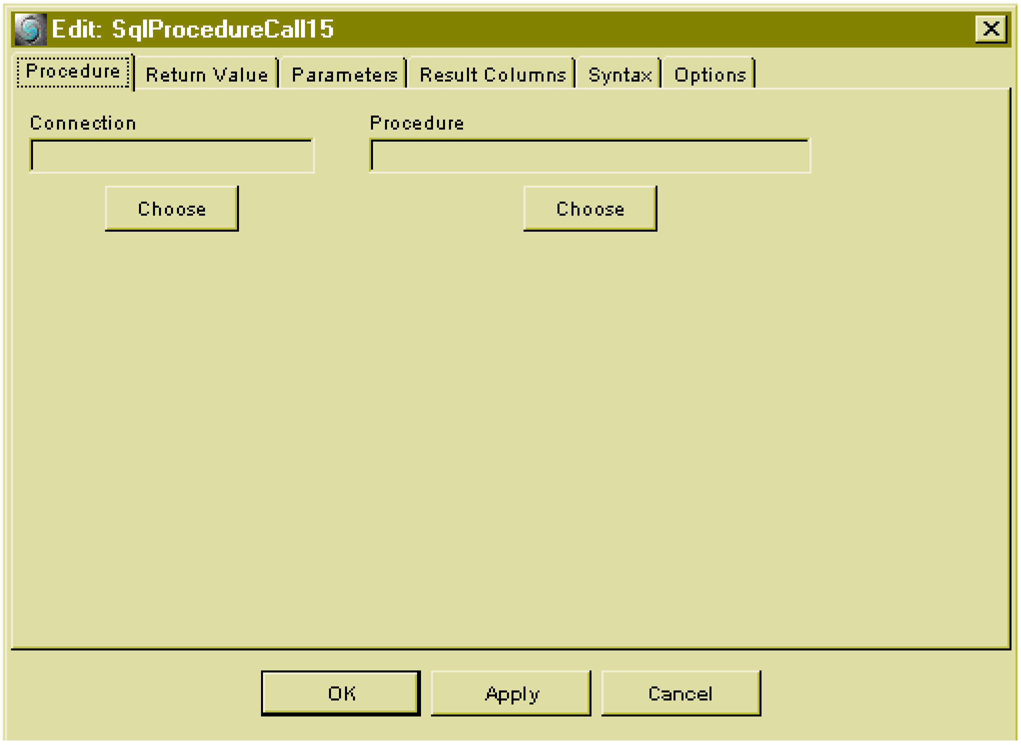
To integrate a stored SQL procedure into a Spin application, Spin needs to know these things about the procedure:

- its name,
- return type,
- a full description of the parameters, and
- the format of the result columns.

Note: Consult the appropriate documentation to ensure that the your JDBC driver supports the execution of stored procedure calls.

To call a stored SQL procedure:

1 Double-click the SqlProcedureCall icon. The editor appears, open to the Procedure tab:



2 Specify the connection. See “Specify a Connection” on page 7.

3 Choose the stored procedure in the same manner.

4 Click the Return Value tab. Enter the SQL type returned from the SQL procedure, and the JDBC type to which to convert it.

5 Click the Parameters tab. You’ll see a list of each parameter that passes a value to the stored procedure (`in`), each parameter that passes a value back from the database (`out`), and each parameter that does both (`inout`).

Resize the columns or change their order by placing your mouse cursor over the column labels, or the boundaries between them, and dragging.

6 Click the Result Columns tab. Specify the format of the result columns.

7 Check syntax and change options settings as required.

Table 3-5: SqlProcedureCall Options

Option	Default
AutoClose	true
AutoNextOnExecute	true
CloseOnException	true
RetrieveLimit	false

Building SQL Statements - Additional Tools

All the SQL statements you have built so far involve use of the graphical customizer that appears when you double-click on an SQL component. For users who would like to build SQL statements using generic properties, the standard Spin property editors may be used. To access these editors, right-click on an SQL component and select an option from the resulting menu.

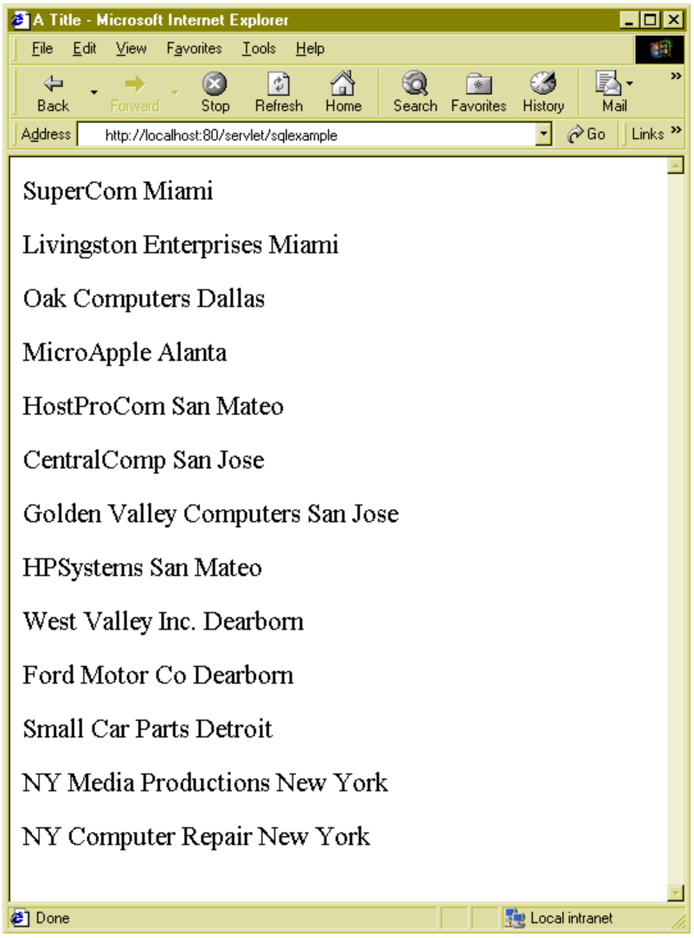
All the properties of the SQL components are directly accessible using string properties. The primary benefit of this capability is to allow Spin users to create Spin applications that dynamically configure components during runtime.

Executing SQL Statements

This section discusses:

- causing execution,
- retrieving results,
- passing parameters, and
- using a dedicated connection.

First, we'll give a general rule for accomplishing these things, then we'll run through an example to show it in the context of an application. The example is very simple: it opens an HTML page in your browser and displays a list of customer names and cities, as shown below:


A Title - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail
Address http://localhost:80/servlet/sqlexample
Go Links
Done
Local intranet

SuperCom Miami

Livingston Enterprises Miami

Oak Computers Dallas

MicroApple Alanta

HostProCom San Mateo

CentralComp San Jose

Golden Valley Computers San Jose

HPSystems San Mateo

West Valley Inc. Dearborn

Ford Motor Co Dearborn

Small Car Parts Detroit

NY Media Productions New York

NY Computer Repair New York

To keep these examples short and to the point, it is helpful to set up the basic application now. To do so:

1. Start Spin, if necessary.
2. Edit > Preferences to allow yourself to open a browser on your application.
   a. Click the DebugServer tab.
   b. Check true for WebServerEnabled.
   c. Check the other settings.
3. File > New Capsule
4. Capsule > Edit Properties
5. In the Type tab, set it to Java Servlet.
6. Insert > com.webgain.spin.sql > SqlSelect
7. Name it SelectStatement.

8 Configure the SqlSelect component you just added:

   a In the Tables tab, set Connection to sample.

   b Set Selected Tables to PUBLIC.CUSTOMER_TBL.

   c Click the Columns tab.

   d Move all the columns into the Selected Columns list. (Repeatedly click the right-pointing arrow.)

   e Click the Options tab.

   f To keep things simple, check IgnoreUnsetParameters. Set a retrieve limit if you like.

   g Check syntax and formats if you wish.

9 Select the capsule to assure that it will become the parent of the item you add next.

10 Insert > com.webgain.spin.html > HTMLDocument

   Spin includes components that implement HTML entities. You've just added an HTML document to your application.

11 If necessary, select the document you just added to assure that it will become the parent of the next item.

12 Insert > User Behavior > ServletGet

13 The user behavior adds two new actions. Examine them.

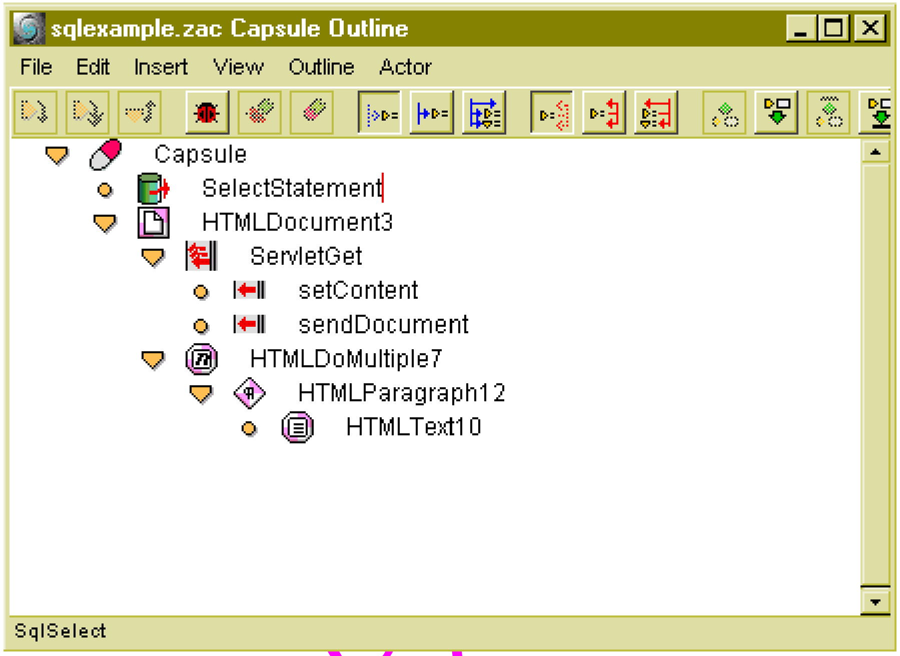
14 Still with the document selected as parent:
   Insert > com.webgain.spin.html > HTMLDoMultiple

   You've just added an HTML component that can contain a number of other entities. It will hold the paragraphs with the customer list, as well as blank paragraphs for white space.

