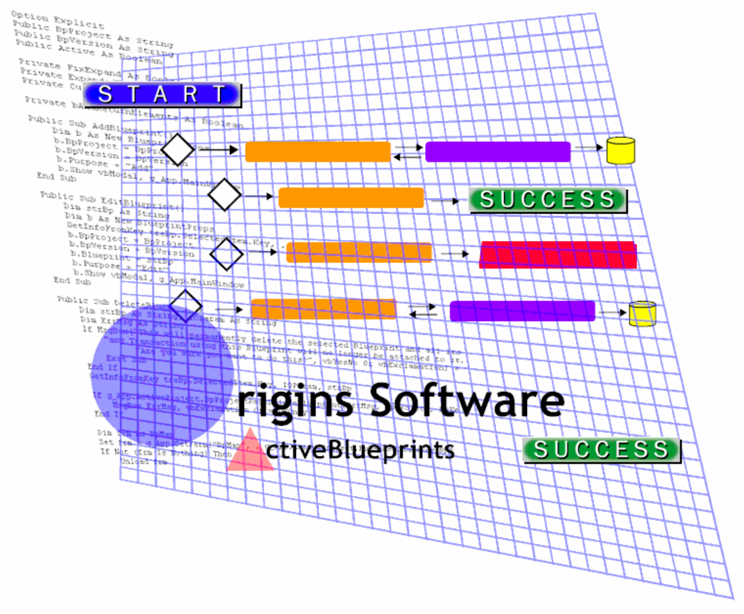


ActiveBlueprints
EasyObjects 1.0
Tutorial and User's Guide

origins

Software Company

2285 Massachusetts Avenue Suite 201
Cambridge, MA 02140
(617) 661-6700

Table of Contents

Chapter 1: Overview

EasyObjects for *Access* is a code generator used for rapid, accurate generation of database access COM components. This version works with the *Access* database. Origins will shortly offer additional *EasyObjects* products that work with all other popular databases, using the same interface. The goal of the *EasyObjects* series of products is simply to create high quality, bug-less objects rapidly and thus save you time and effort.

The development process consists of three easy steps:

1. Create the needed *Access* database using *Access* tools.
2. Generate the data access components using the *EasyObjects* code generator.
3. Test the components generated.

The manual’s tutorial contains examples of objects and specific testers for those objects. The testers are both in Visual Basic (VB) and in Active Server Pages (ASP). You can easily edit the testers to test any object generated by the system. You need a Visual Basic compiler to run the VB applications. In addition, to run the ASP pages you need a browser and a web server (IIS or Personal Web Server).

The Database

The first step is to create the Database. Use the *Access* database tools to create the database tables and relations you need.

EasyObjects

Once the database is defined, *EasyObjects* can create components that operate on the individual database tables. Each component compiles into a single unit (DLL) and consists of multiple *services*. Each service is a function or a method of the component.

There are four classes of services:

1. *Add* a record to a database table.
2. *Delete* a record from a database table.
3. *Search* for one or more records in a database table (and select one or more records).
4. *Update* a record in a database table.

The tutorial in Chapter 2 teaches you how to create all four types of objects.

Testing

The component testers included with the tutorial test the objects and demonstrate how the objects can be used. They check the example objects in the tutorial for proper performance. You can also use the testers as part of your learning after you complete the tutorial. To do so, edit the testers to use them with all of the components you create.

What’s Next

The components generated are sophisticated, they are fast to build and will save you time. After you generate the components you need to build the user interface—with ASP/HTML web pages or VB forms—to complete your application. The interface calls the objects to perform all the database access functions. You will have a three-tier application consisting of an interface (not automated), the generated components, and the database (see Figure 1).

EasyObjects is a member of a larger family of visual software development tools products called *ActiveBlueprints*. ActiveBlueprints can generate the full application for you if you wish.

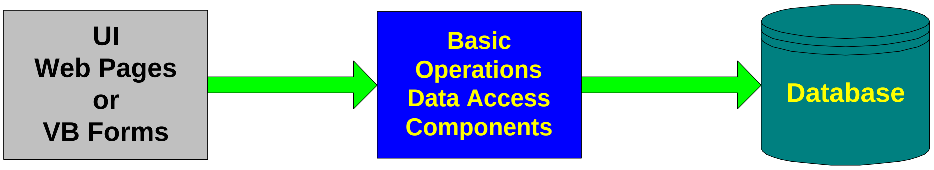


Figure 1: Three-Tier Application Using ActiveBlueprints

Chapter 2: Tutorial

This section provides a step-by-step tutorial about how to use the ActiveBlueprints (ABp) development environment to create database access components. The tutorial is meant to teach basic ActiveBlueprints development concepts as opposed to showcasing the full capabilities of the system. Because ActiveBlueprints introduces new development methods, even experienced programmers may find completion of the tutorial helpful. For the sake of clarity and convenience, the ABpTutorial Databases folder included with the tutorial contains both of the databases used in the lessons.

Terminology and Conventions

The tutorial lessons contain the following icons and terms:

⌨	Data entry using the keyboard.
🖱	Option selection using the mouse.
double-click	Two rapid (successive) **left** mouse button clicks.
right-click	Click the **right** mouse button while the mouse is over a specified object and a popup menu appears. Select an item on the menu.

Organization

The tutorial is composed of several lessons divided conceptually into two sections. In *First Steps* we cover the basics involved in using ActiveBlueprints to create a simple Hello World database access component. We introduce the Integrated Development Environment (IDE) for ActiveBlueprints EasyObjects as well as define some essential terminology. In *Walking* we extend the concepts and procedures learned in the first lesson to build a more complex database access component. This section focuses on database connectivity: how to use ActiveBlueprints to add, delete, update, and search for records in a database.

Because the skills and concepts in later lessons build on material learned in earlier ones, you should go through the lessons in order. Below is a list of the lessons in the tutorial:

Lesson 1: Hello World from a Database

Lesson 2: Add a Record to a Database

Lesson 3: Delete a Record from a Database

Lesson 4: List All the Records in a Database

Lesson 5: Update a Record in a Database

First Steps

The first steps are usually the most difficult ones when trying to learn something new. With respect to ActiveBlueprints, the main difficulty lies in understanding the terminology and how the terminology relates to the user interface. To ease the pain of making the first steps, we begin by creating a simple component that retrieves the message “Hello World”. During this process, we introduce the terminology and graphical user interface elements—such as menus, wizards, and dialog boxes—that make up ActiveBlueprints EasyObjects.

Lesson 1: Hello World from a Database

Objective: *Using a simple database that contains one record, retrieve the message “Hello World” from a database.*

Concepts: *ActiveBlueprints EasyObjects, Component Library, Component, Service, Service Elements.*

Resource: *HelloWorld.mdb*

This example introduces the **ActiveBlueprints EasyObjects** application. ActiveBlueprints EasyObjects is a companion application to ActiveBlueprints used to create Component Libraries, register Component Services, and create Database Access Components.

Using ActiveBlueprints EasyObjects

Use ActiveBlueprints EasyObjects to create and manage **Component Libraries**. A Component Library is a collection of Components used within ActiveBlueprints. The file extension for Component Libraries is **.abc**.

Create a Component Library

To create a Component Library:

A. Launch the ActiveBlueprints EasyObjects application. A shortcut has been provided in the Windows **Start** | **Programs** option under the submenu **ActiveBlueprints**.

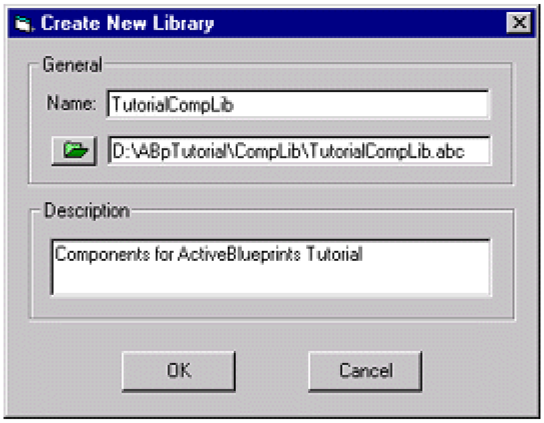
B. With the ActiveBlueprints EasyObjects application open, select **File** | **New Library** from the main menu. The [**Create New Library**] dialog box appears (Figure 2).

C. Fill the [**Create New Library**] dialog box with the following information:
⌨ Name (*of Library*): `TutorialCompLib`
🖱 📂 (*pathname of Library*):
`D:\ABpTutorial\CompLib\TutorialCompLib.abc`

This assumes that the directory *D:\ABpTutorial\CompLib* exists. If it does not, then either create it or place the Component Library somewhere else. The location of the library is arbitrary.

⌨ Description: `Components for ActiveBlueprints Tutorial`

D. Click <OK> to create a Component Library.



Name: TutorialCompLib

D:\ABpTutorial\CompLib\TutorialCompLib.abc

Components for ActiveBlueprints Tutorial

Figure 2: New Component Library Dialog Box

Create a Database Access Component

A Database Access Component (DAC) is an object that accesses information in a database. ActiveBlueprints EasyObjects uses a module called the Database Object Generator to create DACs. The Database Object Generator can create Add, Delete, Update, and Search **Services** for a single table in a database. A Service is a part of a Component that performs some action. A Service may have inputs, outputs, both, or neither. In most cases, however, DAC Services will have at least a return value (ReturnValue) and return message (ReturnMsg). The input and output functions of a service are called **Service Elements**.

To create a Database Access Component:

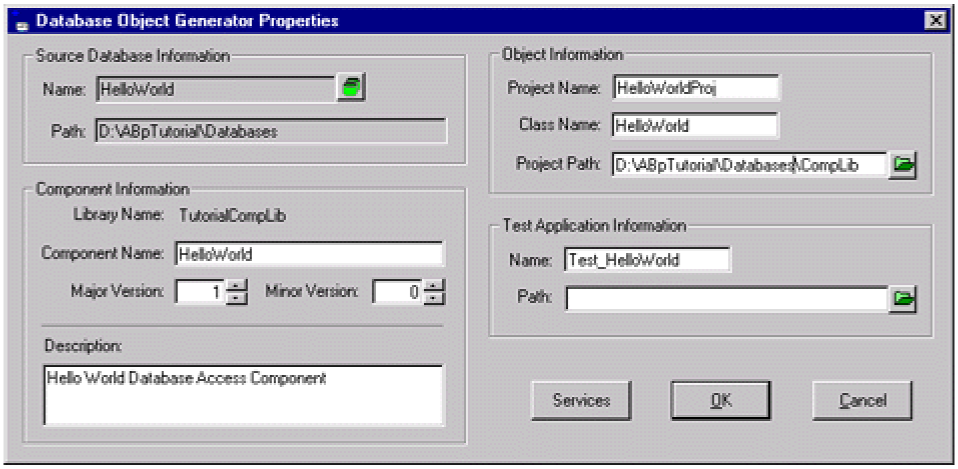
A. Select **Components** | **Create Database Component** from the main menu. The [**Database Object Generator Information**] dialog box appears.

