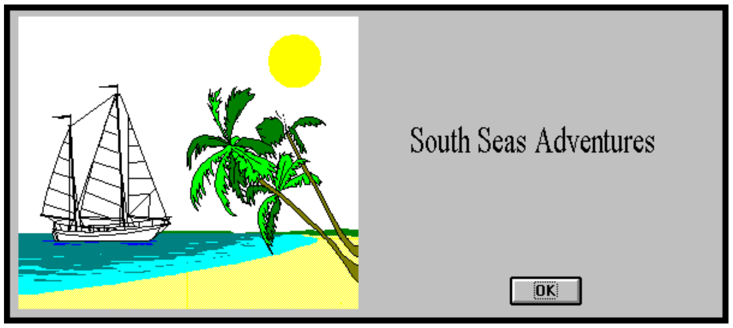
Chapter 1
Introduction

Welcome to South Seas Adventures!

Your adventure is to take a hands-on tour of a sample application, South Seas Adventures, and to discover how quickly and easily you can begin developing your own applications in CA-Visual Objects. The application has been partially developed—allowing you to complete the development process as you work through the tutorial.



South Seas Adventures simulates an operations support system that might be used by the employees of a hypothetical company, South Seas, Inc. Employees of the company act as booking agents to help customers plan an “adventure,” consisting of several vacation activities—such as parasailing, jet skiing, or dinner cruises.

Additionally, South Seas Adventures manages many other aspects of the business, including subsystems to handle customer, employee, order-entry, invoice, and payment information.

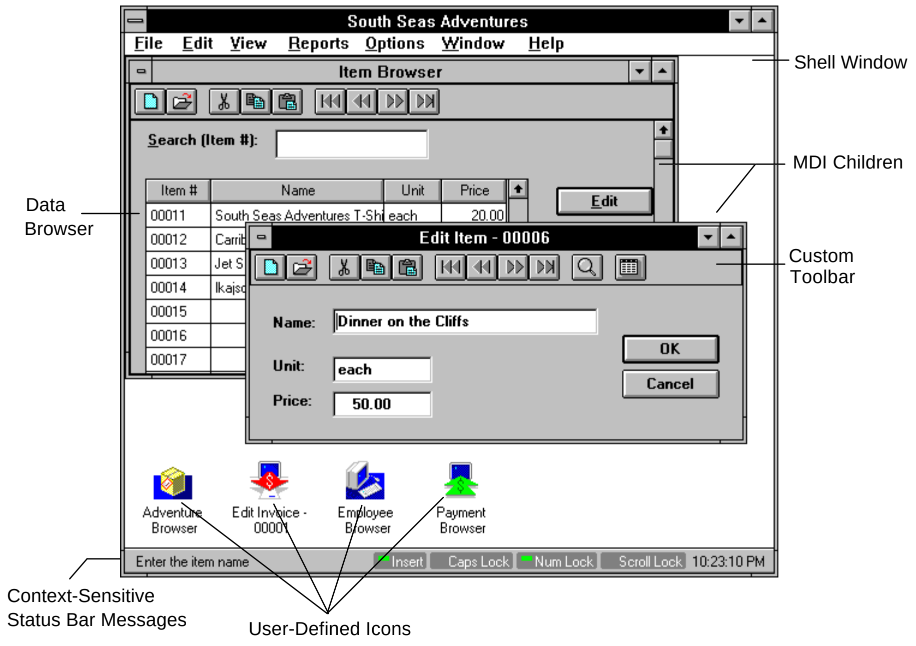
Some of the system operations that are covered in the tutorial are listed below:

- Creation of an order for an adventure
- Creation of an invoice
- Recording of payment information
- Preparing customer reports
- Exporting data to an external SQL-based accounting system

Not only does the application have all the proper components to satisfy the needs of South Seas, Inc. operations, it also employs many user-friendly features (such as windows and toolbars) found in your favorite Windows applications.

South Seas Adventures is an MDI (multiple document interface) application that features a consistent, easy-to-use interface that demonstrates how standard Windows features are created, using the CA-Visual Objects IDE (integrated development environment). The application allows you to view or browse through information easily, as well as create or delete a customer record.