15 With the HTMLDoMultiple selected as parent:
   Insert > com.webgain.spin.html > HTMLParagraph

16 With the HTMLParagraph selected as parent:
   Insert > com.webgain.spin.html > HTMLText

Your capsule now appears as below (the numbers appended to your entities may differ):



Causing Execution

To execute a statement, create a Spin Behavior that invokes its component's `execute()` method. For SqlDelete, SqlInsert, and SqlUpdate components with default options, nothing more needs to be done.

- If you've unchecked `AutoClose`, you must invoke the component's `close()` method. If you've unchecked `CloseOnException`, you must handle exceptions explicitly.
- If you've unchecked `AutoNextOnExecute`, you must invoke the component's `next()` method.

Return to the example you started above to add a simple invocation of `execute()`:

1 With HTMLDoMultiple selected as parent:
Insert > Behavior > Action

2 Rename it onOutputSelectExecute.

3 Edit it.

a Set stimulus:

```
ActivateOn actor.WhenOutput.whenOutput
```

b Set response:

```
Send To: SelectStatement
Message: execute
```

Retrieving Results

SqlSelect components generate a `rowRetrieved` event; to retrieve values from an SqlSelect component, create a Spin Behavior activated by `rowRetrieved`. That behavior, or others triggered with it, can retrieve data from the row one column at a time, using `getColumn`*`XX`*`()` methods.

Or you can retrieve an entire row at once using `getColumnValuesAsHashtable()` and use that to fill in equivalently named values in a form, using the HTML components.

SqlSelect components also generate a `noMoreRows` event; for special processing after the last row is retrieved, make a behavior activated by `noMoreRows`. For the SqlSelect component in the example, we don't need to: its `AutoClose` option is enabled, and there's nothing else we need to do.

Return to the example to add a simple retrieve operation:

1 With HTMLDoMultiple selected as parent:
Insert > Behavior > Action

2 Rename it onRowRetrievedPrint.

3 Edit it.

a Set stimulus:

```
ActivateOn SelectStatement.Sql.rowRetrieved
```

b Set response:

```
Send To: actor
Message: printChildren
```

The actor, in this case, is the parent: HTMLDoMultiple. Its child is the paragraph.

To set the content—the text that prints:

1 With HTMLText selected as parent:
Insert > Behavior > Action

2 Rename it setTextWhenOutput

3 Edit it.

a Set stimulus:

```
ActivateOn actor.WhenOutput.whenOutput
```

b Set response:

```
Send To: actor
Message: setText
Data:(String)
With=SelectStatement.getColumnString("NAME") + " " +
SelectStatement.getColumnString("CITY")
```

To create the final field:

a Click the **`With:`** field to make it become a **`With=`** field.

b Add the first part from the pulldown menu:
`SelectStatement.getColumnString("NAME")`

c Select the text you added and copy it (Control-C or Command-C).

d Deselect the text, place the cursor at the end, and type:

```
 + " " +
```

Include spaces before the first `+` and after the last.

e Paste the text you selected earlier.

f Change `NAME` to `CITY`.

Once again, the actor is the parent: HTMLText.

You can now open a browser on `localhost`, as instructed in "Set Up for Testing" on page 8, and click on your example servlet to see the results, as illustrated on page 19.

Passing Parameters

SqlInsert or SqlUpdate components can set column values in several ways, but one of the most useful is to do so with a Spin parameter. SqlDelete, SqlUpdate, or SqlSelect can specify a condition, for which a Spin parameter can serve as the right-hand operand.

Before executing the SQL statement, set the parameter by invoking a method such as `setParameterString("CITY", "Portland")`.

You can set all parameters at once:

1 Name the Spin parameters (for example, those coming from an HTML form) exactly the same names as the columns of the SQL table (the default).

2 Set the parameters by invoking the method `setMatchingParameterValues()` (part of the superclass SQLProgrammedStatement). It takes a hash table as an argument—a whole row.

To aid in validating input, enable the option `TreatEmptyStringsAsUnsetParameters` on your component. This option transforms empty strings `" "` to NULL values. Your SQL statement can then throw an exception if certain parameters are not supplied.

Return to the example to pass a parameter:

1 Double-click the SqlSelect icon to invoke its editor.

2 Click the tab labeled Conditions.

3 Click the plus sign to add a condition.

4 Under Column, choose CITY.

5 Under Operator, choose = (it's the default).

6 Make sure the rightmost pulldown menu is set to Parameter. In the field beneath it, type `city`.

7 Click OK twice to dismiss the condition editor and the component editor.

8 With HTMLDoMultiple selected as parent:
Insert > Behavior > Action

9 Rename it setCity.

10 Still with the setCity behavior selected:
Outline > Move > Up (Control-U or Command-U)
Outline > Move > Up

Do this twice so that the parameter is set before executing the SqlSelect statement.

11 Now edit the setCity behavior:

a Set stimulus:

```
ActivateOn actor.WhenOutput.whenOutput
```

b Set response:

```
Send To: SelectStatement
Message: setParameterString("CITY",?)
Data: (String, String)
With:CITY
With=capsule.getParameter("city")
```

To make the last line:

a Click `With:` to change it to `With=`

b Choose `capsule.getParameter(String)`

c Select `String` and replace it by typing `"city"`.

12 Save your work.

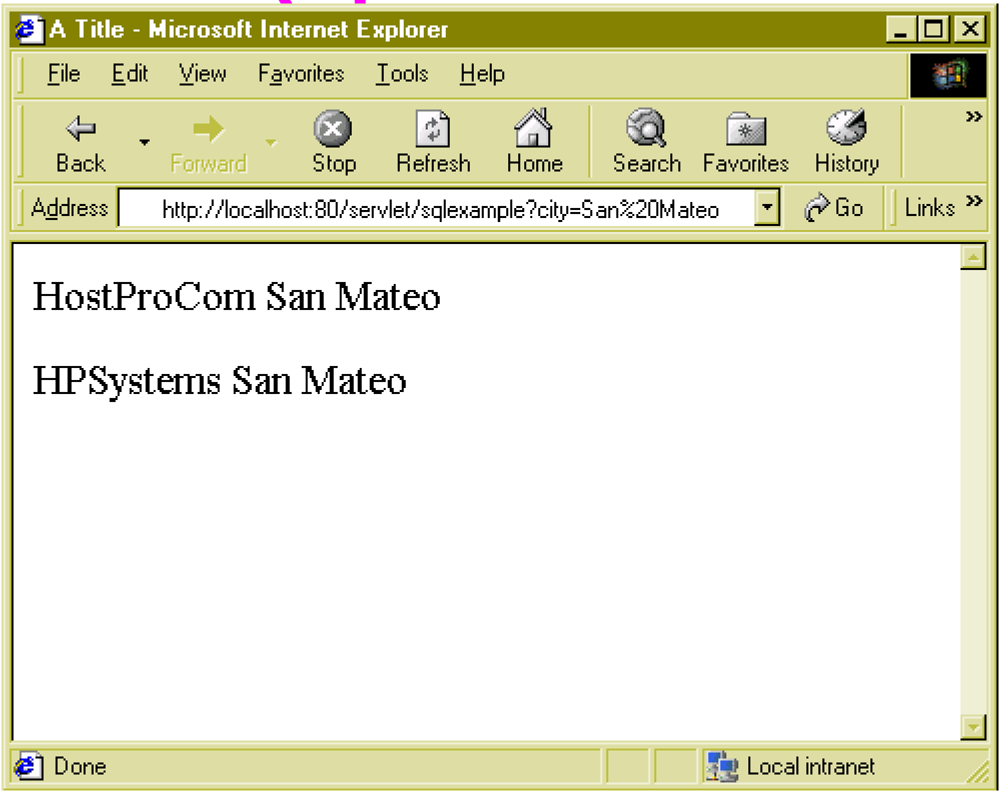
13 We could now build a form to allow users to enter a city name, but you learned how to do that in "Building a Simple Servlet" on page 5. Instead, let's take the shortcut—go back to your browser and add the parameter manually by appending to the URL:

```
?city=Dearborn
```

Your URL now reads:

```
http://localhost/servlet/sqlexample?city=San Mateo
```

14 Press Return to send off your request. The browser returns with:


A Title - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History
Address http://localhost:80/servlet/sqlexample?city=San%20Mateo
Go Links
Done
Local intranet

HostProCom San Mateo

HPSystems San Mateo

Connecting to a Database

Web server applications typically run in demanding concurrent multiuser environments. To make efficient and tunable use of system resources, the Spin database components use a **connection pool**: a group of connections that remain open for any transactions that need them, so that a transaction can use them without the overhead of opening and closing them explicitly. When your application runs, connections from the specified pool are created on demand according to the properties defined for them. If several database components need to connect to the same data source, they share these preconfigured connections.

For efficiency and convenience, your Spin application can connect using these pools, or, when circumstances dictate, it can use a customized SQL connection for explicit control over all properties of a dedicated connection. This section discusses using default connection-pooling. "Using a Dedicated Connection" on page 27 describes how to make and use a customized connection.

Defining Connection Pools

You'll need a connection pool for each database to which your application connects, and for each username and password combination that the application will use to identify itself. You define connection pools in a file named `spinsql.properties`. This is where the connections shown by the Connection Browser are defined. When you first start Spin, the Connection Browser shows only those connection pools defined in the default `spinql.properties` file. After you've defined your own connection pools in that file, you can access them also with the Connection Browser.

To define a connection pool:

1. With a text editor, open the file `spinsql.properties`, located in the Spin installation directory.
2. Specify the following properties for Spin to use when it creates connections for the pool. (You can override them in various ways when appropriate.)

`drivers`	A JDBC driver is necessary to connect to a data source. Specify all JDBC drivers used by any connection pool with their fully qualified classnames. Separate drivers with a semicolon, thus: `;`

Note: Spin ships with the JDBC-ODBC bridge driver and the default PointBase driver, both of which are specified in the default `spinsql.properties` file, listed in "spinsql.properties" on page 26.

`logfile`	The name of the file available for Spin to write its connection log entries: initialization status messages and connection errors.
`defaultConnectionName`	The name of the connection pool to use when no pool is specified. This connection is used the first time Spin invokes the Connection Browser.

3. For each connection pool, specify its properties. Properties take the form:

```
poolname.property=value.
```

poolname.url The URL used to designate the database to the JDBC driver. For the exact syntax required, consult the JDBC driver documentation.

poolname.maxconn
This property determines the number of active connections in the pool. If this property is omitted or set equal to –1, the number of connections is unlimited.

poolname.user The username used to connect to the database.

poolname.password
The password used to connect to the database.

Note: If username and password are to be supplied at runtime, you can omit these properties. However, it's necessary to set these during development in order to use the database component editors.