B. Select the (Browse Database) button and open the `HelloWorld.mdb` file included in the ActiveBlueprints Tutorial Databases directory. `HelloWorld.mdb` is a Microsoft Access database. It contains a single table with a single record.

C. The edit boxes in the [**Database Object Generator Information**] dialog box automatically fill with valid information (Figure 3). To sharpen the description, change the text to:

⌨ Description: `Hello World Database Access Component`

D. Click <OK> to create a new Database Access Component. The Component Explorer is updated with a new HelloWorld v1.0 icon representing the newly created component (Figure 4).



Database Object Generator Properties

Source Database Information

Name: HelloWorld

Path: D:\ABpTutorial\Databases

Component Information

Library Name: TutorialCompLib

Component Name: HelloWorld

Major Version: 1   Minor Version: 0

Description:

Hello World Database Access Component

Object Information

Project Name: HelloWorldProj

Class Name: HelloWorld

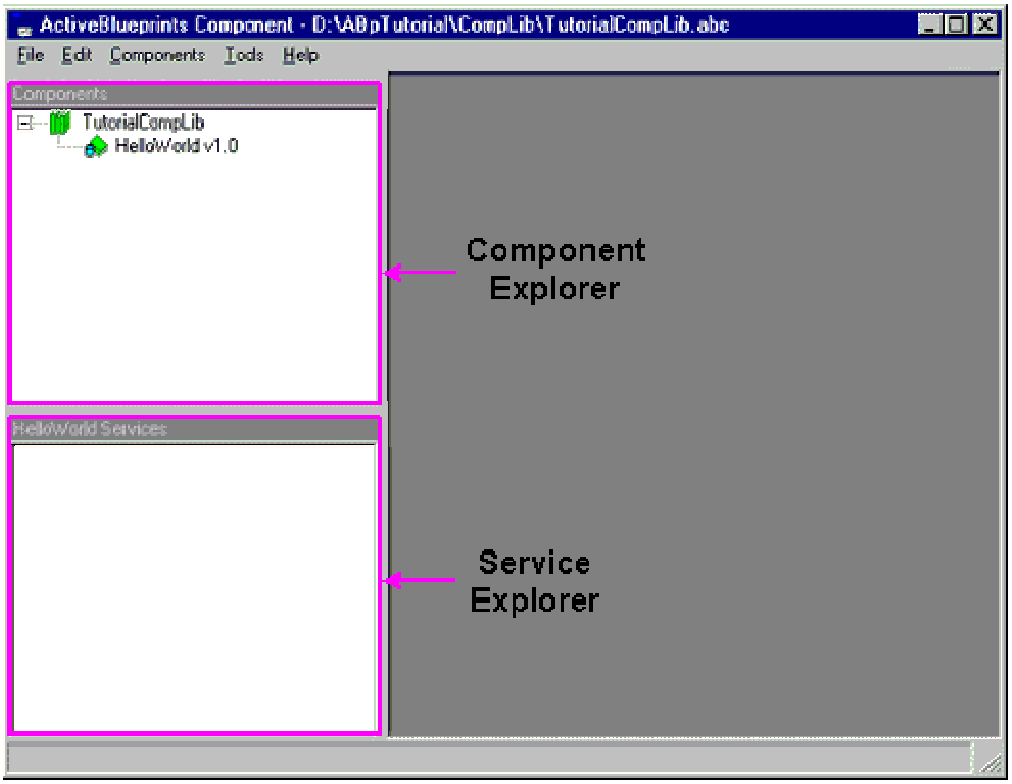
Project Path: D:\ABpTutorial\Databases\CompLib

Test Application Information

Name: Test_HelloWorld

Path:

Figure 3: Database Object Generator Properties Dialog Box



ActiveBlueprints Component - D:\ABpTutorial\CompLib\TutorialCompLib.abc

Component Explorer

Service Explorer

Figure 4: ActiveBlueprints EasyObjects Development Environment

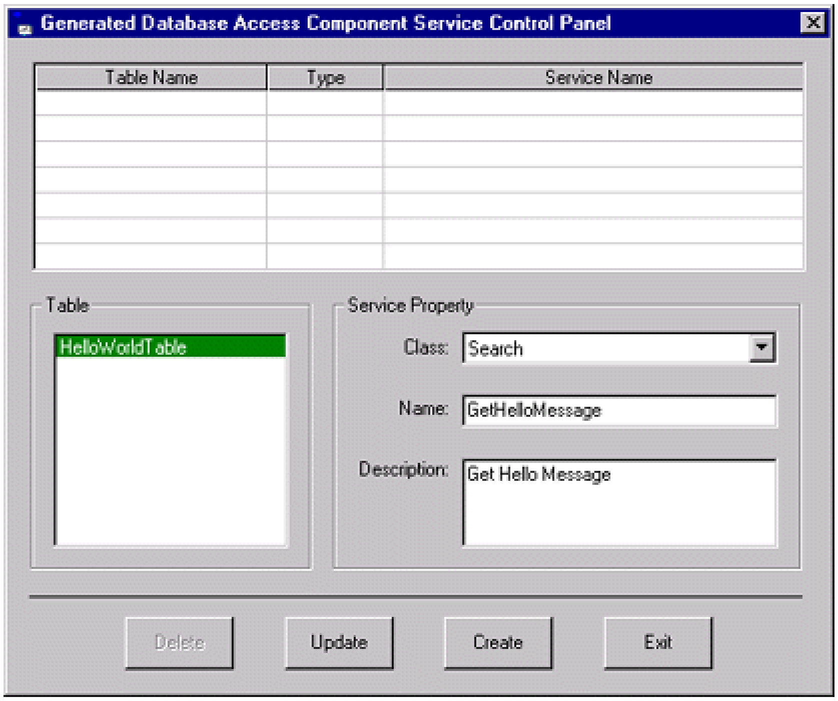
Add a DAC Service to a Component

A Database Access Component is created without any Services. Services must be added to a DAC to make it useful. In this example, we use the Database Object Generator to create a DAC Service to retrieve a message from a database. To create this DAC Service:

A. Right-click the HelloWorld v1.0 component icon (◆) and select **Generated Services** from the popup menu. A [**Generated Database Access Component Service Control Panel**] dialog box appears (Figure 5).

B. Fill out the [**Generated… Service Control Panel**] with the following information:
   - 🖱 Table (*database table accessed by the DAC Service*): `HelloWorldTable`
   - 🖱 Class (*select the type of Service to create from the pick list*): `Search`
   - ⌨ Name (*of Service*): `GetHelloMessage`
   - ⌨ Description: `Get Hello Message`

C. Click <Create> to initiate the Database Object Generator Service creation wizard.



Generated Database Access Component Service Control Panel

Table Name	Type	Service Name

Table: HelloWorldTable

Service Property

Class: Search

Name: GetHelloMessage

Description: Get Hello Message

Figure 5: Generated Database Access Component Service Control Panel

D. For a Search Service, the screen that appears is the [**Criteria Editor**] (Figure 6). The [**Criteria Editor**] is used to define the search criteria for a database query.

The table we want to extract data from contains a MsgType and a Message field. We want to query the database for the data in the Message field where the MsgType is equal to the literal string “HelloWorld”. To do so:

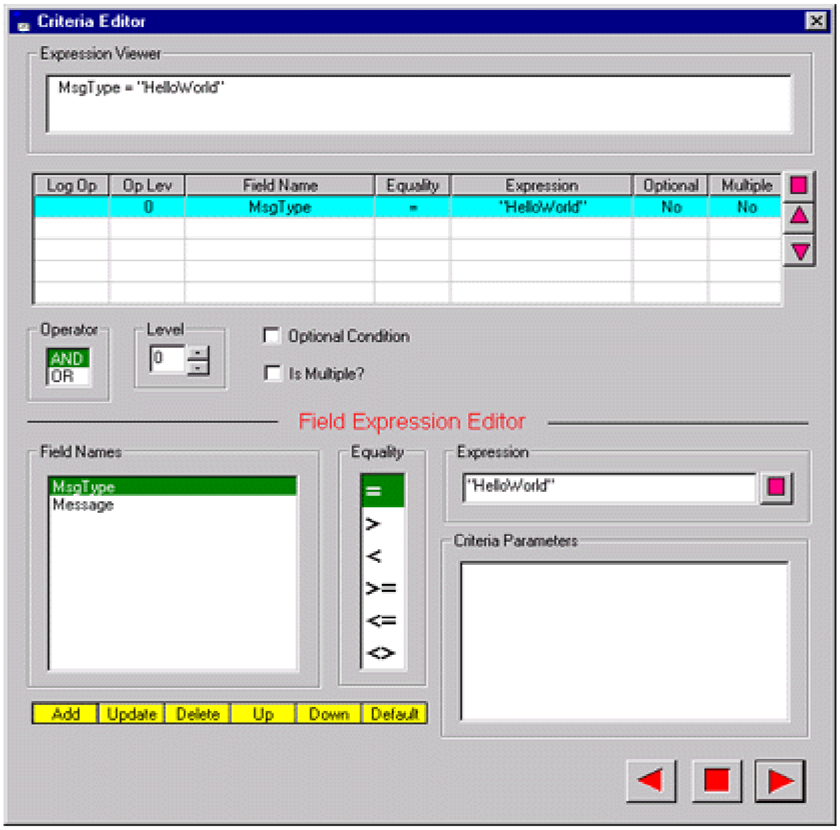
1. Enter the following items in the [**Criteria Editor**] dialog box:

   - Field Names: `MsgType`
   - Equality: `=`
   - Expression: `“HelloWorld”`

   Double-click a field name in the <Field Names> list box to list that name as a Criteria Parameter in the <Criteria Parameters> list box. Use Criteria Parameters to make dynamic queries. Since we want to search statically for the data in the Message field where the MsgType field is equal to “`HelloWorld`”, make certain there are no Criteria Parameters defined. If there are, right-click each parameter in the <Criteria Parameters> list box and select **Delete Param**.

2. Click <Add> in the yellow toolbar near the bottom of the display. The <Expression Viewer> and Logical Condition table under the <Expression Viewer> are updated with the desired search criterion.

E. Press the ▶ (Forward) button to advance the Wizard to the [**Display Editor**] dialog box.



Criteria Editor

Expression Viewer

MsgType = "HelloWorld"

Log Op	Op Lev	Field Name	Equality	Expression	Optional	Multiple
	0	MsgType	=	"HelloWorld"	No	No

Operator: AND / OR

Level: 0

□ Optional Condition

□ Is Multiple?