Some of these standard Windows features are shown below:



Item #	Name	Unit	Price
00011	South Seas Adventures T-Shi	each	20.00
00012	Carrib		
00013	Jet S		
00014	Ikajsd		
00015			
00016			
00017			

Name: Dinner on the Cliffs

Unit: each

Price: 50.00

Enter the item name

Each of these annotated areas will be discussed in greater detail in the remainder of this chapter.

You will now discover how the application is developed—using the CA-Visual Objects IDE tools.

Using the Integrated Development Environment

CA-Visual Objects provides both a visual and incremental development environment. It integrates the many different types of building blocks, or entities, required to build a Windows application like South Seas Adventures. Visual editors allow you to create windows, menus, and other objects. Even more importantly, they automatically generate the related source code entities. You can then create and edit the relatively small number of source code entities that define what happens when a push button is pressed or a menu command is selected. Finally, you can create source code entities that incorporate business and detailed program logic rules.

CA-Visual Objects Repository

All of the entities are stored in the CA-Visual Objects repository. The repository allows you access to all of the entities associated with an application. The entities are organized into modules so that similar entities can be grouped together to make them easy to locate. There are several ways to view entities and modules in CA-Visual Objects:

- Module Browser

  Allows you to select a module and display a list of its entities.

- Entity Browser

  Allows you to see all the entities in an application, organized by entity type.

- Class Browser

  Allows you to see all the entities associated with a class.

The CA-Visual Objects IDE is well suited for incremental development. You can create the data servers, windows, and menus in any order. Once the interface is basically completed, you can start adding source code entities that define what happens when a push button or menu command is selected. The repository manages all the building blocks, compiles the changed entities, and allows you to run the application, so that you can revise the overall design and recreate the building blocks in an incremental fashion.

Creating the Application Building Blocks

You will have the opportunity to add some of the key building blocks to the South Seas Adventures application. Since certain exercises require the use of a building block created in an earlier chapter, you should work through the chapters in sequential order.

After you have completed the development of the application, it will be capable of accessing and updating information about customers, employees, adventures, activities, invoices, and payments. This is accomplished through the use of a variety data-aware windows, dialog windows, menus, and reports.

Here are the building blocks that you will create:

- A data server for customer data based on the traditional Xbase model of a .DBF file, and two SQL servers for data to be exported to an accounting system
- General field specifications, or *field specs*, that can be used with existing data servers
- A data-aware window to edit customer data, using the Auto Layout feature with a data server
- Controls for the customer data window
- A menu and toolbar for the customer data window
- An icon for the customer edit window
- A resizable opening dialog box with a bitmap and a text object
- A report that lists customers
- A library and DLL for .INI file management activities

What You Should Know

The South Seas Adventures application was created using an object-oriented development system. If you are not familiar with object-oriented systems, you can use the following background information to assist you in getting the most out of this tutorial.

What Are Objects?

Objects are special types of programming entities—like windows, controls, menus, or toolbars—that have properties associated with them (like a border or caption) and can perform specific actions (like displaying itself on the screen, in the case of a window, or toggling the check mark indicator, in the case of a check box control).

These animated objects are generally created, and ultimately destroyed, in response to something the end user does via the application interface. Every time a user accesses a data table and it is connected to the application, a data server and related field spec objects are created. Also, special objects that relate to a problem-specific business process (such as creating an invoice) may be created.

What Can Objects Do?

They can walk, talk, and listen! The developer can specify what actions are possible for each type of object; this is done by creating source code entities called *methods*. An object can also provide information for use by other objects or by the business logic routines in the application; the way an object talks is defined in *access* source code entities. An object can also receive information via *assign* source code entities, which can modify its current state.

How Are Objects Defined?

Objects are defined by a special kind of source code entity called a *class*. Approximately 90 percent of all classes you need are defined automatically by CA-Visual Objects. A class specifies the blueprint for each type of object and typically inherits most of its properties from another class, called the *parent* class.

However, in some situations you may want to define a class from scratch. A special Init() method defines all the related details of what must be done when the object is created. Additional methods can be defined to specify the other desired behaviors of the object, including what information can be received by it (assigns) or provided by it (accesses).

How Are Objects Created and Destroyed?

Objects are created in the runtime environment by the CA-Visual Objects runtime system. End-user actions trigger source code that creates a new object; this process is called *instantiation* since a new instance of the object is created. Each object is given an internal name by the runtime system to allow it to manage the activities and communications for all the existing objects. Objects are destroyed as a result of end-user actions. For example, the menu, data server, and controls for a data window are destroyed when the end user closes a window. All windows are destroyed when the end user closes the application.