---

poolname.transactionIsolation
The level of concurrency control to use for any operation performed through the pool. As defined in the JDBC API, possible values are:

READ_COMMITTED	
	Dirty reads, nonrepeatable reads and phantom reads can occur.
READ_UNCOMMITTED	
	Dirty reads are prevented. Nonrepeatable reads and phantom reads can occur.
REPEATABLE_READ	
	Dirty reads and nonrepeatable reads are prevented. Phantom reads can occur.
SERIALIZABLE	
	Dirty reads, nonrepeatable reads and phantom reads are prevented.

The default transaction isolation is READ_COMMITTED. You can override transaction isolation at runtime by calling the `setTransactionIsolation` method. For a complete description of transaction isolation levels, consult the JDBC API documentation.

spinsql.properties

All Spin servlets running on the same Web server must share one `spinsql.properties` file. By default, `spinsql.properties` defines several connection pools for use with the like-named example servlet. Here is the file as shipped:

```
# point to drivers, logfile, default connection pool
 drivers=sun.jdbc.odbc.JdbcOdbcDriver
         com.pointbase.jdbc.jdbcUniversalDriver
 logfile=zatsql.log
 defaultConnectionName=sample

# for PointBase sample database
 sample.url=jdbc:pointbase://127.0.0.1/sample
 sample.maxconn=2
 sample.user=PUBLIC
 sample.password=public
```

```
# for bug database example
  buguser.url=jdbc:pointbase://127.0.0.1/zatexample
  buguser.maxconn=2
  buguser.user=buguser
  buguser.password=buggy
  buguser.transactionIsolation=READ_COMMITTED
# for polling example (webuser and polladmin)
  webuser.url=jdbc:pointbase://127.0.0.1/zatexample
  webuser.maxconn=10
  webuser.user=webuser
  webuser.password=spinup
  webuser.transactionIsolation=READ_COMMITTED

  polladmin.url=jdbc:pointbase://127.0.0.1/zatexample
  polladmin.maxconn=2
  polladmin.user=polladmin
  polladmin.password=super1
  polladmin.transactionIsolation=READ_COMMITTED
# for netcard example
  netcard.url=jdbc:pointbase://127.0.0.1/zatexample
  netcard.maxconn=2
  netcard.user=netcard
  netcard.password=netcard
  netcard.transactionIsolation=READ_COMMITTED
```

Using Connection Pools

To use Spin's default connection pooling, set the SQL statement property `connectionName` to the desired pool, as named in `spinsql.properties`. When the component is called upon to execute that statement, it transparently obtains a connection of the required kind. Details and examples are available in "Building SQL Statements" on page 3.

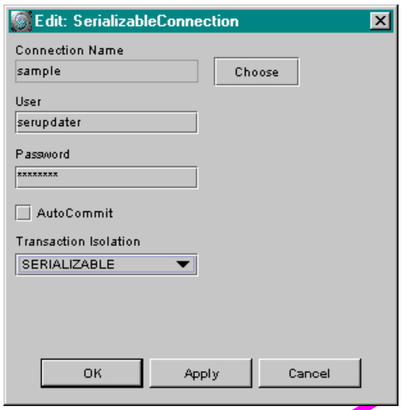
Using a Dedicated Connection

On occasion, it may be necessary to connect to the database using an explicitly defined, dedicated connection—for example, if the order of certain transactions matters, or if a certain operation is privileged and must be kept confidential.

To define the required connection explicitly with Spin's SqlConnection component:

1 Before it is needed, insert an SqlConnection component.
Insert > com.webgain.spin.sql > SqlConnection

2 Double-click on its icon to invoke the editor:


Edit: SerializableConnection

Connection Name: sample

User: serupdater

Password: ********

☐ AutoCommit

Transaction Isolation: SERIALIZABLE

3 Specify the connection. See "Specify a Connection" on page 7.

4 Specify the user name and password that the application will use to connect to the database.

5 If you wish the application to commit a transaction after every statement executes, check the `AutoCommit` option. If you leave it unchecked, your application must send the `commit` method to the dedicated SqlConnection when you wish to commit the transaction.

6 Specify the level of concurrency control you wish. Transaction isolation levels are briefly described in "Defining Connection Pools" on page 25. For a complete description, consult the JDBC API documentation. Click OK.

7 Before executing the SQL statement that must use the dedicated connection, create a Spin behavior to point the corresponding SQL component to the SqlConnection you've just configured:

`activateOn: parent.Behavior.activated`
`sendTo:` *your SQL statement*
`message: setSqlConnection` (inherited from SqlAbstractStatement)
`with:` *your SqlConnection*

8 Also before executing the sensitive SQL statement, create Spin behaviors that invoke the following methods on the SqlConnection:

❖ `connect()` to connect to the database,

❖ `commit()` or `rollback()` (if `AutoCommit` is not set) to commit successful changes or back out unsuccessful ones, and

❖ `disconnect()` to disconnect from the database when finished.

For More Information

1 Select any of Spin's database components (or indeed, any Spin component).

2 View > Get Object Info.

A browser opens on the JavaDoc documentation for the selected component, including its superclass, a brief description, and a description of every publicly accessible method it implements.

To play with examples, open the files in:

```
../Spin/exampleservlets/
```

Especially useful are the `sample`, `netcard`, and `bugdb` examples, and `tableExample.zac`.

CHAPTER 4

Using the Spin JSP Tag Library

JSP (Java Server Pages) tags provide a standardized means of incorporating dynamic elements into static HTML pages. JSP tag allows you to separate the model of the data (specified in the capsule) from the viewing of the data (specified in the HTML/JSP file). Spin's JSP tags let you create clean connections between capsules and HTML. For more on what you can accomplish with other, non-Spin JSP tags, consult a JSP book.

Most of the JSP tags implemented in Spin relate to user interface functions. These user interface JSP tags are described in detail in "JSP User Interface Tags" on page 4. Spin also provides a type of JSP tag called a control tag. JSP control tags are described in "JSP Control Tags" on page 1.

JSP Control Tags

Spin uses three JSP control tags, call, conditional, and repeat. JSP control tags control the HTML output sent from the servlet to the browser. To use the call, conditional, and repeat tags, you must first understand capsule methods. Methods are implemented in Spin by Action objects.

When a method is sent from a JSP tag, the capsule sees the method as an incoming event and relays it to any object listening for that event. A method invocation never directly returns a value; it's impossible. So any desired result must be accomplished using side effects. For example, the method might set a data item to a particular value, which in turn is used to set the value attribute of a tag. Sometimes methods "return" their value by sending the message

```
setTestResult
```

with a Boolean value. The result can then be queried during rendering.

Table 4-1 lists the JSP control flow tags implemented in Spin. All tag attributes for these tags may be specified at runtime.

Table 4-1: JSP control flow tags implemented in Spin

Tag name	Implemented by Spin class	Functionality
call	CallTag	Allows arbitrary HTML to be inserted into the document
conditional	ConditionalTag	Conditionally includes the body of the tag
repeat	RepeatTag	Repeats the body of the tag each time the tag's method evaluates to true

call tag

The simplest control flow tag is the call tag. Its attributes are listed in Table 4-2.

Table 4-2: call tag attributes

Attribute	Required	What it does
method	yes	The method name

To see an example of a call tag in use, open the following NetCard project:

```
spin/exampleservlets/NetCard/netcard.zap
```

Next, open the project's pickup.zac capsule. Double-click on the enclosed JSPTemplate icon and note the associated HTML file name. Open that HTML file with a text editor. Look for occurrences of `spin:call`. There should be four calls: one each for methods named `image`, `to_name`, `from_name`, and `message`. For each call tag encountered, an event is generated to the capsule and handled through any listeners.

In the pickup capsule, find the print on image action heading (under the cardimage heading). Double-click on the icon. Note that the action activates when the capsule.Capsule.image event is received and that the resulting action prints the image to the HTML file being generated. Look at other headings related to the call tags: print on fromName, print on toName, and print on message for similar usages. Close the capsules, project, and text editor windows.

conditional tag

Table 4-3 lists the attributes of the conditional tag:

Table 4-3: conditional tag attributes

attribute	required	what it does
method	yes	The method name

To see an example of how the conditional tag is used, open the following capsule:

```
spin/exampleservlets/JSPCFormTagTesting/JSPFormTagTesting.zap
```

This opens the JspFormTagTesting project. In the project editor view, double-click on the JSPConditionalTest.zac icon to display the capsule outline editor. In the capsule outline, double-click on the JSPTemplate icon and note the name of the corresponding JSP file.

Open the JSP file with a text editor. The file contains three conditional tags:

```
<spin:conditional method="verity">This text ought to show up.</
  spin:conditional>
<spin:conditional method="falso">This text should not show up.</
  spin:conditional>
<spin:conditional method="noexisto">This text should not show up.</
  spin:conditional>
```

Each tag invokes a different method. If the method evaluates to true, Spin processes the body of the tag; if the method returns false (or is not found) Spin does not process the tag body. Run the capsule to see the results.

Note that methods in Spin cannot directly return a value. Go to the capsule and double-click the verity heading. Rather than directly returning a value, the processing of the verity event causes the message `setTestResult` to be sent to the actor (the JSP template). The ConditionalTag bean instance then fetches the test result to determine whether to evaluate the body of the tag. Close the capsule and any open editors.

repeat tag

Table 4-4 lists the attributes of the repeat tag:

Table 4-4: repeat tag attributes

attribute	required	what it does
method	yes	The method name
limit	no	Specifies a maximum number of times the body of the repeat tag will be executed. If you do not specify a time, a default of 1000 iterations is enforced. Specify a limit or use –1 to prevent this default from being applied.

To see an example of how the repeat tag is used, open:

```
spin/exampleservlets/JspFormTagTesting/JSPRepeatTest.zac
```

This capsule makes use of four methods: yayOrNay, inductionVariable, squared, and cubed. When the yayOrNay event is received, the result is set to true — in order to continue execution no matter what — and the induction variable is bumped by one. While this might seem like a bug, recall that the number of times the repeat tag body is executed can be controlled by the limit attribute.

HTML creates five tables in the corresponding HTML file (specified in the JSPTemplate heading). The rows of the tables are filled in with the spin:repeat tags. To determine whether to print the row, the yayOrNay method is invoked. For the first column of the table, the inductionVariable method is invoked. For the second column, the squared method is invoked, and the third column is populated using the cubed method. In these examples, the number of times the repeat body is executed is controlled using the limit attribute. The example could have been written so that the yayOrNay method had a more complex expression to control the evaluation of the tag body. Close the capsule, and any text editors.

JSP User Interface Tags

Most UI tags have some attributes that are useful only with Javascript. These attributes are:

- onBlur
- onChange
- onClick
- onDblClick
- onFocus
- onKeyDown
- onKeyPress

- onKeyUp
- onMouseDown
- onMouseMove
- onMouseOut
- onMouseOver
- onMouseUp

Spin simply passes these attributes to Javascript, so any errors generated by incorrect use of these attributes are generated by Javascript and not by Spin.