Field Expression Editor

Field Names: MsgType, Message

Equality: = > < >= <= <>

Expression: "HelloWorld"

Criteria Parameters

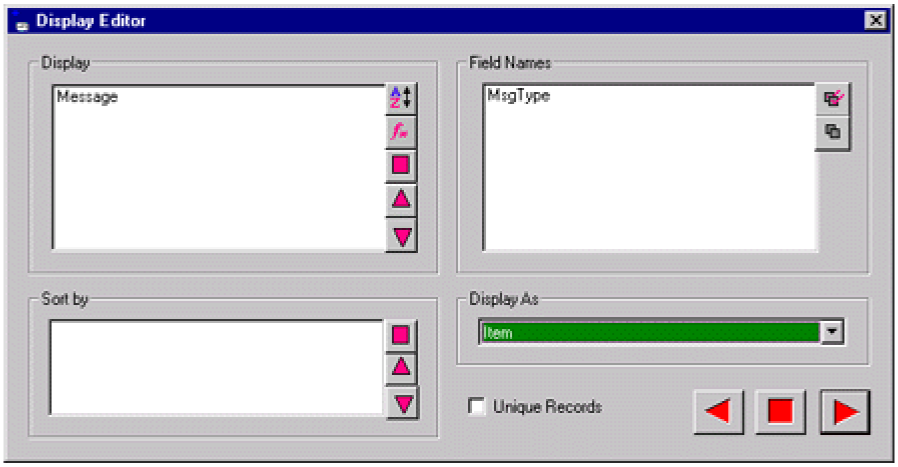
Figure 6: Criteria Editor

F. Use the [**Display Editor**] to specify the display format, the order of the fields returned, and the sorting criteria for the records returned (Figure 7). In this exercise, we want to return the single message retrieved with the expression defined in step D. To accomplish this task:

1. Double-click Message in the <Field Names> list box. The field Message moves over to the <Display> list box, indicating that the data in the Message field will be returned from the Service.
2. In the frame <Display As>, select the following from the pick list:

   ☞ Display As: `Item`

   When you select Item, the Search service returns the contents of the Message field as a single text string, not a list of text (a Collection) or a table of text (a Recordset). The Data Object Generator knows to use a text string as opposed to a number or a date because it reads the data type definition for the Message field found in the database. The Data Object Generator always determines the data type to use, which eliminates the chore for us.



Display: Message

Field Names: MsgType

Display As: Item

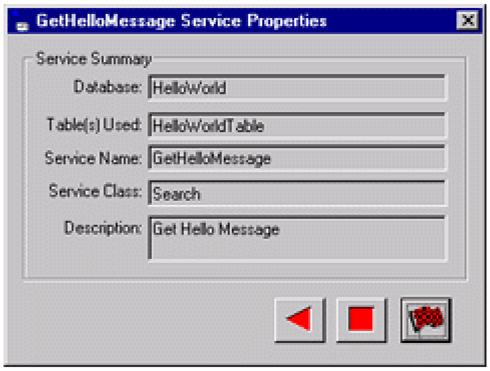
□ Unique Records

Figure 7: Display Editor Dialog Box

G. Press the ▶ button to advance the Wizard to the [**GetHelloMessage Service Properties**] display (Figure 8). This display summarizes important information about the Service you have created.

H. Press the (Finish) button to add the GetHelloMessage Service to the HelloWorld Component.

I. Click <Exit> to close the [**Generated…Service Control Panel**].



GetHelloMessage Service Properties

Service Summary

Database: HelloWorld

Table(s) Used: HelloWorldTable

Service Name: GetHelloMessage

Service Class: Search

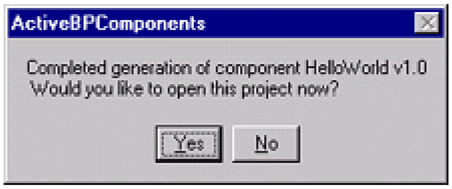
Description: Get Hello Message

Figure 8: Service Properties Summary

Generate Database Access Component

A. To generate the HelloWorld Database Access Component, right-click on the HelloWorld icon in the Component Explorer and select **Generate Database Component** from the popup menu. ActiveBlueprints generates and compiles the source code for the component.

B. An option to open the Visual Basic project for the HelloWorld Component appears after the component is successfully compiled (Figure 9). Select <Yes> to bring up the Visual Basic IDE with the HelloWorld Component code. Select <No> to bypass this option (it is not necessary to view the code to continue with the tutorial).



ActiveBPComponents

Completed generation of component HelloWorld v1.0
Would you like to open this project now?

Figure 9: View Source Code Dialog Box

First Steps Summary

First Steps introduces ActiveBlueprints EasyObjects. The first lesson defines basic terms and concepts, and uses the application to create a simple Database Access Component. The Component was purposely made simple in order to remain focused on the mechanics of the development process. *First Steps* establishes a foundation of procedures, vocabulary, and concepts that we will build on in the next lessons.

Testing in Visual Basic

You can test the object by running the visual basic program *TestExamples* available to you in the Test/VB subdirectory. The program assumes that you have not modified the component project and class names, and that you have registered the component by activating **Generate Database Component**. If you have modified the project and class names, you will need to edit the test program.

Testing in Active Server pages

A directory consisting of ASP/HTML pages exists in the Test/ASP subdirectory. You can use those pages to test the objects you have created. When you generate the objects, make sure that the ASP flag is on. When you begin the testing process, register the directory in the web server as an active directory. (See *Registering Web Pages* on page 45). Once registered, use your web browser to view the site.

Walking

We now know enough to begin developing more complicated components. In this section we reinforce what we learned in *First Steps* by creating a set of services within a component that interacts with a database. In order to provide more time for new concepts that are introduced in this section, we will move quickly through the simpler processes covered in the previous lesson.

The Database Access Component created in this section interacts with a Contact database. The Contact database contains names and addresses (Table 1). The services in the Database Access Component we build allow us to add an address, delete an existing address, view the addresses in the database, or edit an existing address. This portion of the tutorial is divided into four lessons. Lesson 2 describes how to add an address to the database. Lesson 3 explains how to remove an address from the database. Lesson 4 shows how to view the addresses in the database and introduces the concept of working with tables. Lastly, Lesson 5 explains how to update an address in the database.

Overview of the Contact Database

We will use the Microsoft Access database Contact.mdb included in the ActiveBlueprints Tutorial subdirectory (the default directory location is *C:\Program Files\Origins Software\Tutorial*). The Contact database contains a single table called Addresses. Table 1 contains a description of each field in the Addresses table.

Field Name (🔑Primary Key)	**Description**
🔑FirstName	First name of contact
🔑LastName	Last name of contact
Address	Street address of contact
City	City of contact
State	State of contact
Zip	Zip code of contact

Table 1: Field Definitions in the Contact Database

The fields FirstName and LastName are the primary keys for the table. The table uses the primary keys to ensure the uniqueness of its records. The Addresses table does not allow two records (addresses) with the same FirstName and LastName fields. This limitation is an artifact of how the table was designed. It is definitely plausible and perhaps even desirable for the table to have more than one address associated with each contact (a home and a business address, for example). We could have also included other fields, such as the contact's fax number, telephone number, e-mail address, or web site. Additional features and fields add to the complexity of the database without contributing to our understanding of the component development process used by ActiveBlueprints. Since we want to focus on the development process, we have purposely kept the database simple.

Lesson 2: Add a Record to a Database

Objective: *Create a component service that adds a new address to the Contact database.*

Concepts: *Add Service, Field Selector, Single Service Transaction Wizard*

Resource: *Contact.mdb*

With the TutorialCompLib.abc Component Library open in ActiveBlueprints EasyObjects, create a new generated component:

🖱 Name: `Contact`
The default database location is:
[*ActiveBp Install Dir*]\Tutorial\Databases\Contact.mdb

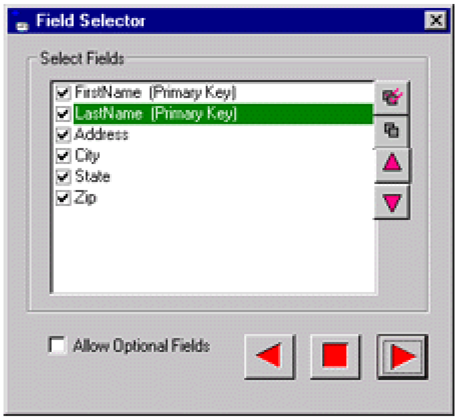
⌨ Description: `Addresses Database Access Component`

Then create a generated Add DAC Service:

A. Create a generated DAC Service in the Contact v1.0 Component that adds an address to the Contact database.

1. Select **Components | Generated Services** and set the following items in the [**Generated…Service Control Panel**] dialog box:
   - 🖱 Table: `Addresses`
   - 🖱 Class: `Add`
   - ⌨ Name: `AddAddress`
   - ⌨ Description: `Add an address record to the Addresses table`
2. Click <Create> to advance the Wizard to the Field Selector.

B. Use the Field Selector to set the fields you want to include when you add a record to a table. Since we want to enter information for every field of the record, all fields should be checked. To select all the fields at once, press the ☑ button (Figure 10).



Field Selector

Select Fields

☑ FirstName (Primary Key)
☑ LastName (Primary Key)
☑ Address
☑ City
☑ State
☑ Zip

□ Allow Optional Fields

Figure 10: Field Selector Dialog Box

C. Advance the Wizard until the 🚩 button appears. Press the 🚩 button to create the AddAddress DAC Service.

D. Click <Exit> to close the [**Generated…Service Control Panel**] dialog box.

E. Select **Components | Generate Database Component** to generate and compile the source code for the Contact v1.0 Component.

Lesson 3: Delete a Record in a Database

Objective: *Create a component service that deletes an address from the Contact database.*

Concepts: *Delete Service, Criteria Editor*

Resource: *Contact.mdb*

With the TutorialCompLib.abc Component Library open in ActiveBlueprints EasyObjects, create a generated Delete DAC Service:

A. Create a generated DAC Service in the Contact v1.0 Component that, given the name of the contact, deletes an address in the Contact database.

1. Select **Components | Generated Services** and set the following items in the [**Generated…Service Control Panel**] dialog box:
   - 🖱 Table: `Addresses`
   - 🖱 Class: `Delete`
   - ⌨ Name: `DeleteAddress`
   - ⌨ Description: `Delete an address record in the Addresses table`
2. Click <Create> to advance the Wizard to the Criteria Editor.

B. Use the Criteria Editor to determine which record to delete from the database (Figure 11). Since the primary keys ensure the uniqueness of the records in the Addresses table, you should base the criteria for which record to delete on the primary keys. In this case, we want to delete a record based on the first name and last name of a contact. To set up these criteria:

1. Double-click the FirstName field in the <Field Names> list box.
2. Make sure that the "=" operator is selected in the <Equality> list box.
3. Click <Add> in the yellow toolbar. The criterion FirstName = FirstName appears in the <Expression Viewer> box. The left hand side of the equality operator is the Field Name. The right hand side is the Criteria Parameter. Interpret the expression as delete the record where the database field FirstName in the table Addresses is equal to the parameter FirstName.