How Do Objects Interact?

All actions and communications between objects and other variables are managed by the CA-Visual Objects runtime system. When an end user starts an application, a well-defined environment, or universe, is created. Once objects are created in response to end-user actions, all the behaviors and information transfers progress according to the source code specifics. Each object is a well-behaved automaton that does what is asked of it, speaks only when spoken to, and accepts only certain kinds of information. An end user is the invisible hand that triggers the creation, activities, and destruction of each object.

When Does It All Begin and End?

The runtime universe begins when the user starts the application. This triggers the App:Start() method, which is similar to CA-Clipper’s

START() function. The App:Start() method defines the shell window to be opened, and then triggers the running of the application with the App:Exec() method. The end user then defines all the other events, including the creation and destruction of objects. Finally, the application is closed when the App:Quit() method is activated.

How Are Objects Used in South Seas Adventures?

If you have not used objects before, you should be reassured by the fact that approximately 90 percent of all the classes, methods, accesses, and assigns in the South Seas Adventures application are generated automatically by CA-Visual Objects. Every time you use one of the visual editors and save whatever it is you are creating, all the related source code entities are generated.

This tutorial also highlights the few classes that are not generated automatically. These classes include dialog window classes that inherit properties of dialog windows generated by the Window Editor, an Invoice class created from scratch, and two file specification classes.

Creating Objects

When you create an object, it is named in the source code like any other variable, but the letter “o” is used as a prefix for the name. For example, since “oDC” indicates that an object is a data-aware control, a single-line edit control on a window could be named oDCFirst_Name. It would then be instantiated with the following source code:

```
oDCFirst_Name := SingleLineEdit{...}
```

This control is instantiated from the parent class (the standard SingleLineEdit class) according to the details and parameters within the {} braces. These curly braces indicate that an object is being instantiated.

Working with Objects

You can access character information from the object, by using the Value access of the parent SingleLineEdit class:

```
cTempName := oDCFirst_Name:Value
```

Or you can assign character information to the object with the Value assign of the parent class:

```
oDCFirst_Name:Value := cNewName
```

If you want to clear the contents of this control, use the name of the object followed by the Clear() method defined for the parent class, as follows:

```
oDCFirst_Name:Clear()
```

Finally, remember that all the interactions between objects are handled automatically by the CA-Visual Objects runtime engine. Once the end user starts the application, all these details should be of no concern to the developer.

That is all you really need to know about objects to benefit from the South Seas Adventures tutorial. Remember that as you create data servers, windows, and menus with the visual editors, the related object-oriented code is generated automatically. That means that the majority of your time as a developer can be spent designing the interface and business logic, while you leave the intricacies of creating a Windows application to CA-Visual Objects.

South Seas Adventures Application Design

Let’s now begin to take a closer look at the internal workings of the South Seas Adventures application by examining its various components. This section can be used as a road map to what follows in the tutorial.

Creating the Primary Building Blocks

Entities that are created by the CA-Visual Objects visual editors are discussed in this section. You will learn how easily they can be created and incorporated into an application.

Data Tables, Servers, and Fields

The South Seas Adventures application provides access to information about customers, employees, adventures, activity items, invoices, and payments—the six primary types of business data used in the application—by means of the following data structures:

Data Tables

The types of tables that are needed to store primary (and other types) of data are listed below:

- Individual DBF tables for storing customer, employee, item, and payment information.
- Paired DBF tables for adventures and invoices (one for header information and the other for a varying number of detail records).

  For example, an adventure is defined for a customer whose vacation travel spans a fixed period of time (header information) and contains one or more adventure activity items (detail information).

- SQL tables for accounting invoices (AccInv) and accounting payments (AccPay), simulating the external accounting system.
- DBF lookup tables (State, SysKey, and Tender).

Data Servers

A data server must be created as an interface to each of these tables. You can use the DB Server and SQL Editors to accomplish this task quickly and easily. The IDE automatically creates the server entities and all the related field spec entities when you save a server from its editor.

The South Seas Adventures application requires data servers for the DBF tables and SQL data servers for the accounting tables. You will see how the Customer, AccInvc, and AccPay servers are created in the "Working with Data Servers" chapter.