Most UI tags also support style, styleClass, and tagIndex attributes. These attributes are similar to the identically named attributes in HTML. See an HTML reference for usage descriptions.

Table 4-5 lists the JSP user interface tags implemented in Spin. Because these tags mirror HTML tags of the same or similar name, you can find further details in an HTML reference document.

Table 4-5: JSP user interface tags implemented in Spin

Tag name	Implemented by Spin class	Equivalent HTML
button	ButtonTag	<input type="button">
cancel	CancelTag	<input type="submit"> with a default value of cancel/
checkbox	CheckBoxTag	<input type="checkbox">
file	FileTag	<input type="file">
form	FormTag	All the other tags must be contained by the spin:form tag to gain their extra functionality.
hidden	HiddenTag	<input type="hidden">
option	OptionTag	<option>
password	PasswordTag	<input type="password">
radio	RadioTag	<input type="radio">
reset	ResetTag	<input type="reset">
select	SelectTag	<select>
submit	SubmitTag	<input type="submit">
text	TextTag	<input type="text">
textarea	TextAreaTag	<textarea>

Although similarly named HTML tags give you the same functionality as user interface JSP tags, JSP provides some data storage capabilities that HTML does not. This, and other attributes of JSP are demonstrated in the following example.

1. Enable Spin's debug web server:

   Edit>Preferences>Debug Server tab

   Set WebServerEnabled to true
2. Create a new capsule and mark it as a servlet:

   File>New Capsule

   Capsule>Edit Properties
3. Select Type Java Servlet and close the dialog box.
4. Change the name of the capsule to frodo.
5. Within the capsule, insert a JSPTemplate:

   Insert>com.webgain.spin.html>JSPTemplate
6. Double-click on the JSPTemplate icon and type the following in the dialog box:

   `public_html/frodo.html`
7. Close the dialog, select the JSPTemplate icon, and insert a ServletGet behavior.

   Behavior>UserBehavior>ServletGet
8. Set the capsule to respond to both Get and Post HTML requests:

   Double-click ServletGet icon

   Click on + in Project Editor window

   Activate On>capsule.ServletRequest.doPost
9. Create a new text file called `frodo.html` within the directory `spin/public_html`. Paste the following into the new file:

```
<%@page language="java" buffer="4kb" isErrorPage="false" %>
 <%@taglib uri="/WEB-INF/lib/SpinTags.jar" prefix="spin" %>
 <HTML>
 <HEAD>
 <TITLE>
              Tag Example
 </TITLE>
 </HEAD>
 <BODY>
 <spin:form action="frodo" method="GET">
        <spin:checkbox name="check1"/>Check One<BR>
 <spin:checkbox name="check2"/>CheckTwo<BR>

 </spin:form>
 </BODY>
```

10. Save the file and run the capsule.

    File>Run (or Ctrl-R)

Your browser should now show two checkboxes. Click on the buttons and they change. Here's what the JSP lines of the file do:

The first two lines specify that the file contains JSP tags. (See the JSP documentation for more details.) The most important attribute is

```
    prefix="spin".
```

This tells the JSP compiler that

```
  <spin:something
```

is the start of a JSP tag.

Next, the

```
  <spin:form action="frodo" method="GET">
```

line specifies that a Spin form is to be created and that submitting the form from a browser will result in an HTTP Get request sent to the frodo servlet. This tag provides the hook to the ServletGet action you added to the JSPTemplate.

The

```
  <spin:checkbox name="check1" checked="false">Check One<BR>
        <spin:checkbox name="check2" checked="false">Check Two<BR>
```

specifies that two checkboxes are to be created. One is labeled “Check One” and the other “Check Two”. When the button is checked (and the submit button you will add below is pressed), the servlet will be passed parameters telling which checkboxes are checked by using the name attributes. The

```
        <BR>
```

tag is a standard HTML tag to break the line so the boxes are on separate lines.

Let’s make the example more interesting. Open the HTML file again. After the two checkbox lines, add the following new lines:

```
        <spin:radio name="radio" value="red" checked="true"/>Red
        <spin:radio name="radio" value="blue" checked="false"/>Blue
        <spin:radio name="radio" value="puce" check="false"/>Puce
```

Save the file and refresh your browser. Three radio buttons have appeared. Since the tags specify the name attribute of each is “radio” they are grouped together and only one button in the group may be pushed. If a submit button were present and pushed, the selected button would be sent to the servlet.

Note that all JSP tags require their attribute values (the text to the right of the = sign) to be enclosed between quotes. Probably the most common error in using JSP tags is forgetting the quotes. All attribute names are case-sensitive.

Note also that every attribute specified in a JSP tag is of the form

```
        name="value"
```

even if the corresponding HTML tag doesn’t take an argument. This is simply standard JSP syntax. Now add a submit button.

In your HTML file, after the last radio tag, add the following:

```
  <spin:submit name="submit" value="Make it so!"/>
```

Save the file and refresh your browser. Click the checkboxes and radio buttons, then push the button. See your data disappear! The trick is, that although the servlet saw the updated values for the buttons, it didn't put them into the new HTML when the browser refreshed (after the submit button was pressed). Go to the ServletGet multi-action in the capsule and insert a new child behavior.

```
Insert->Behavior->Action.
```

Double-click the new behavior's icon and tell it to:

```
Activate if parent.Behavior.activated
If= capsule.getParameter("submit") != null
Send to: actor
Message: setFormFillInParameters
Eval=: capsule.getAllParameters().
```

Close the editor window, then use the up-arrow button to position the new behavior before the SendDocument behavior.

We'll explain what all that means in a moment. First, you need to know that when the submit button is pressed, the values for the widgets on the form are sent to the servlet, and Spin stores them in a hash table. That table is accessed by sending the capsule the message getAllParameters(). So, the result of the above actions is that when the parent behavior (the ServletGet) is activated, the actor (the JSPTemplate) is asked to fill in its parameters from those held by the capsule.

Refresh the browser and push some buttons. Now hit the submit button. This time, the settings are remembered. This "remembering" is one of the advantages Spin's JSP tags have over standard HTML forms.

Attributes for Individual User Interface Tags

The tables in this section detail the attributes you can specify for each Spin user interface tag. Be sure to enclose all attribute values in quotation marks. All tag attributes for these tags may be specified at runtime.

button tag attributes

attribute	required	what it does
name	no	The identifier for the action button
value	no	The label which will appear on the button

Table 4-6: cancel tag attributes

attribute	required	what it does
name	no	The identifier for the cancel button
value	no	The label which will appear on the button. Also, the value which will be sent to the servlet if the button is clicked. Note: This attribute defaults to "Cancel" if it is not specified.

Table 4-7: checkbox tag attributes

attribute	required	what it does
name	yes	Specifies a name for a checkbox
value	no	Defines the value which will be sent to the servlet when the box is checked.
checked	yes	Specifies whether the checkbox is initially checked

Table 4-8: file tag attributes

attribute	required	what it does
accept	no	
maxlength	no	
name	yes	Identifies to the servlet which file tag is providing input
size	no	Width, in characters, of the typein field.
value	no	Initial value sent to browser

Table 4-9: form tag attributes

attribute	required	what it does
action	no	Specifies which capsule runs on the server when the form is submitted
enctype	no	Specifies encoding type. See an HTML reference.
focus	no	For Java scripting
method	no	Specifies how the form data is sent to the web server when it is submitted. GET and POST are two acceptable values.
target	no	For Java scripting

Table 4-10: hidden tag attributes

attribute	required	what it does
name	yes	Specifies the name of the field to the servlet
value	no	The information you want stored

Table 4-11: option tag attributes

attribute	required	what it does
initialvalue	no	Used with select tag, specifies the value sent to the servlet when the menu item is selected.

Table 4-12: password tag attributes

attribute	required	what it does
maxLength	no	Specifies maximum length of the input
name	yes	Identifies password box to the servlet
size	no	Specifies size of password box (in number of characters)
value	no	Default value

Table 4-13: radio tag attributes

attribute	required	what it does
name	yes	All same-named radio buttons are part of the same group (max of one pushed within the group)
value	yes	The value which is sent to the web server if this radio control is checked when the form is submitted.
checked	yes	Whether button is initially checked when the form is rendered

Table 4-14: reset tag attributes

attribute	required	what it does
name	no	Ignored since reset is handled by the browser (and not the servlet!)
value	no	The button's label

Table 4-15: select tag attributes

attribute	required	what it does
multiple	yes	Specify that the menu supports multiple selections (**Note:** Spin currently remembers only one selection – even if the menu supports multiple).
name	yes	Identifies the menu to the servlet
size	no	Height, in lines, of the menu.
initialvalue	no	Specifies a default selection in the menu.

Table 4-16: submit tag attributes

attribute	required	what it does
name	no	The identifier for the submit button
value	no	The label in the button. Also, the value which is sent to the servlet if the button is clicked.

Table 4-17: text tag attributes

attribute	required	what it does
maxlength	no	Maximum number of characters allowed to be typed in text box
name	yes	Identifies text box to server.
size	no	Size in characters of the text box.
value	no	Initial text to display.

Table 4-18: textarea tag attributes

attribute	required	what it does
cols	no	Width of the text area in characters.
name	yes	Identifies text box to server
rows		Height of the text area in characters
value	no	Initial text to display.

Creating a JSP with Embedded Spin Capsule

This section introduces the capsule of type JSP Bean, and provides a larger context for working with JSPs in Spin. Earlier sections in this chapter explained individual JSP tags. The following example show you how to incorporate JSPs into a Spin project.

1 Start Spin.

2 Create a new capsule

File -> New Capsule

3 Change the Capsule type to JSP Bean

   a Capsule ->Edit Properties (or right-click the capsule bean icon).

   b Select the JSP Bean radio button.

   c Click the Close button.

4 Add content to the capsule

Now add whatever content is appropriate to your project to the capsule. For this example, which constructs a page to display the time, do the following:

   a Click the capsule bean to select it.
   Insert -> com.webgain.spin.misc -> Clock.

   b Select the capsule bean again.
   Insert -> com.webgain.spin.html -> HTMLText.

   c Select the HTMLText bean.
   Insert -> Behavior -> Action

   d Double-click on the action behavior you just created to edit it. Set the fields in the dialog box as follows:

   Activate On: capsule.Capsule.newMethod… When prompted, name the method getTime.