   Use a Criteria Parameter as a variable in a Field Expression to create a dynamic search. A dynamic search queries a database with a search condition that varies from search to search. In the first lesson, we used a constant string instead of a Criteria Parameter in the Field Expression. Since the string was constant, the search created in the previous lesson was a static one.

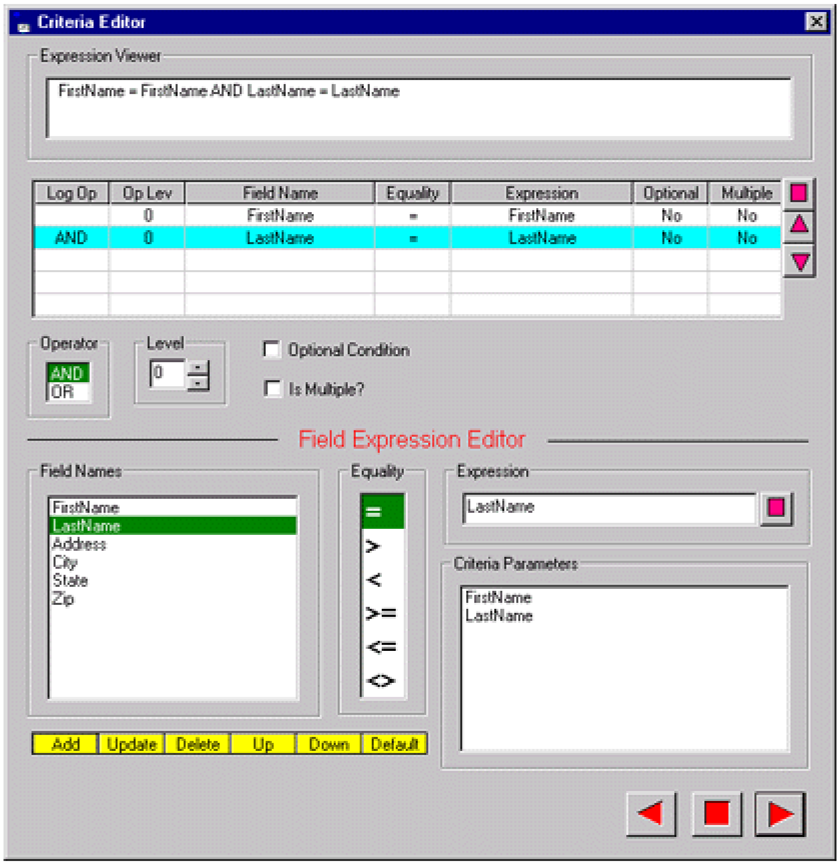
4. Since the field FirstName by itself does not uniquely define a record, we also need to have the field LastName as one of the criteria. Double-click the LastName field in the <Field Names> list box.

5. With the "=" operator still selected in the <Equality> list box, click <Add> in the yellow toolbar to add the second criterion. Now the Criteria Editor contains two Criteria Parameters (Figure 11).

6. Advance the Wizard until the 🏁 (Finish) button appears. Press the 🏁 button to create the DeleteAddress Service.

C. Click <Exit> to close the [**Generated…Service Control Panel**] dialog box.

E. Select **Components | Generate Database Component** to generate and compile the source code for the Contact v1.0 Component.



FirstName = FirstName AND LastName = LastName

Log Op	Op Lev	Field Name	Equality	Expression	Optional	Multiple
	0	FirstName	=	FirstName	No	No
AND	0	LastName	=	LastName	No	No

Level: 0

Field Names: FirstName, LastName, Address, City, State, Zip

Expression: LastName

Criteria Parameters:
FirstName
LastName

Figure 11: Criteria Editor for Lesson 3

Lesson 4: List All the Records in a Database

Objective: *Continuing with Contact database, add the ability to view all of the address book records.*

Concepts: *Search Service, Recordset, Table*

Resource: *Contact.mdb*

With the TutorialCompLib.abc Component Library open in ActiveBlueprints EasyObjects, create a generated Search DAC Service:

A. Create a generated DAC Service in the Contact v1.0 Component that retrieves all of the addresses in the Contact database and returns the result in a table.

1. Select **Components | Generated Services** and set the following items in the [**Generated…Service Control Panel**] dialog box:
   - 🖱 Table: `Addresses`
   - 🖱 Class: `Search`
   - ⌨ Name: `ViewAddresses`
   - ⌨ Description: `View all addresses in the Contact database`
2. Click <Create> to advance the Wizard to the Criteria Editor.

B. Since we are interested in searching for *all* records in the Addresses table, we do not have to enter a specific search criterion in the Criteria Editor. Press the ▶ button to advance the Wizard to the Display Editor.

C. Use the Display Editor to define which fields of a record are returned from a search, the sorting order of the records, and the kind of object that displays the records found. We want to set the Display Editor to return the contents of all fields in a table. We also want to sort the records alphabetically by last name, and then, if necessary, alphabetically by first name.

1. To return all fields from a record, press the ⇉ button in the <Field Names> box. All fields in the <Field Names> box move over to the <Display> box.
2. To sort the output by last name and then by first name:

   a) Select the LastName field in the <Display> list box and press the ⇅ (Sort Field) button.

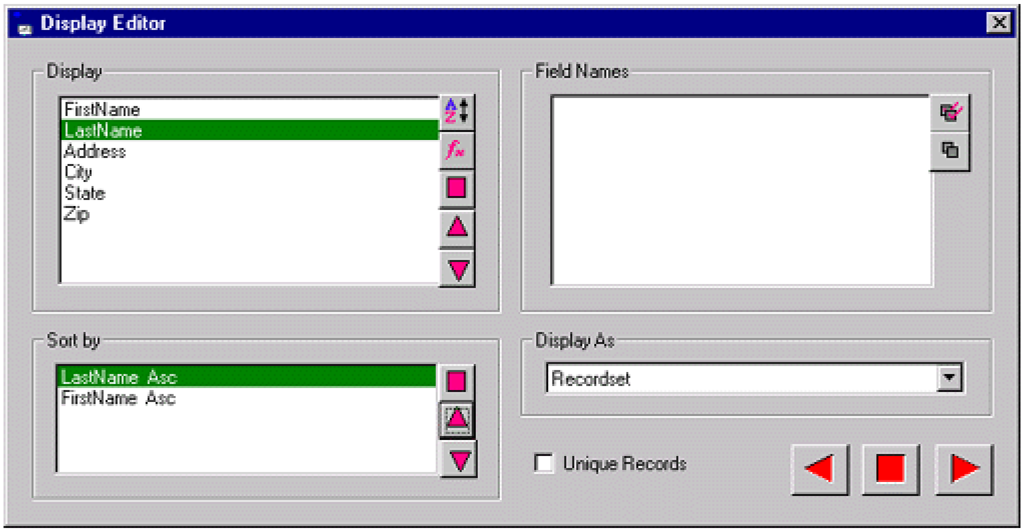
   b) Select the FirstName field in the <Display> list box and press the ⇅ button.

3. To display returned records in a table, select the following in the <Display As> box:

   🖱 Display As: `Recordset`

   A Recordset is a data object used to hold records from a database in a table.

After you execute these steps, the Display Editor appears as it does in Figure 12.



Display:
- FirstName
- LastName
- Address
- City
- State
- Zip

Sort by:
- LastName Asc
- FirstName Asc

Display As: Recordset

□ Unique Records

Figure 12: Display Editor Settings for Lesson 4

D. Advance the Wizard until the button appears. Press the button to create the ViewAddresses DAC Service.

E. Click <Exit> to close the [**Generated…Service Control Panel**] dialog box.

F. Select **Components | Generate Database Component** to generate and compile the source code for the Contact v1.0 Component.

Lesson 5: Update a Record in a Database

Objective: *Using the Contact database, add the ability to update selected address book records.*

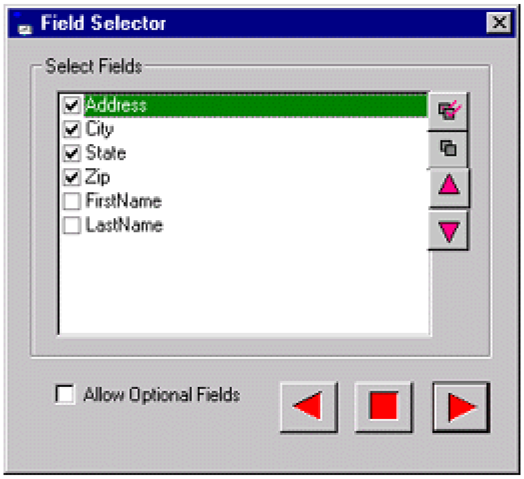
Concepts: *Update Service*

Resource: *Contact.mdb*

With the TutorialCompLib.abc Component Library open in ActiveBlueprints EasyObjects, create a generated Update DAC Service:

A. Create a generated DAC Service in the Contact v1.0 Component that, given the first and last names of a contact, updates an address in the Contact database.

1. Select **Components | Generated Services** and set the following items in the [**Generated…Service Control Panel**] dialog box:
   - 🖱 Table: `Addresses`
   - 🖱 Class: `Update`
   - ⌨ Name: `EditAddress`
   - ⌨ Description: `Edit an address in the Contact database`
2. Click <Create> to advance the Wizard to the Field Selector.



Field Selector

Select Fields

☑ Address
☑ City
☑ State
☑ Zip
☐ FirstName
☐ LastName

☐ Allow Optional Fields

Figure 13: Field Selector Settings for Lesson 5

B. Use the Field Selector to select which fields of a record are updated. For simplicity, assume that the primary keys never change. Therefore, select every field except for FirstName and LastName (Figure 13).
🖱 Select Fields: `Address, City, State, Zip`

The selected fields move to the top of the list.

C. Advance the Wizard to the Criteria Editor and create the same criteria you created for Lesson 3: Delete a Record in a Database (Figure 11 on page 17):

```
[FirstName] = FirstName AND [LastName] = LastName
```

D. Advance the Wizard until the ▣ button appears. Press the ▣ button to create the EditAddress DAC Service.

E. Click <Exit> to close the [**Generated…Service Control Panel**] dialog box.

F. Select **Components | Generate Database Component** to generate and compile the source code for the Contact v1.0 Component.

Walking Summary

The primary focus of *Walking* is database connectivity. We created several Services for a Database Access Component. The Services enable us to add, delete, search, and update records in a database. Along the way, we learned how to use the development environment of ActiveBlueprints EasyObjects to create objects in Visual Basic that are both functional and versatile.