Data Fields

When a data server is created, the data fields for each server are defined in either the DB Server or SQL Editor. Typically, the server is created by importing a STRUCTURE from a .DBF file or an SQL table. If this is not done, the name, length, and type must be specified for each data field. The related field spec entities for each data field are automatically created when you save the server.

You will get the opportunity to examine the field specifications for various data fields within the South Seas Adventures application.

Data Windows

Data windows not only provide access to information that you need—but also allow the user to create, edit, and delete records. All the related source code entities needed to perform these functions (classes, methods, accesses, assigns, and defines) are created automatically when you save your window design.

The following data windows are used in the South Seas Adventures application:

- Browser window

  Contains a subform that displays a table showing several data fields for all records and an edit control for searching the table. One table is used for each business data type.

- Subform window

  Contains a multi-column tabular display for use in the browser window. One table is used for each business data type.

- New window

  Used to create a new record. It applies to all business data types, except invoices (since invoices are created from the Edit Adventure window).

- View window

  Used for viewing payment information, which is not editable.

- Edit window

  Used to edit all fields of a single record. It applies to the customer, employee, and item business data types.

- Master-detail edit window

  Used to display header information and a detail subform. It applies to adventure and invoice business data types.

- Detail subform

  Contains a tabular display for use in a master-detail edit window. It applies to adventure and invoice detail data.

You will learn how to create the Edit Customer data window as part of this tutorial.

Shell and Dialog Windows

A shell window provides the framework for all the other windows and menus in your application. Dialog windows provide a way to communicate with the end user. These windows are also designed using the Window Editor and all the related source code entities are created automatically when you save the window design.

The South Seas Adventures application contains a single shell window and several dialog windows, including:

- Shell window: SSAWindow
- Navigational dialog windows: Opening, FileNew, FileOpen
- Print dialog windows: CustAdv, CustRpt, InvcRpt, PayRpt, Printer, PrintReport
- Other dialog windows: About, Find, Login, NewPassword, Progress

Window Controls

CA-Visual Objects has many types of controls that can be used in a window design. These controls are derived from standard classes. When you save a window design, the required source code for each control is automatically generated. Alternatively, you can make use of these controls by writing source code directly.

The majority of the controls are called “data-aware” controls, because they can be directly linked to a field in a data server, and therefore, can directly communicate with a field in a data table. When the Window Editor generates source code, the object name is prefixed with the three letters, “oDC”.

A few types of controls are not data-aware—for instance, push buttons, radio buttons, fixed text objects, fixed icons, and group boxes. Object names for these controls are prefixed with “oCC”.

The “Adding Controls to Your Windows” chapter steps you through the process of placing controls on the Edit Customer window created in “Creating and Using Windows.”

Application Menus

Menus allow a user to communicate with an application to perform an action. The Auto Layout feature of the Menu Editor can be used to get a quick start on your menu design. You can then add or delete menu items. You can also define a toolbar for the menu. When you save a menu design, the Menu Editor automatically generates all the source code entities for the menu.

The SSAShellMenu menu is the one associated with the main shell window that you see when the application begins. It allows you to select the next window to be opened. This *child* window has an associated SSAChildMenu menu that invokes various data windows.

In the “Creating Menus and Toolbars” chapter, you will create the CustomerMenu menu, which will then be attached to the Edit Customer window.

Event Handlers

Once the primary building blocks are created, you can begin to write the source code that defines what should happen, for instance, when the user activates a push button or a particular menu command. A Window class method (or event handler) can be created that holds the code that is executed when a push button is clicked. The Window Editor allows you to write this source code during a work session, by invoking the Source Code Editor. In the Menu Editor, you must specify the name of a window, report, or method for each menu item.

In addition, you can create special methods that are activated when certain events occur in a window (for example, when any button on a window is clicked or the contents of any edit control is modified). This type of source code must be included in special window event handler methods, which can also be done easily during a Window Editor work session.

You will learn about window event handlers for the South Seas Adventures application in the "Customizing Window Event Handlers" chapter.

As you define the event handlers for your controls and menu items, the building blocks of your application are linked together. At this point in the development process, you can begin to run the application from the IDE to see how the various components work together.

Reports

Reporting can be a powerful feature in a business application. The CA-Visual Objects Report Editor allows you to specify which tables are to be used in a report, by defining the data server names. Once you select a report style—tabular, form, labels, letter, or freestyle—a basic report definition is generated which you can then modify. When you save the report definition, an external CA-RET file is generated, as well as several related source code entities.