   Send To: actor

   Message: print

   Eval: capsule.getWriter()

   e Select the HTMLText bean.
   Insert -> Behavior -> Action

   f Double-click on the action behavior you just created to edit it. Set the fields in the dialog box as follows:

   Activate On: actor.WhenOutput.whenOutput

   Send To: actor

   Message: setText

   Eval: Clock.getCurrentTime

The capsule you just created will get and output the current time when it receives a “getTime” call.

5 Save the Capsule

- **a** Stay in the capsule view. Select File -> Save.
- **b** Navigate to a folder of your choice. Save the file as `clock.jsp`.

6 Add the Capsule to a Project

- **a** Activate the Project window.
- **b** Project -> Add
- **c** Select the capsule you just saved (clock.jsp). The file clock.jsp appears in the project window.
- **d** File -> Save
- **e** Save the project to the folder where you saved the capsule, naming the project `clock.zap`.

7 Edit the JSP

- **a** Open `clock.jsp` in your favorite text or HTML editor. You see that Spin has stored the contents of your capsule at the top of the file, between the lines marked

```
<%!//SPIN GENERATED:DO NOT MODIFY
```

and

```
//END SPIN GENERATED:DO NOT MODIFY%>.
```

- **b** Place your HTML and JSP content in the Head and Body sections at the bottom of the file. For this example, add the following line below the `HTML <BODY>` tag:

```
The current time is: <spin:call method="getTime"/>
```

This line of text includes a Spin tag for communicating with the Spin capsule. See documentation earlier in this chapter for information concerning how this works.

- **c** Save and close the file.

8 Load the JSP in your browser With Spin still running and the Clock project window still open, start your browser and go to the following address:

```
http://localhost/clock.jsp
```

If all works as expected, the browser should display something like the following:

```
The current time is: Wed Oct 31 14:56:27 PDT 2000
```

C H A P T E R 5

Working with EJBs in Spin

Configuring Your Environment

Before using an EJB in a Spin capsule for the first time, you must configure both your EJB server and Spin to be aware of that EJB. This section explains the configuration process, using the SimpleBank EJB that ships with Spin as a reference example.

This example documents configuration and use of the jBoss EJB server that ships with Spin. If you are using another EJB server, please see your server documentation for configuration procedures.

Conventions

In these examples, *FileName* denotes the install path to a program or file named FileName. So *Spin* denotes the directory where you installed Spin, (for example c:\Program Files\Spin) and *jBoss* denotes the install directory for jBoss, which is *Spin*\jBoss.

Configuring the Server

This section explains the steps necessary to deploy EJBs to the jBoss EJB server. For each step general information is given first, followed by the specific changes required for the Bank Transactions project.

Copy the EJBs to the jBoss\deploy folder

You must place a copy of the EJB .jar files in the *jBoss*\deploy folder in order for them to be deployed. For the BankTransactions project, copy the file SimpleBank.jar to *jBoss*\deploy\SimpleBank.jar from *Spin*\exampleservlets\Bank\EJB.

Copy other files

If your beans require auxiliary files (e.g. a database) then copy those files as well. The BankTransactions project requires the ejbdata database. This database has been distributed with Spin in the *Spin*\pointbase\databases folder, and for this configuration should be left where it is.

Edit the jBoss.properties file

The jBoss.properties file contains, among other things, information regarding java database drivers. Open the file in a text editor and check if your database driver is present. If your driver is not present then you should add it to the list. The BankTransactions project uses the Pointbase net driver, `com.pointbase.net.netJDBCDriver`. If you open the *jBoss*\conf\jBoss.properties files, you will see that Spin ships with that driver added to the driver list.

Add Database Drivers

In order for jBoss to use your database driver, it must be able to find the driver's .jar file. An easy way you can facilitate this is to place the driver .jar files in the `jBoss\lib\ext` folder.

For the BankTransactions project, the driver .jar file is pbclient31RE.jar. This file ships in the *Spin*\pointbase\classes folder, where it is referenced from the jBoss configuration files, so you do not need to move it.

Edit the jBoss.conf file

The jBoss.conf file contains a variety of configuration information. If your EJB requires a database connection, this is where you set up the connection pool information. For the BankTransactions project, you will see the following lines:

```
<MLET CODE = "org.jboss.jdbc.DataSourceImpl" ARCHIVE="jboss.jar,../../
  ../pointbase/classes/pbclient31RE.jar" CODEBASE="../lib/ext/">
    <ARG TYPE="java.lang.String" VALUE="jdbc:pointbase://127.0.0.1/
  ejbdata">
    <ARG TYPE="java.lang.String" VALUE="BankDataSource">
    <ARG TYPE="java.lang.String" VALUE="com.pointbase.net.netJDBCDriver">
    <ARG TYPE="java.lang.String" VALUE="public">
    <ARG TYPE="java.lang.String" VALUE="public">
</MLET>
```

Configuring Spin

This section explains the steps required to make your EJBs accessible to Spin.

Process EJBs for Spin

Start Spin.

When Spin finishes loading, you see a Project window. Depending on your settings, you may also see a Console window. Click in the Project window, then select File -> Create Spin EJB… In the Open dialog box that appears, navigate to the first of your EJB .jar files you wish to process, select it, and hit Open. Spin copies the bean to the Spin\beans\ejb folder and processes the EJB into a format that it can use. Information regarding the status of the process is output to the Console window. If the processing is successful, you see a message that the bean(s) was found and successfully processed.

Repeat these steps for each bean you wish to process.

For the BankTransactions project, you need to process the SimpleBank.jar file, which can be found at `Spin\exampleservlets\Bank\EJB\SimpleBank.jar`.

Enable the Web Server in Spin

To test servlets in Spin, you must enable the web server. Click in the Project window, then select Edit -> Preferences… In the Preferences Dialog Box, select the DebugServer tab. Change the WebServerEnabled radio button at the bottom of the dialog box to "true". Close the dialog box. The Web Server will now be enabled the next time you start Spin.

Enable jBoss in Spin

To have jBoss started automatically when Spin starts up, you must enable it in Spin. Click on the Project window, then select Edit -> Preferences…. In the Preferences Dialog Box, select the EnterpriseJavaBeans tab. Change the JBossEJBServerLoadsAtStart radio button in the middle of the dialog box to true. Close the dialog box.

Enable Pointbase in Spin

In order for the EJB to talk to the Poinbase database, you must start the Pointbase server within Spin. Edit->Preferences one more time. Select the General tab. Set the PointbaseServerLoadsAtStart radio button to true. Close the dialog box.

Restart Spin

In order to start the jBoss Server, you must restart Spin. Quit the Spin application and then start it up again.

As Spin starts up, it spawns a new process which loads jBoss. You can check the status of the jBoss Server by choosing the jBoss tab in the console window. Among other things, you will see information regarding the EJBs you have deployed and the database connections you have set up.

Running the BankTransactions Example

Now that you have configured your environment, you may run or create projects that use the EJBs you specified in the previous sections. Follow the steps in this section to play with the Bank Transactions example.

1. Open the Project File (This step assumes that you have an open Project view resulting from your re-starting Spin in the last section.)

   To run the BankTransactions example, begin by opening the project file in Spin. Click on the Project window, then select File -> Open. Navigate to the folder `Spin\exampleServlets\Bank\BankTransactions`, and open the file `BankTransactions.zap`.

2. Open the Capsule

   The Project window now shows the contents of the BankTransactions project. One of the items is a Spin capsule, called BankTransactions.zac. Open this file by double clicking on it. In a separate Capsule Outline window, you see the outline format for the BankTransactions capsule. Near the top of the outline, you see the SimpleBank EJB that is used by this example.

3 Run the Example

Click on the capsule window, then select File -> Run. This causes a variety of things to happen, including the launching of your default browser. If all goes well, you see a logon screen for our simple bank project. Note that output from our session is sent to the Console window. It should now tell you that the server has been started, among other things.

Play around with the BankTransactions example. To get past the logon screen, enter a valid account and password. One valid account is "111111" with password of "whatever".

Learning from the Bank Transactions Example

The BankTransactions project uses the SimpleBank EJB as a way to talk to the database to validate logons and handle deposit and withdrawal transactions. Check out the way the SimpleBank object is used within the Spin capsule. Some things that may be of interest are:

- The bean itself had to be included and configured. Double-click the SimpleBank bean instance to see which properties have been set.
- The "findAccount" action calls the findByPrimaryKey() method on the bean, using account information that is sent to the capsule during the servlet post request.
- The "testPassword" conditional expression uses the EJB to validate the password entered by the user.
- The "storeAccountName" action requests the name associated with the account from the bean, which in turn retrieves it from the database.
- The "ifSufficientFunds" conditional expression asks the bean if there are sufficient funds for the requested withdrawal before processing the user request
- The "deposit" action tells the bean how much money to deposit to the account

APPENDIX A

Events

This appendix lists the events generated by Spin's built-in behaviors, capsules, and other components. In addition to the events listed here, many beans generate their own specific events. For details of the properties, methods, and events of a particular bean, choose View>Get Object Info. Similarly, for details on Java classes, see their JavaDoc documentation.

Events Generated by Behaviors

All behaviors generate:

```
Exception.activate
Exception.activateThenRethrow
```

IfTest

IfTest Behaviors generate:

```
Test.ifTrue
Test.ifFalse
```

ActionGroup

ActionGroup Behaviors generate:

```
Behavior.activated
```

Timeline and Counter

Timeline and Counter Behaviors generate:

```
When.started
When.stopped
Counter.count
```

Events Generated by Capsules

All capsules generate:

```
WhenCapsule.started
WhenCapsule.cloned
WhenCapsule.isNewClone
WhenCapsule.propertySet
```

And, for each method defined on a capsule:

```
Capsule.methodName
```

Applet Capsules

Java class `com.webgain.spin.CapsuleAppletClass`.

Run-time instances inherit from `java.applet.Applet`, and generate the events that class generates.

Application Capsules

Java class `com.webgain.spin.CapsuleApplicationClass`.

Run-time instances inherit from `java.awt.Frame`, and generate the events that class generates.

Nonvisual Actor Capsules

Java class `com.webgain.spin.CapsuleStaticClass`.

Visual Actor Capsules

Java class `com.webgain.spin.CapsuleComponentClass`.

Run-time instances inherit from `java.awt.Container`, and generate the events that class generates.

Servlet Capsules

Java class `com.webgain.spin.CapsuleServletClass`.

Servlets generate:

```
HttpService.httpService
ServletRequest.doDelete
ServletRequest.doDestroy
ServletRequest.doGet
ServletRequest.doInit
ServletRequest.doOptions
ServletRequest.doPost
ServletRequest.doPut
ServletRequest.doTrace
ServletRequest.getLastModified
```

Java Server Page Bean Capsules

Java class `com.webgain.spin.CapsuleJSPBean`.