Chapter 3: User's Guide

Components are the building blocks of applications. A component is a reusable, interoperable software module composed of one or more objects with a well-defined interface like ActiveX, COM, DCOM, or CORBA. Components can vary in size and complexity. They can be small and simple like a label used in a form or large and complex like the spreadsheet module used in Microsoft Excel. Furthermore, a component can be made up of other components. Because the term component is so generic, it is meaningless without some context.

Components in ActiveBlueprints

In the context of ActiveBlueprints, **Components** are interoperable ActiveX or COM software modules that contain Services. A **Service** is the active part of a Component that performs some function. Since Services can be used independently within a Component, Components can be thought of as a way to organize Services. Services are themselves composed of **Service Elements**, which are objects that input information to or output information from a Service.

ActiveBlueprints can use any ActiveX or COM component that does not need to be placed in a form. These components are known as back-end components because they do not necessarily have a front-end or user interface. The text editor in Microsoft Word and the graph viewer in Microsoft Graph are examples of back-end components.

Before using a Component in ActiveBlueprints, it must be registered to a Component Library using the program ActiveBlueprints EasyObjects. ActiveBlueprints does not distinguish among the types of Components registered in the Component Library. It does, however, differentiate components into two categories based on how they are created. **Standard Components** are created by a developer or purchased from a third party, whereas **Generated Database Access Components (GDAC)** are built using the ActiveBlueprints EasyObjects Service Wizard. Both types of Components are registered to the Component Library: Standard Components are registered manually while GDACs are registered automatically by the Service Wizard. *Register a Standard* Component on page 25 describes the process of registering a Standard Component to a Component Library. *Creating Generated Database Access Components* on page 30 shows how to create a Component with Services using the Service Wizard.

ActiveBlueprints EasyObjects

ActiveBlueprints EasyObjects is a Windows application used to create and maintain Component Libraries for ActiveBlueprints (Figure 14). The program is located in the same directory as ActiveBlueprints under the filename ActiveBPComponents.exe.

The tree view located in the upper left portion of the ActiveBlueprints EasyObjects main window is the **Component Explorer**. The Component Explorer contains a list of all the Components in the Component Library. The tree view in the lower left portion of the ActiveBlueprints main window, under the Component Explorer, is the **Service Explorer**. The Service Explorer contains a list of the Services and Service Elements in the selected Component in the Component Explorer. Selecting a Component in the Component Explorer updates Services and Service Elements displayed in the Service Explorer. The name of the selected Component appears in the border area of the Service Explorer.

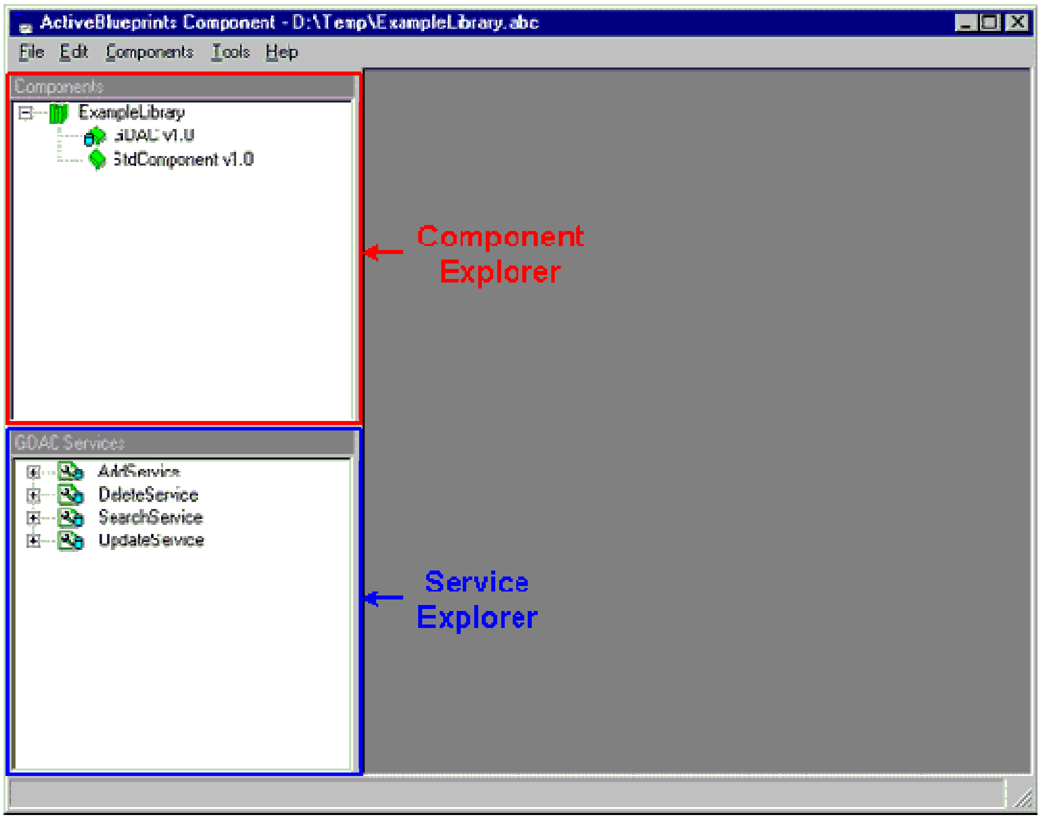


Figure 14: ActiveBlueprints EasyObjects Development Environment

Selecting any item in either Explorer and choosing the **Components** option in the main menu bar opens a submenu unique to the type of item selected. Right clicking on the mouse in either Explorer while an item is selected also provides the same submenus. Since right clicking to select actions on items in the Explorers is equivalent to choosing the **Components** option in the main menu bar, only the main menu bar method is described in the upcoming text to minimize redundancy.

As a further aid to ActiveBlueprints EasyObjects developers, the Component and Service Explorers use specific icons which provide additional information about the Component and Service type, the Service Element type and purpose, and the error status of Generated Database Access Components, Services, and Service Elements. Table 2 in *ActiveBlueprints EasyObjects Icons* on page 43 contains a complete list of the icons used in the Explorers and a description of each symbol.

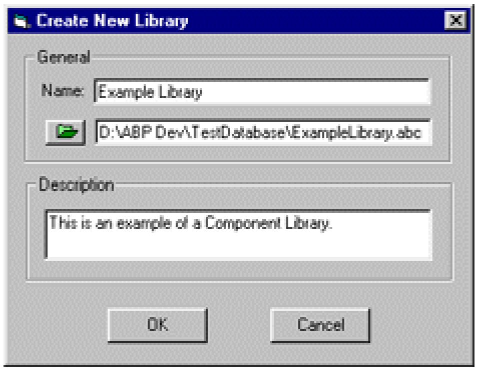
Component Library

ActiveBlueprints organizes components into Component Libraries. A **Component Library** is a database with a file extension of **.abc** (**A**ctive**B**lueprints **C**omponents). For example, the filename of a Component Library might be libraryfile.abc, where libraryfile is the name of the Component Library. The filename libraryfile does not have to be the same as the name of the Component Library. To minimize confusion, though, it is recommended that the name of the database and the database filename be the same.

A Component Library is composed of one or more Components. Each Component within a Component Library must have a unique name and version number. If two Components have the same name, then their version numbers must be different. Each Component consists of one or more Services. No two Services within the same Component may have the same name. Each Service consists of one or more Service Elements. No two Service Elements within a Service may have the same name. Valid names for Components, Services, and Service Elements begin with an uppercase or lowercase letter followed by no more than 19 alphanumeric characters. Spaces are allowed in the names, but no other characters may be used.

Creating a Library

To create a Component Library, run ActiveBlueprints EasyObjects from **Start** | **Programs** | **ActiveBlueprints** and select **File | New Library**. The program brings up the dialog shown in Figure 15. Enter a name for the new Component Library and click the 📂 icon to select a directory and set the filename of the Component Library. Then enter a description for the Component Library in the <Description> edit box. Although the description is not absolutely necessary, it prevents confusion with other libraries. Click <OK> to create a new Component Library.



Name: Example Library

📂 D:\ABP Dev\TestDatabase\ExampleLibrary.abc

This is an example of a Component Library.

Figure 15: Create New Library Dialog Box

General

Name

Enter name of Component Library.

📂

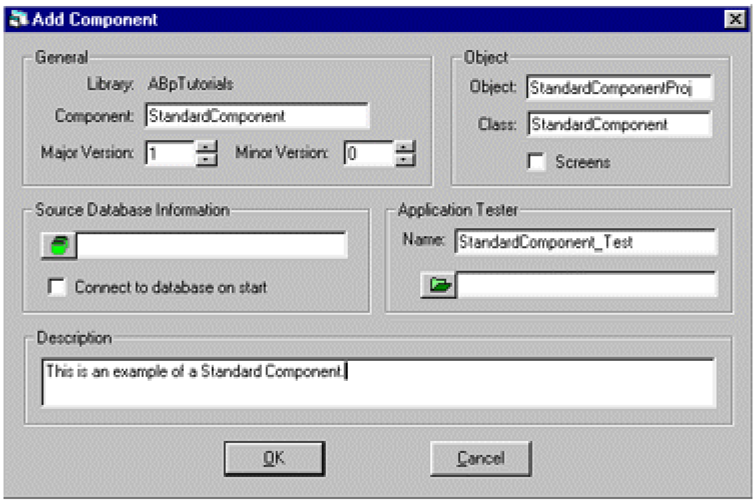
Press button to bring up filename selection dialog. Select a filename for the Component Library.

Description

Provide a description of the Library. The description might include the purpose of the library.

Register a Standard Component

Registering a Standard Component to a Component Library provides ActiveBlueprints with the information that it needs to use the Component's Services. To add a Standard Component to a Component Library, select the Component Library icon (📚) or any of the Component icons (◆,◆) in the Component Explorer and then go to the menu item **Components** | **Add a Component**. Select **Add a Component** to bring up the dialog box in Figure 16. Filling out the mandatory information in this dialog box and pressing <OK> creates a name space for the Component in the Component Library. At this point, the Component does not have any Services; consequently, the Service Explorer is empty after you select the Component in the Component Explorer. To make this Component useful in ActiveBlueprints, Services must be registered to this component.



Library: ABpTutorials

Component: StandardComponent

Major Version: 1

Minor Version: 0

Object: StandardComponentProj

Class: StandardComponent

□ Screens

□ Connect to database on start

Name: StandardComponent_Test

This is an example of a Standard Component.

Figure 16: Register a Standard Component to a Component Library

GENERAL

Library

Name of the Component Library.

Component

Name of component. The component name is the name used within ActiveBlueprints to identify the component (*mandatory*).

Major Version

Major version number (*mandatory*).

Minor Version

Minor version number (*mandatory*).

OBJECT

Object

Name used to identify object for COM (*mandatory*).

Class

Name used to identify class with a given object under COM (*mandatory*).

Screens

Feature will be removed.