There are several reports that are supplied with the South Seas Adventures application that you can examine. In the "Reporting with CA-RET" chapter, you will also create a report that lists all customers.

Help Systems

Another primary building block of an application is the help system. This can be an important part of making your application easy-to-use.

Context-sensitive help can easily be designed into your application. First, you must create a help file that has all the required help topics. You must then attach the help file to the application by specifying the help file name as a property of the shell window. An individual help topic can be designated in the visual editors for each window, control, menu item, server, and field. Thus, there are many linkages between the application and the help file.

You will add help to the South Seas Adventures application in the "Adding Help to Your Applications" chapter.

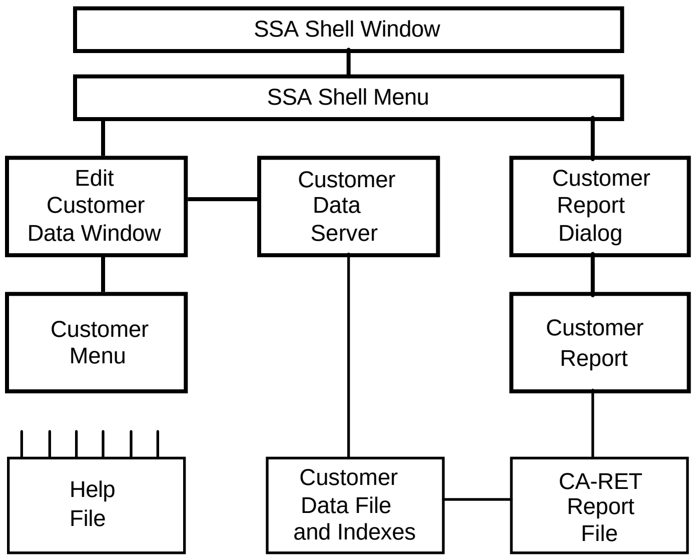
Icons, Cursors, and Draw Objects

In addition to the primary building blocks, there are a few more entities that you can add to your application that can enhance the "look" of your application, including icons and cursors.

You will create an icon (MyIcon) in the "Creating and Using Icons and Cursors" chapter of this guide. You will also see how to work with bitmaps and text objects in "Working with Draw Objects," and learn how the Window:Draw() method can be used to display them.

Linking the Primary Building Blocks

The following diagram illustrates the basic structure of the South Seas Adventures application and the relationships among the primary building blocks (those that can be created using the editors in the CA-Visual Objects IDE):



Note: Rectangles with thick borders are the primary building blocks, while those with thin borders are external files. Design linkages are shown as thick lines, while external linkages are shown with thin lines.

The main application window is the shell menu, which allows you to open a data or dialog window. Menus are associated with windows, and data servers act as the interface between data windows and external data files. There can be many linkages to the help file. Finally, each report is linked to an external report definition file which provides a linkage to the data files.

Completing the Remaining Building Blocks

At this point, you have learned how the CA-Visual Objects visual editors help simplify the development process when creating many of the required application entities. Now, let's look at some of the remaining building blocks that you will need.

The following diagram summarizes how the different types of application entities are created:

Visual Editor Design Entities	**Developer-Coded Entities**	**System-Generated Entities**
Cursors	Functions	Defines
Data servers	Global variables	Resources
Field specs	Classes	Classes*
Icons	Accesses	Accesses*
Menus	Assigns	Assigns*
Reports	Methods	Methods*
SQL servers		
Windows		

Aside from the visual editor design entities, the remaining entities are created either by the developer or the system. You do not need to be concerned with the system-generated entities, since they are generated automatically.

Depending on the amount of customization you want to make to your interface design, and the complexity of the event-handler methods, about one-third to one-half of your development time may be spent on creating the visual editor-generated building blocks, while the remaining time can be spent developing your own custom code.

The next two sections describe developer-coded and system-generated entities in greater detail.

Developer-Coded Entities

Your part in the development process lies in the creation of methods—such as event handlers—that describe what should happen when an end user clicks on a push button or selects a menu command.