JSP beans generate:

```
JSPRequest.doInit
```

Events Generated by Other Built-in Components

When immediate subcomponents move to hit the boundary of their scene, or move to hit one another within their scene, the built-in bean Scene generates:

```
Boundary.hit
Collision.hit
```

CHAPTER B

User Interface Reference

This appendix describes the following Spin editor windows and other user interface features:

- "Viewing JavaDocs" on page B-1
- "Project Editor" on page B-1
- "Capsule Outline Editor" on page B-5
- "Toybox Editor" on page B-13
- "Debugging Using an Error Window" on page B-14
- "Debugging Using the Console Window" on page B-15

Viewing JavaDocs

You can display JavaDoc documentation using any one of the following methods:

- View > Get Object Info
- Actor > Get Object Info
- Control-I (Command-I on the Macintosh)
- Right-click (Command-click on the Macintosh).

Project Editor

You use the project editor to manage your project at the top level. You must use the project editor if your project contains more than one capsule or if you want to deploy your project as a web archive (WAR) file. (To perform tasks on a capsule and its components, use the capsule outline editor; see "Capsule Outline Editor" on page 5.) Figure B-1 contains an example of the project editor.

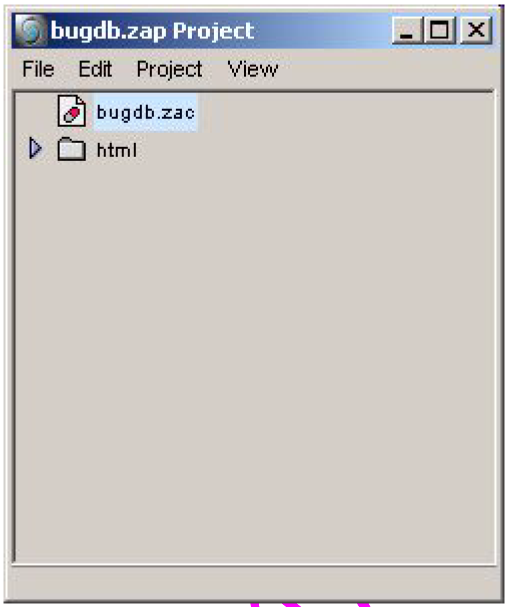


Figure B-1: Project Editor Window

File Menu

The File menu in the Project Editor window contains the following menu options:

New Capsule	Creates a new capsule and opens a new capsule outline editor window.
New Project	Creates a new project and opens a new project editor window.
Open	Opens an existing project or capsule.
Close	Closes the current window.
Save	Saves the current project.
Save As...	Prompts for a new file name under which to save the current project.
Save As Jar	This function is enabled only in the Capsule Editor.
Save Behavior	This function is enabled only in the Capsule Editor.
Run	This function is enabled only in the Capsule Editor.
Print...	This function is enabled only in the Capsule Editor.
SQL Connections	Opens a dialog that allows you to manage SQL connections. Enabled only when the Spin SQL components are loaded.
Create Spin EJB...	Prompts for an EJB to import into Spin, then adapts that EJB to make it usable in a Spin capsule. Enabled only if the application server classes and javax.ejb classes from your application server are in your classpath. (This menu item appears only when classpath includes ejbsuptjar, which it does on a normal installation.)

Quit	Exits Spin, prompting you to save any unsaved windows.

Edit Menu

Cut	This function is enabled only in the Capsule Editor.
Copy	This function is enabled only in the Capsule Editor.
Paste	This function is enabled only in the Capsule Editor.
Delete	Removes the currently selected files from the project.
Preferences...	Opens the Preferences dialog box so that you can modify the following fields:

General

JDKDocsDirectory	Specifies the directory (in addition to the Spin `docs` directory) in which the Get Info command looks for documentation.
DebugPortStartOf32	Specifies the first port in a range of 32 ports used by Spin's debugger. The default is 9544.
ExternalCommandPort	Specifies a port Spin can use to send itself messages. Used when Spin is invoked externally after one instance of Spin is already running. The default is 9543.
PointBaseServerLoadsAtStart	Specifies whether Spin should load the point base server at startup. The default is false.

Display

EditorFontSize	Specifies the font size used in editor windows.
DialogFontSize	Specifies the font size used in regular dialogs.

ScriptOptions

WithMenuSearchDepth	Specifies the depth of menus that Spin will prebuild.
Imports	Lets you specify the names of Java packages so you do not have to specify full package class names.

EnterpriseJavaBeans

DefaultContextFactory	Specifies the name of the EJB context factory on the EJB server.
DefaultEJBServerURL	Specifies the URL for your default EJB server. The default is t3://localhost:7001.
DefaultHomePrefix	Specifies the prefix JNDI name used to look up the home interface to create an EJB instance.

JBossEJBServerLoadsAtStart	Specifies whether Spin loads the JBoss EJB server at startup. The default is false.
DefaultUrlPkgPrefixes	To be supplied.
DefaultPropertiesFile	Lets you specify a filename from which all EJB properties except Home will be read. Properties specified in this file override properties specified in the EJB menu.

DebugServer

URLOrDirectoryForNonServletRequests	Specifies the directory (relative to Spin's installation directory) used to satisfy requests to the web server that do not invoke a servlet.
ServerPort	Specifies the port used by Spin's built-in web server, which is used to test servlets.
NumberOfPostParameters	Specifies the number of parameters that can be passed to a servlet with a Post request.
ServletAccessDirectory	Specifies the directory in which Spin looks for any servlet that is not part of a project.
WebServerEnabled	Specifies whether Spin's built-in web server is enabled. Default is false.

Project Menu

Add...	Imports a file into the project editor.
Remove...	Removes any currently selected files from the project.
Project Inspector	Opens a window from which you may edit the properties of a currently selected Spin capsule or JSP on the Project file list.
Show Full Path/Hide Full Path	Toggles between showing only the file name or the entire path of files listed in the Project file list. The default is to display the file name only.
Deploy Project as WAR	Packages all the project files into a WAR file. Used for deploying servlets and web applications.

View Menu

Console	Toggles on and off Spin's console window.
Spin Documentation	Displays the Spin Documentation page in your default browser.
Spin Resources	Display's Spin's home page.
About Spin	Shows the About Spin dialog. Click anywhere on the window to close it.

Capsule Outline Editor

You use the capsule outline editor to insert, edit, and manage components within a capsule. Figure B-2 contains an example of the capsule outline editor.

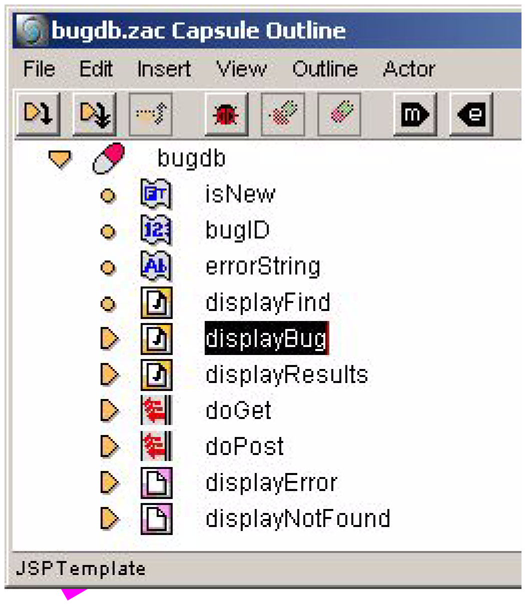


Figure B-2: Capsule Outline Editor Window

Drag and Drop in the Outline

You can drag and drop objects in the capsule outline to reorganize them. When you drag an object's icon onto another object, that object (along with all its children) becomes a child of the other object.

You can also create a copy of what you are dragging, leaving the original intact. To do this, hold down the Control key (Windows) or the Option key (Macintosh) while you drag.

Using the Toolbar

You can use the toolbar buttons in the capsule outline to perform most of the functions available from the Outline menu. Figure B-1 shows the capsule outline toolbar.

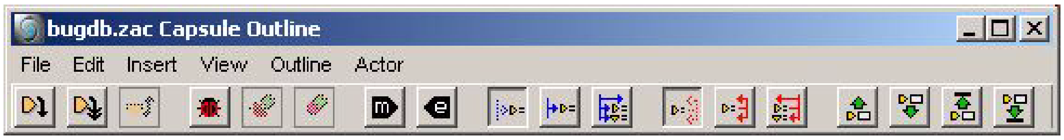


Figure B-1: Capsule Outline editor toolbar

Expands selected item.

Expands all items in outline.

Collapses selected item.

Enables or disables debugging. For details, see the description of Outline>Debug>Enable Debugging on page B-11.

Adds or removes watch variable. For details, see the description of Outline>Debug>Add Watch Variable on page B-12.

Adds or removes breakpoint. For details, see the description of Outline>Debug>Add Breakpoint on page B-11.

Shows methods. For details, see the description of Outline>Show Methods on page B-11.

Shows events. For details, see the description of Outline>Show Events on page B-11.

Disables the arrows that show the flow of activation events.

Shows the flow of activation events to and from the selected item. For details, see the description of Outline>Show Activation Events on page B-11.

Shows the flow of all activation events.

Disables the arrows that show the flow of events sent to action targets.

Shows the flow of events sent to action targets by the selected item. For details, see the description of Outline>Show Action Targets on page B-11.

Shows all action targets. For details, see the description of Outline>Show Action Targets on page B-11.

Moves selected item up.

Moves selected item down.

Moves selected item to top.

Moves selected item to bottom.

File Menu

New Capsule	Creates a new capsule and opens the capsule editor on it.
New Project	Creates a new project and opens the project editor on it.
Open	Opens an existing capsule.
Close	Closes the current window. If the current window is the last window, also closes Spin.
Save	Saves to the currently defined file name.
Save As	Prompts for a new file name under which to save the current capsule.

Save As Jar	Writes out a finished capsule into a .jar file (Java archive file) in a format specific one of the following types of capsule: ■ applet ■ application ■ visual actor (component) ■ nonvisual actor ■ servlet
Save Behavior	Saves the selected behavior. The saved behavior is usually a group behavior that is the root of a tree containing other behaviors and data components. This behavior is saved as a JavaBean into the `behaviors` subdirectory of the Spin installation directory.
Run	Precompiles all Java expressions and runs the current capsule as if you were running it as a stand-alone application or as an applet in a browser.
Print	Prints the contents of the current outline window.
SQL Connections	Opens a dialog from which you can change parameters related to your SQL connections. Enabled only when the Spin SQL components are loaded.
Create Spin EJB...	Allows you to specify a `.jar` file. Spin then copies the specified bean to `Spin/beans/ejb`, and processes the EJB into a format that Spin can use.
Quit	Exits Spin, prompting you to save any unsaved windows.