Source Database Info

If the component uses a database, select the name of the database for the component to open.

Connect to database on start

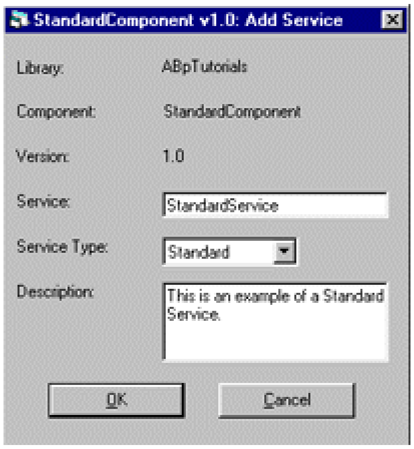
If this option is checked and a database name has been provided, then ActiveBlueprints will connect to the database when the component is initialized.

Application Tester

Feature will be removed.

Register a Service

To register a Service to a Standard Component, select the target Component in the Component Explorer, or any of the registered Services (◆) of the target Component in the Service Explorer. Then select the menu item **Components** | **Add a Service**. The dialog box in Figure 17 appears and prompts you for information needed to register a Service to a Component.



StandardComponent v1.0: Add Service

Library: ABpTutorials

Component: StandardComponent

Version: 1.0

Service: StandardService

Service Type: Standard

Description: This is an example of a Standard Service.

Figure 17: Register a Service to a Standard Component

Library

Name of the Component Library

Component

Name of the target Component (Component where Service will be registered).

Version

Version number of the target component.

Service

Name of service that will be used to call the service through COM. The name should be the exact name used by COM.

Service Type

Type of service.

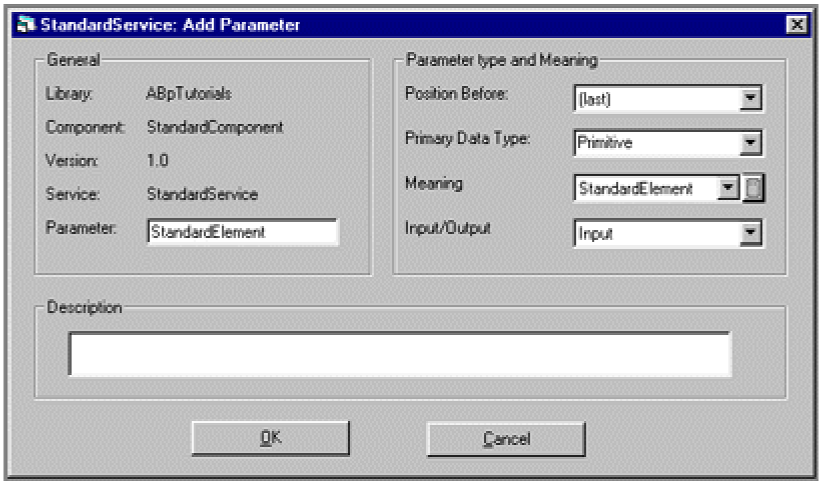
- **Standard**: A Service with an integer return value and a string return message as the first Service Element followed by zero or more Service Elements of any arbitrary data type.
- **Function**: A Service with an arbitrary data type return value and zero or more Service Elements of any arbitrary data type.
- **Subroutine**: A Service with no return value and zero or more Service Elements of any arbitrary data type.

Description

Description of Service (include the purpose of the Service).

Register a Service Element

A **Service Element** is an object used to send information to or receive information from a Service. To register a Service Element to a Service in a Standard Component, select the target Service (🔧) or one of the target Service Elements (➔🟩,⬅🟩 ,⇅🟩) in the Service Explorer and then select the menu item **Components** | **Add a Parameter**. The dialog box in Figure 18 comes up and prompts you for information needed to register a service element to a service.



StandardService: Add Parameter

General

Library: ABpTutorials

Component: StandardComponent

Version: 1.0

Service: StandardService

Parameter: StandardElement

Parameter type and Meaning

Position Before: (last)

Primary Data Type: Primitive

Meaning: StandardElement

Input/Output: Input

Description

Figure 18: Register a Service Element to a Service

GENERAL

Library

Name of the Component Library.

Component

Name of the target Component (Component where Service is registered).

Version

Version number of the target Component.

Service

Name of the target Service (Service where Service Element will be registered).

Parameter (Service Element)

Name of the registered Service Element.

PARAMETER TYPE & MEANING

Position Before

Position of the element being registered. Element being registered will appear before the element listed. If you select [last], the element being registered appears last in the Service Element list.

Primary Data Type

Primary data type of the element.

- **Primitive**: Basic data type like Byte, Integer, Long, Single, Double, String.
- **Collection**: Microsoft Visual BASIC list data structure.
- **Origins Selection List**: Origins Software Company list data structure.
- **Workspace**: Microsoft DAO data structure for representing a database workspace.
- **Database**: Microsoft DAO data structure for representing a database.
- **Recordset**: Microsoft DAO data structure for representing tables and queries from a database.
- **Origins Grid**: Origins Software Company array data structure.
- **Array**: Microsoft Visual BASIC array data structure.

Meaning

Meaning of the element.

Input/Output (Purpose)

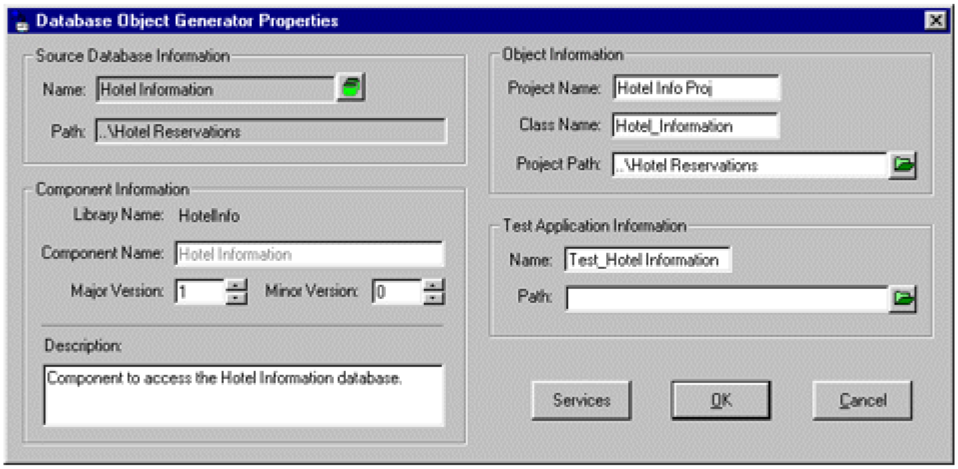
How the element will be used.

- **Input**: Use for entering information into a service.
- **Output**: Use for retrieving information from a service.
- **Both**: Use for entering information into and retrieving information from a service.

Creating Generated Database Access Components

Once the Component Library has been created, ActiveBlueprints EasyObjects is now ready to create a new Generated Database Access Component (GDAC). To create a new GDAC, select **Components** | **Create Database Component** from the main menu bar. An empty [**Generated Access Database Component Properties**] dialog box appears. Fill out the mandatory fields in the dialog box (Figure 19).

After all of the mandatory information is provided to the [**Generated Access Database Component Properties**] dialog box, click <OK> or <Services> to create a Component. Please note that although this action creates a Component, it contains no Services and therefore is of little use in ActiveBlueprints.


Database Object Generator Properties
Source Database Information
Component Information
Object Information
Test Application Information
Services
OK
Cancel

Name: Hotel Information

Path: ..\Hotel Reservations

Library Name: HotelInfo

Component Name: Hotel Information

Major Version: 1

Minor Version: 0

Description: Component to access the Hotel Information database.

Project Name: Hotel Info Proj

Class Name: Hotel_Information

Project Path: ..\Hotel Reservations

Name: Test_Hotel Information

Path:

Figure 19: Add a Generated Database Access Component

Source Database Information

Name

Press 🟩 to bring up a dialog box to select the database for which the Services are to be created.

Path

Path of the selected database.

Component Information

Library Name

Name of the Component Library.

Component Name

Name of the Component. This will be the name referred to in ActiveBlueprints.

Major Version

Major version number of the Component.

Minor Version

Minor version number of the Component.

Description

Description of the Component.

OBJECT INFORMATION

Project Name

Name used to identify the object for COM (*mandatory*). This is also name of the project file for generated code. Must be less than 15 characters.

Class Name

Name used to identify the class for COM (*mandatory*). This is also the name of the class file for generated code. Must be less than 15 characters.

Project Path

Press 📂 to bring up a directory selection dialog box. Select directory where the generated project will be created.

TEST APPLICATION INFORMATION

Feature will be removed.

SERVICES (button)

Creates a Generated Database Access Component, registers it to a Component Library, then brings up the GDAC Service Control Panel.

GDAC Service Wizard

The GDAC Service Wizard consists of the [**Service Control Panel**], [**Field Selector**], [**Criteria Editor**], [**Display Editor**], and [**Service Properties**] dialog boxes. The [**Service Control Panel**] appears at the beginning and at the end of the GDAC Service Wizard. From the [**Service Control Panel**], a GDAC Service can be created, edited, or deleted. To launch the GDAC Service Control Panel, select the target GDAC () in the Component Explorer and select **Components** | **Add a Generated Service** from the main menu or double-click any of the existing target GDAC's Services. The dialog box in Figure 20 appears.

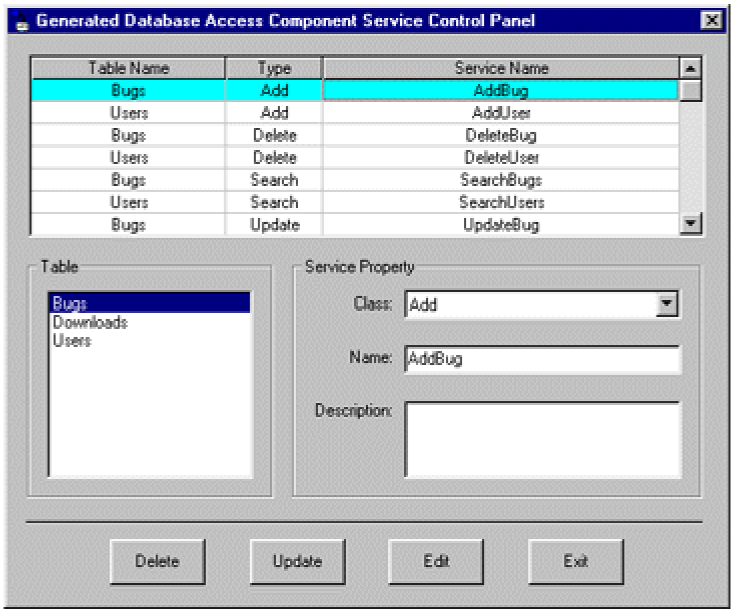


Table Name	Type	Service Name
Bugs	Add	AddBug
Users	Add	AddUser
Bugs	Delete	DeleteBug
Users	Delete	DeleteUser
Bugs	Search	SearchBugs
Users	Search	SearchUsers
Bugs	Update	UpdateBug

Table:
Bugs
Downloads
Users

Class: Add

Name: AddBug

Description:

Figure 20: Generated Database Access Component Service Control Panel

TABLE (AND QUERIES)

List box of all tables and queries in the source database.