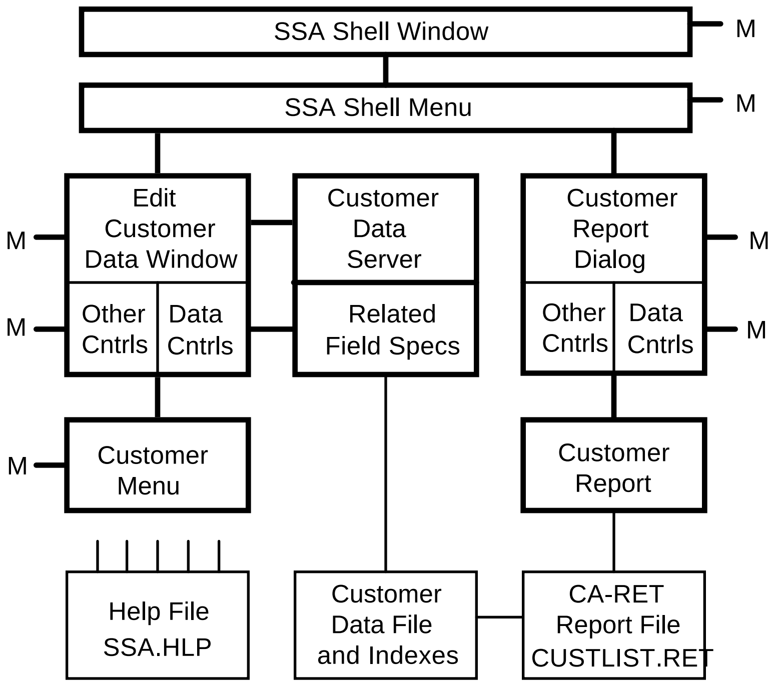
In the South Seas Adventures application, subclasses for the shell window and the dialog windows are used to demonstrate how code reuse can greatly reduce your overall development time. (Subclassing is discussed in greater detail in the “Inheritance and Subclassing” chapter.) A few special window event handlers are also used, as you will see in the “Customizing Window Event Handlers” chapter.

System-Generated Entities

Entities that are generated automatically by CA-Visual Objects typically represent approximately 80 to 90 percent of the entities for an application like South Seas Adventures. The define entities make up about a third of the entities in the entire application. The vast majority of the other system-generated entities—including classes, access methods, and assign methods—are automatically generated by the IDE visual editors (as indicated by an asterisk in the previous table).

Linking the Remaining Building Blocks

The following diagram illustrates the entire structure of the South Seas Adventures application, including the supplementary building blocks, and the relationships among all the entities:



Key: M = method (event handler, button, or menu event)

Note: As noted earlier, rectangles with thick borders are the primary building blocks, while those with thin borders are external files. Design linkages are shown as thick lines, while external linkages are shown with thin lines.

This diagram expands upon the primary building block diagram, showing several additional relationships among the application entities.

It shows the controls that you can place on each window when you are using the Window Editor. There are two types of controls—data controls, which could be linked directly to the fields in a data server (and, therefore, to a data file) and other controls, which are not data-aware. These other controls are used to accept user-control actions (associated with push buttons and radio buttons) or as inanimate display objects (such as, group boxes, fixed icons, and fixed text).

Secondly, the important role played by field specs is shown as the rectangle adjacent to the data server. Note the design linkage to the data controls (via the name of the data server associated with a window). A dialog window may contain both types of controls, but there is no connection between a dialog's data controls to the fields in a data server and data file.

Lastly, you will notice the areas where the developer can explicitly code methods to describe how the application should act when the user makes certain control choices. These methods fall into the developer-generated entity category. Each place where a developer-coded method could be created is indicated by the letter "M." There are three types of such event-oriented methods:

- Push button methods

  Can be added to indicate what should happen when the user clicks on a specific push button (refer to the "Creating and Using Windows" and "Adding Controls to Your Windows" chapters for more information).

- Menu event methods

  Can be added to indicate what should happen when the user selects a specific menu command. As discussed in "Creating Menus and Toolbars," each menu item has a related event, which can either be the name of a window, report, or method.

- Event handler methods

  Can be added to any type of window. These methods are activated when the user takes certain actions on a window, such as clicking a mouse button or changing the contents of any edit control on the window (refer to "Customizing Window Event Handlers" for more information).

Begin Your South Seas Adventure...

Now, it is your turn to explore the world of CA-Visual Objects. The tutorial exercises take some time to work through, so try to complete a few chapters in each work session. Most importantly, you should complete the chapters in order, since certain exercises depend upon the building blocks created in earlier chapters.

Enjoy your adventure!