Edit Menu

The Edit menu contains the Cut, Copy, Paste, Delete, and Preferences commands.You can use Cut, Copy, Paste, or Delete on any item or selected text in the capsule outline. You can copy and paste text and components between Spin windows, but not to and from other applications. The Preferences command displays the Spin Preferences dialog box. You can set Spin preferences using any editor window's Edit>Preferences command; for more details, see the Preferences dialog box description on page B-3.

Insert Menu

Capsule	Inserts a capsule into the outline.
Scene	Inserts a scene (a visual component used to group other visual components). Note that a capsule is associated with an implicit scene by default, but you can add additional scenes to a capsule.
Behavior	Inserts one of the following built-in Spin behaviors: Action: An event that invokes a method or script. Script: Arbitrary Java code.

	IfTest: A behavior that generates specific events when an expression evaluates to true or false.
	ActionGroup: A set of actions grouped into a single unit.
	Counter: An object that maintains a running count when activated.
	Timeline: An object that allows the modification of actor properties over a span of time.
User Behavior	Inserts one of the behaviors saved using the File>Save Behavior command. This menu item may include any of the following:
	ServletGet: An action that activates when capsule.ServletRequest.doGet is invoked.
	Drag: A Spin behavior that allows a visual component to be dragged.
	Spiral: A Spin behavior that causes a visual component to spiral when clicked.
	Jump: A Spin behavior that causes a visual component to jump when clicked.
Data Item	Inserts one of the built-in data types that can be used in a capsule:
	int: An integer value.
	boolean: A true/false value.
	float: A 32-bit floating point value.
	double: A 64-bit floating point value.
	string: A character string (java.lang.String).
	color: A color (java.awt.Color).
	Hashtable: A name/value object pair (java.util.Hashtable).
	Vector: A growable array of objects (java.util.Vector).
	Point: An X/Y position in two-dimensional space (java.awt.Point).
	Rectangle: A rectangle definition (java.awt.Rectangle).
	Dimension: A height and width value (java.awt.Dimension).
	URL: A "Uniform Resource Locator" (java.net.URL).
	HttpCookie: Private information on the client.
AWT Components	Contains Sun's Abstract Windowing Toolkit widgets, which can be used as actors:
	Button: An object that can be labeled and that activates events when clicked.
	Checkbox: A box, with associated text, that displays a binary selection.
	Choice: An object often used for displaying drop-down menus.
	Label: Displays specific text at a specific location.
	List: A list of strings.
	Scrollbar: A visual scrollbar.

TextArea: An area in which multiline text can be displayed and edited.

TextField: A line of text that you can set to be edited by the user.

Frame: An application window that you can set to be resized by the user.

The bottom part of the Insert menu contains an entry for each package Spin finds in the files located in the `beans` subdirectory on startup. These are the actors you can use in capsules. If you have installed the Standard Beans, this section initially contains five entries:

- com.webgain.spin.demo, the Juggler visual actor
- com.webgain.spin.display, various sample visual actors
- com.webgain.spin.html, various actors useful for creating HTML documents
- com.webgain.spin.misc, various sample nonvisual actors
- com.webgain.spin.sql, various actors useful for connecting to a database and manipulating its content

View Menu

Open Toybox	Opens a toybox editor for this capsule.
Open Layout	Opens a layout editor either for a selected scene or for the implicit scene associated with a capsule.
Palettes	Opens separate palette windows that you can use instead of the Insert menu to insert actors, behaviors, and data items into a capsule.
Reveal Palettes	Brings the palette windows to the front if they are hidden by other windows.
Edit Selected Object	Used to edit an actor, behavior, capsule, or data item. Opens the editor for that entity. Double-clicking on an entity's icon is usually a shortcut for this command.
Get Object Info	Opens an Info window for the selected actor. The Info window allows you to view Variables, Properties, Methods, Events, and JavaDoc (when available) for that actor.
Source	Opens a window that shows the source generated when you compile your capsule.
Console	Toggles on and off Spin's console window.
Spin Documentation	Displays the Spin Documentation page in your default browser.
Spin Resources	Display's Spin's home page.
About Spin	Shows the About Spin dialog. Click anywhere on the window to close it.

Outline Menu

Collapse	Collapses the selected outline item if it has children.

Expand	Expands the selected outline item if it has children.
Expand All	Expands all outline items in the current view.
Move	Moves the selected outline item up or down among its siblings.
Show Activation Events	Displays arrows that show you the flow of *activation events* (events that activate behaviors). You can choose from None, To/From Selection, and All. If you choose To/From Selection, Spin displays only the activation events that activate the selected item or are activated by the selected item.
Show Action Targets Submenu	Displays arrows that show you which actions send events to which *action targets*. Action targets are actors to which events are sent. You can choose from None, To/From Selection, and All. If you choose To/From Selection, the arrows point to only those action targets to which the selected item sends events.
Show Methods	Displays arrow bars that point to items activated by methods. Black arrow bars denote defined capsule methods; red arrow bars denote undefined capsule methods. The arrow bar is displayed as a solid color when it points to the specific item that uses the method; when pointing to a collapsed item that contains a child that uses the method, the arrow bar is displayed in outline. Ellipses (...) show that the item can be activated by more than one method.
Show Events	Displays arrow bars that point to items that send an event. Black arrow bars denote defined capsule events; red arrow bars denote undefined capsule events. The arrow bar is a solid color when pointing to the specific item that sends the event; when pointing to a collapsed item that contains a child that sends the event, the arrow bar is displayed in outline. Ellipses (...) show that the item sends more than one event.
Show Properties	Displays an icon next to items with capsule property bindings. The icon is a solid color when pointing to the specific item that has capsule property bindings; when pointing to a collapsed item that contains a child with capsule property bindings, the arrow bar is displayed in outline.
Debug Submenu	Servlets are not visual and benefit from additional capabilities to debug them. Except for the Enable Debugging option, the following menu items are enabled only when using servlets. Enable Debugging: Enables debugging and opens the debugger window for the servlet displayed in the outline. If debugging is already turned on, this menu item disables debugging. Add Breakpoint: Adds the selected behavior to the list of breakpoints. Breakpoints are displayed in the outline by a circle to the left of the associated behavior. Remove Breakpoint: Removes the selected behavior from the list of breakpoints. Remove All Breakpoints: Clears all active breakpoints.

Add Watch Variable: Adds the selected data item to the list of variables whose value can be watched, and displays it in the debugger window. Each step in the execution of a servlet refreshes all the watch variables in the debugger window.

Remove Watch Variable: Removes the selected data item from the list of watch variables.

Remove All Watch Variables: Clears all active watch variables.

Capsule, Behavior, or Actor Menu

The title of the Capsule, Actor, or Behavior menu changes according to the currently selected object. You use this menu to edit attributes of the selected object. The menu contents are identical to those of the pop-up menu that appears when you Right-click (Command-click on a Macintosh) on the selected object.

Capsule Menu

Edit Capsule	Opens an outline editor for the selected capsule.
Get Object Info	Opens the Object Info window, displaying the JavaDoc for the Capsule.
Edit Properties	Opens the properties editor for the selected capsule:

Behavior Menu

Edit Behavior	Opens the behavior in its own window for editing. (If the behavior is already open, this brings its window to the front.)
Delete	Deletes the behavior from the capsule.

Actor Menu

The Actor menu is the same as the pop-up menu available on an actor. It contains:

Edit-Custom	Opens the custom editor provided by the JavaBean for editing the selected actor. Enabled only when a custom editor is provided. This is the same editor you get when double-clicking on the actor, unless no custom editor exists, in which case a generic editor opens instead.
Edit-Generic	Opens the generic editor, which allows you to edit all the properties of the selected actor.
Edit Name	Allows you to edit the name of the selected actor.
Get Object Info	Opens the Object Info window. This window provides information on the Actor's variables, properties, methods, and events, and displays JavaDoc documentation.
Delete	Deletes the actor from the capsule.
Edit Property	Displays a submenu of all the properties available to the selected actor. When you select a property, Spin displays a dialog box that allows you to edit that property.

Toybox Editor

The toybox editor is a powerful editor that displays visual components using a graphical interface. The toybox editor lets you manipulate visual components in the capsule while you are constructing it. The toybox editor displays only visual actors. If an actor does not have a visual component, you must edit it in the outline editor. You use the capsule outline editor to insert, edit, and manage components within a capsule. Figure B-3 contains an example of the toybox editor.

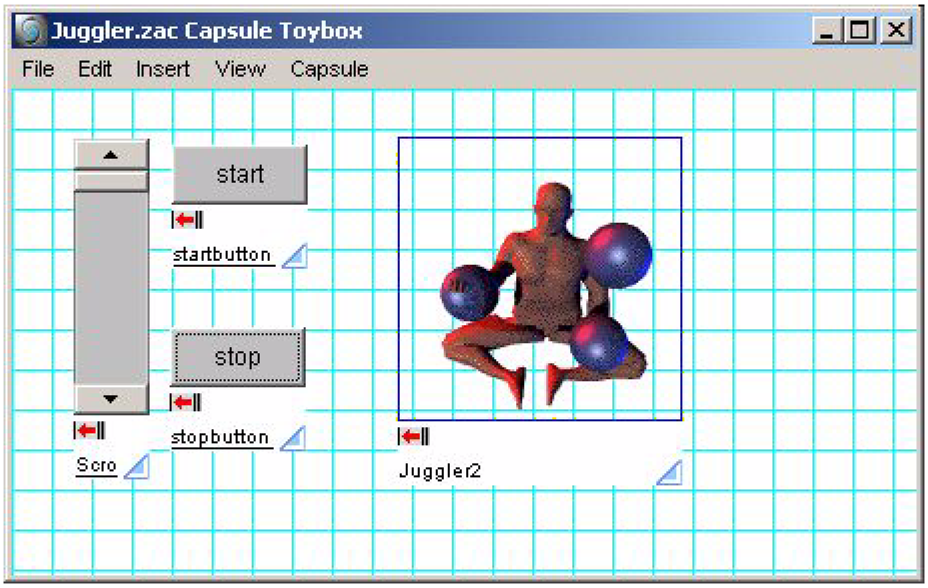


Figure B-3: Toybox Editor Window

Editing Handles

Each visual actor in the toybox view has an editing handle below it. You can hide the handle using View>Hide/Show Editor. You can move the actor by clicking and dragging on the editing handle. Rght-clicking on the editing handle (Command-click for Macintosh) displays a pop-up menu that is the same as the Capsule Outline Editor's Actor menu. (For details, see "Actor Menu" on page B-12.)