SERVICE PROPERTY

Class

Select the GDAC service class to create. Once a Service is defined, the service class cannot be changed.

- **Add**: Adds a record to the selected table.
- **Delete**: Deletes one or more records from selected table.
- **Update**: Updates a record from a table.
- **Search**: Searches for records in a table.

Name

Unique name of the Service.

Description

Description of the purpose of the Service.

DELETE (button)

Deletes highlighted Service from GDAC.

UPDATE (button)

Updates the name or description of the highlighted Service. No other property may be changed. To change the Service’s Table/Query or service class, delete the Service and build it again with the appropriate table and class.

CREATE/EDIT (button)

If the value in the <Name> edit box does not already exist, a <Create> button is enabled. Otherwise, an <Edit > button is enabled. In either case, pressing the button brings up the Service Wizard's dialog boxes. The dialog boxes that appear depend on the class of the Service you are creating or editing.

Table 3 on page 44 shows a list of the navigation controls for the Service Wizard.

The GDAC Service Wizard creates Services that add, delete, search, or update records for a single table in a database. All four services can be generated for any user defined table, but only the search Service is available for a query. All generated services act on a single table or query.

Add Service

An Add service is a service that adds a new record to a table in a database. The Add service uses the [**Field Selector**] (see Figure 21 on page 34) to determine which fields are needed for a record to be added to the table. The [**Field Selector**] indicates which fields are primary and required for the selected table.

The **Primary Key** and the **Required Key** fields are mandatory for adding a new record. Consequently, these fields are always checked in the Field Selection dialog box. If the <Allow Optional Fields> checkbox is selected, then the non-Primary Key and non-Required Key fields can be filled optionally when adding a new record. If the <Return Auto Number> checkbox is selected, then the Add service will return the automatically generated, unique identification number from the table. An Add service will return a 0 if it has succeeded in adding an additional record to the table and a negative number if an error has occurred.

Delete Service

A Delete service is a service that removes from a table one or more records matching the criterion defined using the [**Criteria Editor**] dialog box. If a criterion is not provided, then all records in the table are deleted. A Delete service returns the number of records deleted or a negative number if an error occurred.

Update Service

An Update service updates one or more fields of a record matching the criterion defined using the [**Criteria Editor**]. Use the [**Field Selector**] dialog box to set which fields from the table are updated. It is inadvisable to update the contents of the Primary Key field. An Update service returns a 1 if a record was successfully updated, a 0 if no record is found, and a negative number if an error occurred.

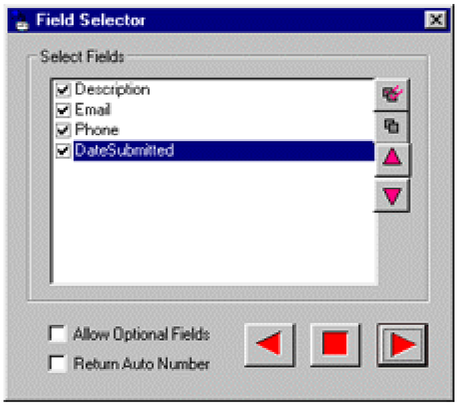
Search Service

A Search service finds one or more records matching the criterion defined using the [**Criteria Editor**]. Use the [**Display Editor**] to format the output of the records found. The Search service can return the search results in a table (Recordset), a list (Collection), an item, a collated string item (String Item), or a list of collated string (String Collection). A collated string is a string with all of the fields in a record combined and separated by a separator defined at design time. Because an item and a collated string item display just one record at a time, only the first record is returned if multiple records are found in the search.

The results of a Search service can be sorted by ascending or descending order. The <Sort By> list box found in the [**Display Editor**] allows the developer to sort the search results by any returned field. The Search service returns the number of records found or a negative number if an error occurred.

Field Selector

The [**Field Selector**] dialog shown in Figure 21 is used in Add and Update services. In Add services, the [**Field Selector**] allows the developer to choose fields from the selected table in the [**Service Control Panel**] to include when adding a new record. The fields marked as Primary or Required in the <Select Fields> list box are mandatory. In Update services, the [**Field Selector**] allows the developer to select the fields to be updated. A field designated with the Auto Number data type cannot be updated and is not available in the <Select Fields> list box.



☑ Description
☑ Email
☑ Phone
☑ DateSubmitted

□ Allow Optional Fields
□ Return Auto Number

Figure 21: GDAC Field Selection Dialog Box

Select Fields

List box containing all relevant fields from the table selected in the Service Control Panel. For Add Services, all fields from the table with the exception of Auto Number data type fields are displayed. For Update Services, all fields with the exception of the Auto Number field and the fields that are part of the primary key are displayed.

Allow Optional Fields

If selected, the non-mandatory fields checked in <Select Fields> have Service Elements that are optional to the Service. Optional Service Elements are they are not essential for the Service to add a record to a table. The selection of this option also causes **mandatory fields** (fields that are part of the primary key or required by the database) to be automatically moved above any non-mandatory field.

Return Auto Number

This option is only available for Add Services on tables that have an Auto Number field. If selected, the Service returns the Auto Number value that is generated by the database when a new record is added to a table.

Selects all fields and moves all mandatory fields above the non-mandatory ones.

Deselects all non-mandatory fields.

▲

Move selected item up. If <Allow Optional Fields> is checked, fields that are non-mandatory cannot be positioned above mandatory ones.

▼

Move selected item down. If <Allow Optional Fields> is checked, fields that are mandatory cannot be positioned below fields that are not mandatory.

Criteria Editor

Delete, Search, and Update Services use the [**Criteria Editor**] in Figure 22 on page 37 to define the search criteria for retrieving records. Each Service uses the records retrieved in different ways. The Delete Service removes the records from a table; the Update Service finds specific records and changes the contents of fields selected in the [**Field Selector**]; and the Search Service returns the records found in a format defined in the [**Display Editor**].

Use the [**Criteria Editor**] to construct a series of field expressions joined by logical operators. A **Field Expression** consists of a field name from the <Field Name> list box, a comparison operator from the <Equality> list box, and an expression composed of constants and Criteria Parameters combined with the proper mathematical or string operators in the <Expression> edit box. The field names come from the table selected in the [**Service Control Panel**]. The comparison operator can be =, >, <, >=, <=, or <>. **Criteria Parameters** are user variables in the <Criteria Parameters> list box. The following are examples of Field Expressions, where Field1 and Field2 are field names from the <Field Name> list box and Param1 and Param2 are Criteria Parameters. Furthermore, let Field1 be a string and Field2 be a date:

Valid Examples

Field1 = “Hello”

Field1 = “Hello” & Param1

Field2 = #10/6/71# - 3

Field2 > Param2

Field1 = Param1 & Param2

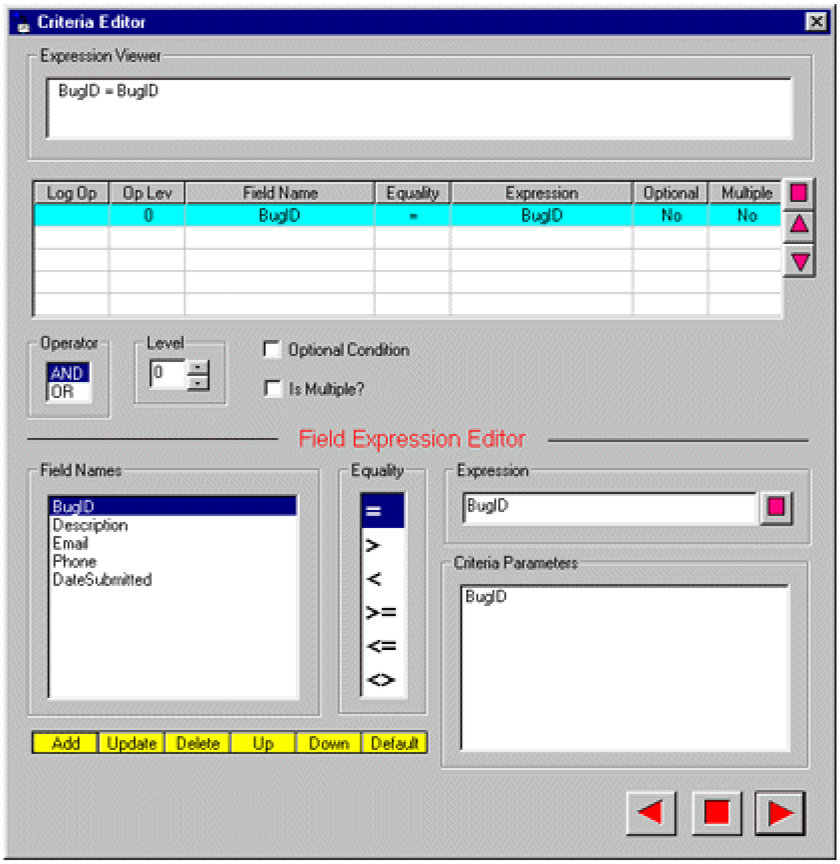
Invalid Example

Field1 = Param1 + Param2

(This example is invalid because Param1 is defined a string. A string added to an integer is meaningless and an illegal expression.)

To define a Criteria Parameter, right-click in the <Criteria Parameters> list box, enter a name for a parameter next to the <Add> edit box, and then press the Enter/Return key twice while the <Add> edit box is still in focus. When a Criteria Parameter is first specified, its data type is undefined, so it can be used with any field name in a Field Expression. Once you have used a Criteria Parameter a field in a Field Expression that has been added to the Logical Condition table, its data type is bound to the data type of that field and it can only be further used with similarly typed fields. For example, if Param1 is a Criteria Parameter used in an expression with Field1, a date, then Param1 can only be used afterwards with other fields that are also dates.

To create a Field Expression, select a field name from the <Field Names> list box, select a comparison operator from the <Equality> list box, and enter an expression in the <Expression> edit box. Then choose a logical operator from the <Operator> list box and the expression level from the <Level> list box. The expression level is the evaluation order of a Field Expression. Field Expressions with high levels are evaluated first. After you have defined a Field Expression, select <Add> in the yellow toolbar at the bottom left-hand corner of the [**Criteria Editor**]. The Field Expression is added to the Logical Condition table and the <Expression Viewer> is updated.



Expression Viewer

BugID = BugID

Log Op	Op Lev	Field Name	Equality	Expression	Optional	Multiple
	0	BugID	=	BugID	No	No

Operator: AND / OR

Level: 0

□ Optional Condition

□ Is Multiple?