The editing handle consists of three parts:

- Icons representing the actor's immediate children. This part is displayed only if the actor has immediate children.
- The name of the actor. You can edit the name by double-clicking on it.
- The resize handle. This is a triangle on the right side of the editing handle that lets you change the size of the actor. Resizing the actor with this handle is equivalent to manually editing the actor's size property.

Drag and Drop or Copy Objects in the Toybox

You can drag and drop objects in the toybox to reorganize them, but only if they are child objects of a visual actor. You drag an object by dragging its icon onto the name tab of another object. That object and all its children then become children of the new object.

You can also create a copy of an object by holding down the Control key (Windows) or the Option key (Macintosh) while you drag.

Toybox Editor Menus

The menus in the toybox editor are the same as the menus in the capsule outline editor, except for the following additions to the toybox editor's View menu. Also, there is no Outline menu in the toybox editor.

View>Open Outline	Opens the capsule outline editor window.
View>Arrange	Moves objects to the front or back of the visual hierarchy. This menu item changes the order of the objects in the outline view as well.
View>Hide/Show Editor	Hides the editing handles in the toybox view.

Running versus Editing

The toybox view lets you play with a capsule while you edit it. When you add a new component to a capsule, it is instantly active: separate compile and run phases are not required.

When a capsule is running, however, Spin must distinguish between permanent changes to a capsule and those changes that occur only because the capsule is running. For example, if you add a drag behavior to an actor, you can play with the actor by dragging it, but you probably do not want these run-time changes to be a permanent part of the application. And they won't be.

The toybox view distinguishes between run-time changes, which are temporary, and edit-time changes, which are permanent. It does this by differentiating between changes caused by a behavior and changes caused directly by you as a user. For example, changes in an actor's position caused by a drag behavior are not permanent, because they are caused by a behavior (even though the drag behavior utilizes user input). However, if you move the actor by dragging its editing handle, that move is permanent.

Debugging Using an Error Window

When testing an application in the toybox view, if an error occurs, a window containing the error message opens. Double-click on the error message to open an editor on the code that caused the error.

Debugging Using the Console Window

Spin also has a console window. On Windows this is provided as text output in a DOS window. On the Macintosh, this is provided in a window opened by MRJ.

Some failures report only to the console. These might be problems in the tool itself, problems within a JavaBean that you are using, or simply text output printed to *System.out*. If you need to report any errors in the operation of Spin, please check for console window messages that can help us identify the problem and include the messages in your bug report.

To use the console window to help you debug your application, you can create script behaviors that contain commands such as:

```
System.out.println("got here");
```

When your script is activated, Spin will send your `got here` message output to the console.

Glossary

A

action behavior

A Spin behavior that receives an event as a stimulus and responds by invoking a method on an actor. See *behavior.* Compare *action group behavior, conditional behavior, counter behavior, script behavior, timeline behavior, user behavior.*

action group behavior

A convenience allowing you to group several Spin behaviors so they can be moved or copied as a unit. See *behavior.* Compare *action behavior, conditional behavior, counter behavior, script behavior, timeline behavior, user behavior.*

action target

An actor to which a method is sent. See *actor.* Compare *activation event.*

activate

To send the stimulus to a behavior that causes it to execute. See *behavior, stimulus.* Compare *deactivate.*

activation event

An event that activates a behavior. See *behavior, event.* Compare *action target.*

actor

A Spin component to which you can assign behaviors; Spin allows you to use any JavaBean component as an actor. Also, a Spin alias referring to the first actor above a behavior in the Spin capsule hierarchy. See *alias, behavior.*

alias

One of the three Spin pseudovariables `capsule`, `parent`, or `actor`, that refer to the specific entity fulfilling that role above the reference entity in the capsule hierarchy. See *actor, capsule, parent.*

applet

A small (typically) application that is downloaded in a web browser and executes on the web client. Compare *distributed application, servlet.*

AWT

Abstract Windowing Toolkit, the components used to build a user interface in Java applications. See *user interface.*

B

bean

A Java component. See *component.*

behavior

A Spin entity having a stimulus and a response, used to cause an actor to do something, to communicate with another actor, or to respond to user input. Because the stimulus is always an event, a behavior is way to ensure that a given event is followed by the specified method execution, property modification, or other event. See *actor, event, response, stimulus.*

browser view

The Spin view that allows you to see your servlet in a web browser as a user will see it. See *view.* Compare l*ayout view, outline view, project view, run view, toybox view.*

C

capsule

In Spin, a means of organizing an application into modules, each having a hierarchy of components. A capsule itself is a component that can be contained in other capsules. Also, a Spin alias referring to the first capsule above an entity in the Spin capsule hierarchy. See *alias.*

child

A Spin entity that is directly beneath another in the capsule hierarchy. See *capsule.*

client

A computer or program that requests resources from a server. For example, a web client (browser) requests web pages from a web server. A program can act as both client and server; for example, a web server acts as a client when it requests data from a database server. Compare *server.*

component

A persistent reusable building block of code that works as one functional unit and is can be used in various applications.

conditional behavior

A Spin behavior with an embedded *if* test. When a conditional behavior receives its stimulus, it evaluates the associated *if* test, generating an `ifTrue` event if the result is true, or an `ifFalse` event if the result is false. See *behavior.* Compare *action behavior, action group behavior, counter behavior, script behavior, timeline behavior, user behavior.*

connection pool

A group of database connections that remain open for any transactions that need them, so that a transaction can use them without the overhead of opening and closing them explicitly.

cookie

A text file sent by a web server, which the web browser stores on the client and sends back in response to requests from the same web server, used by Spin to store a session identifier. See *session, web server.*

counter behavior

A Spin behavior that counts time, starting when it receives its stimulus, and generates specified events at specified intervals until it reaches the specified stopping point. See *behavior.* Compare *action behavior, action group behavior, conditional behavior, script behavior, timeline behavior, user behavior.*

D

deactivate

To cause a behavior to stop executing. Compare *activate.*

deploy

To move a file into a production environment where it can be used. Typically this relates to making WAR or JAR files available to a server.

distributed application

An application whose parts run on more than one computer across a network. Compare *applet, servlet.*

E

editing handle

Small tabs attached to a Spin actor or behavior, visible in the toybox view, that allow you to modify the entity to which they are attached. See *actor, behavior, toybox view.*

EJB

Enterprise JavaBean, a robust bean that adds portable access to services such as database access, distributed transactions, and messaging. See *bean.*

event

An occurrence of possible interest to an application, such as a mouse click.

F

form parameter

A name and value pair derived from an input field of a web form.

H

HTML

hypertext Markup Language consists of sets of tags that mark the structure of a web page's contents, thus enabling a web browser to determine how to display the page.

HTTP

HyperText Transfer Protocol, the protocol used by web servers and browsers to request and return web pages and other data. See *web server.*

I

Internet

A worldwide network of computers connected by various means to allow the exchange of information.

J

JDBC

Java Database Connectivity, an interface that allows Java applications to access the data in a database.

JSP

Java Server Page, a kind of servlet stored in a web page. See *servlet.*

L

layout view

The Spin view that allows you to create and modify a scene. See *scene, view.* Compare *browser view, outline view, project view, run view, toybox view.*

link

A hypertext pointer from one web page to another.

logic layer

The programming code that manipulates the underlying data in a software program.

M

method

A sequence of Java statements attached to a component that execute when invoked, usually by sending the name of the method as a command to an actor. See *actor, component.*

MIME

Multipurpose Internet Mail Extension, a way of identifying the kind of content sent over a network, used by web servers to identify what is being sent to the web browser. See *HTML, web server.*

N

nonvisual actor

An actor with no visible representation on the computer display. See *actor.*

O

ODBC

Open DataBase Connectivity, an interface that allows Java applications to access the data in a database.

outline view

The Spin view of the capsule hierarchy. See *capsule, view.* Compare *browser view, layout view, project view, run view.*

P

parent

A Spin entity that is directly above another in the capsule hierarchy. Also, a Spin alias referring to the first entity above this behavior or actor in the Spin capsule hierarchy. See *alias, capsule.*

project view

The Spin view that lets you group all the capsules related to your current application. See *capsule, view.* Compare *browser view, layout view, outline view, run view, toybox view.*

R

response

The method that executes, or the property modification, or the event that is generated, when the stimulus of a behavior is received. See *behavior, event.* Compare *stimulus.*

run view

The Spin view that allows you to see your application exactly as a user will see it. See *view.* Compare *browser view, layout view, outline view, project view, toybox view.*

S

scene

A visible component that is a kind of container, such as a window. See *component.*

script

An arbitrary sequence of Java statements.

script behavior

A Spin behavior that allows the execution of any arbitrary chunk of Java code in response to an event. See *behavior.* Compare *action behavior, action group behavior, conditional behavior, counter behavior, timeline behavior, user behavior.*

serialize

To turn an executable entity such as a component into data that can be sent across a network or stored. See *component.*

server

A computer or program satisfying the requests of a client; for example, a web server offering resources over the World Wide Web. See *web server.* Compare *client.*

servlet

An application that runs on a web server and constructs appropriate responses to requests as they stream in from web browsers. See *web server.* Compare *client, distributed application.*

session

A means by which a web server can identify multiple requests as coming from the same web browser. See *client, server, web server.*

stimulus

The event that triggers a behavior, causing its response to execute. See *behavior, event*. Compare *response*.

T

timeline behavior

A Spin behavior that modifies the properties of one or more actors over time. See *behavior*. Compare *action behavior, action group behavior, conditional behavior, counter behavior, script behavior, user behavior*.

toybox view

The Spin view that lets you see your application run as you create and modify it. See *view*. Compare *browser view, layout view, outline view, project view, run view*.

U

URL

A Uniform Resource Locator is used to specify resources on the World Wide Web, such as web pages and servlets. See *HTTP, web server*.

user behavior

A named, saved, reusable Spin behavior. See *behavior*. Compare *action behavior, action group behavior, conditional behavior, counter behavior, script behavior, timeline behavior*.

user interface

Means by which users make their will known to an application—commonly, with windows, buttons, and menus.

V

view

A view allows you to see, and often to edit, a specific object. For example, Spin allows you to view and edit a capsule in several ways. See *capsule*. Compare *browser view, layout view, outline view, project view, run view,* toybox view.

visual actor

An actor with a visible representation on the computer display. See *actor*.

W

watch variable

A variable whose value is visible in the debugger, so that you can watch it change.

web page

A file whose contents are encoded with HTML, thus permitting access from a web browser such as Netscape Navigator or Internet Explorer.

web server

A computer more or less permanently connected to the Internet that offers to web users a set of resources—text, images, music, or other downloadable files. See *Internet*.

World Wide Web

An application that runs on the Internet: it consists of a network of computers that function as web servers. See *web server*.