Field Expression Editor

Field Names: BugID, Description, Email, Phone, DateSubmitted

Equality: = > < >= <= <>

Expression: BugID

Criteria Parameters: BugID

Figure 22: GDAC Criteria Editor

Expression Viewer

The read only text box at the top of the display that contains the search criteria expression.

Logical Condition Table

Table under the Expression Viewer that lists the individual logical expressions for the search criteria.

■

Delete selected logical expression from criteria.

▲

Move selected logical expression up.

▼

Move selected logical expression down.

Operator

Logical operator to link criteria expressions.

- **AND**: Intersection of associated criteria
- **OR**: Union of associated criteria

Level

Parentheses level for a criteria expression. 0 equals no parentheses around the expression, whereas 9 (maximum level) indicates the expression is nested 9 levels deep.

Optional Condition

Not implemented yet. If checked, allows criteria to be optional.

Field Names

Names of all fields in the selected table.

Equality

Equality and inequality operators used to compare selected Field Names with selected Criteria Parameters.

EXPRESSION

A logical expression that uses Field Names, Criteria Parameters, constants, and mathematical and string operators to select specified records from a database.

■

Clears the contents of the <Expression> edit box.

CRITERIA PARAMETER

A Service Element introduced by the user to represent a variable in an equation. To display the popup menu for manipulating criteria parameters shown in Figure 23, right-click the Criteria Parameters list box. The order of the parameters in the Criteria Parameters list box determines how the parameters are ordered under the Service in the Service Explorer.

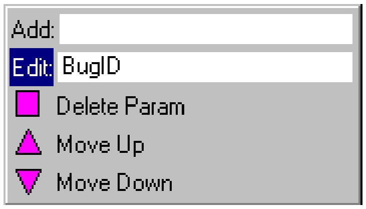


Figure 23: Criteria Parameter Popup Menu

CRITERIA EDITOR TOOL BAR (yellow bar under the <FIELD NAMES> list box)

Add

Adds a Logical Condition to the Logical Condition table. To enter simple conditions like FieldName = ParamName, double-click the field name and then click <Add>.

Update

Updates the characteristics of the selected Logical Condition in the Logical Condition table.

Delete

Delete the selected logical expression from the Logical Condition table.

Up

Move the selected logical expression up in the Logical Condition table.

Down

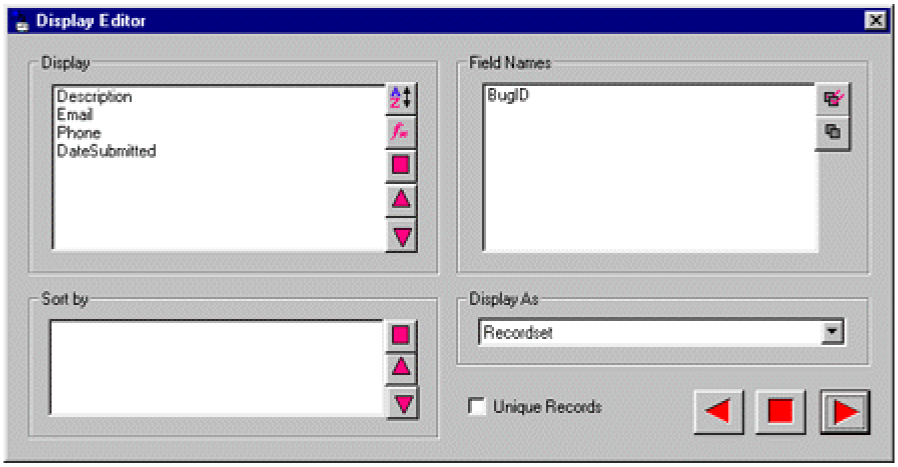
Move the selected logical expression down in the Logical Condition table.

Default

Provides a one-to-one mapping of all fields joined together with an AND statement.

Display Editor

Click the ▶ (Next) button in the lower right-hand corner of the [**Criteria Editor**] to bring up the [**Display Editor**] shown in Figure 24. The [**Display Editor**] is used only when you want to create a Search service. The Display Editor determines what fields are returned by a search as well as the format and order of the fields.



Display: Description, Email, Phone, DateSubmitted

Field Names: BugID

Display As: Recordset

☐ Unique Records

Figure 24: GDAC Display Editor Dialog Box

Display

Lists the fields from the records found in a search that will be displayed in the output of the search. The order of the fields in the list box determines their order when displayed. The object that displays the fields is determined by the <Display As> property.

AZ↕

Press to sort on the selected field.

fx

Not implemented yet. Provides functions support for fields.

■

Remove selected field from the display.

▲

Move selected field up.

▼

Move selected field down.

Field Names

Lists all fields from the selected table or query that are not included in the <Display> list box.

⧉✓

Display all fields in the table.

⧉

Remove all fields from the display.

Sort By

Determines how the results of the search should be sorted. Only fields that are in the <Display> list box can be sorted. To choose a field to sort by, select a field in the <Display> list box and press the sort button next to the <Display> list box. The field name appears in the <Sort by> list box. To change the sort order of a field, double-click the field. If the field was sorted by ascending order (Asc), it changes to descending order (Desc). If the field was originally sorted in descending order, the order changes to ascending. To remove a sort condition and the field associated with it, select the field to delete and press the delete button next to the <Sort by> list box. The order of the sort conditions in the <Sort by> list box determines the order of the sorted records in the database.

■

Delete selected sort condition for specified field.

▲

Move sort condition up.

▼

Move sort condition down.

DISPLAY AS

The data carrier type for Service Elements used to display the results of a search.

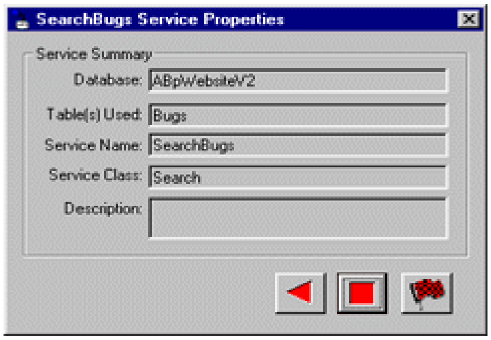
- **Recordset**: Table with the fields as columns and the records as rows.
- **Collection**: Fields are broken into lists, one list for each field.
- **Item**: Fields are broken into items, one item for each field. For searches that return multiple records, only the first record is returned as the item.
- **String Item**: String with the fields concantenated. For searches that return multiple records, only the first record is returned.
- **String Collection**: List of strings with the fields concantenated. Each item in the list represents a record that was retrieved.

UNIQUE RECORDS

For searches that yield duplicate records, checking <Unique Records> will prevent duplicate records from being returned by the search.

Service Properties

The GDAC Creation Wizard displays the Service Properties dialog box in Figure 25 to summarize information about the service you have created. Until you click the <Finish> button, the changes made to the services are not committed to the Component Library. To review the specifics of the service, just press the <Previous> button, but be careful not to back up too far or the changes that you made to the service will be lost.



SearchBugs Service Properties

Service Summary

Database: ABpWebsiteV2

Table(s) Used: Bugs

Service Name: SearchBugs

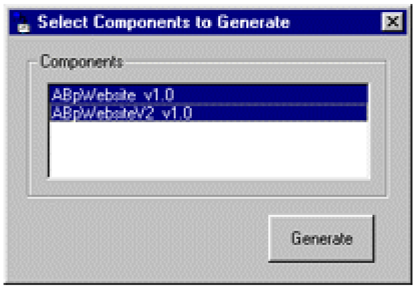
Service Class: Search

Description:

Figure 25: GDAC Service Properties Dialog Box

Generating Code for a GDAC

ActiveBlueprints EasyObjects provides a single Component and a multiple Component method for generating and compiling code for a GDAC. The single Component method is used to generate code for a single GDAC. To use the single Component method, select the GDAC in the Component Explorer for which code is to be generated and then select **Components | Generate Database Component** in the main menu. To generate code for multiple Components, select the Component Library icon (🟩) and then select **Components | Generate Database Component** in the main menu. The [**Component Selector**] dialog box in Figure 26 appears with a list of all of the GDACs in the Component Library selected. ActiveBlueprints will generate code for the selected GDACs. To remove or add GDACs to the list, hold down the control key and click on the GDAC to remove or add. To begin generating code, press the <Generate> button.



ABpWebsite v1.0
ABpWebsiteV2 v1.0

Figure 26: Select Components to Generate Dialog Box

If ActiveBlueprints EasyObjects detects that you have modified the GDAC's source code file, it prompts you with the [**Preserve Selected Services During Regeneration**] dialog box in Figure 27. The dialog lists all of the possibly modified or unregistered Services found in the GDAC's source code. If a Service is no longer needed, move it from the <Preserve> list box to the <Discard> list box. The <Preserve> list box contains a list of all of the Services in the source code that ActiveBlueprints EasyObjects will not overwrite when it is regenerating code for the GDAC. Likewise, the <Discard> list box contains a list of the Services that the code generator will remove while regenerating code for the GDAC.

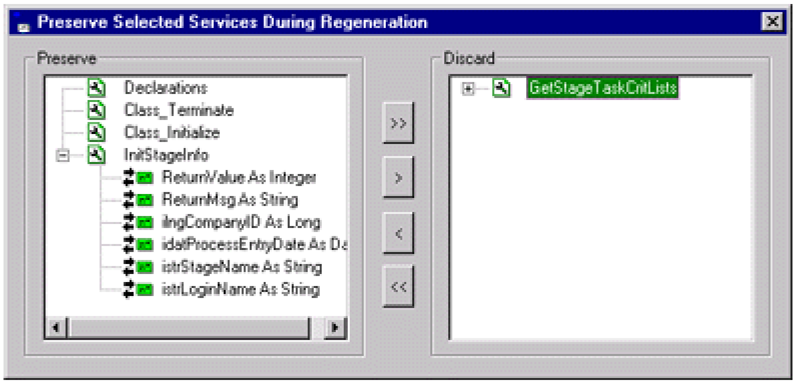
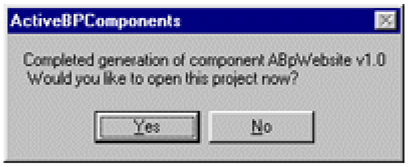


Figure 27: Preserve Custom Modifications to a GDAC Dialog Box

To move a Service from the <Preserve> list box to the <Discard> list box, select the Service in the <Preserve> list box and press the single service discard button (>). To add a Service to the <Preserve> list box from the <Discard> list box, select a Service in the <Discard> list box and press the single service preserve button (<). To move all Services from the <Discard> list box to the <Preserve> list box, press (<<). Press the multiple service discard button (>>) to move all Services from the <Preserve> list box to the <Discard> list box. This action tells the code generator to regenerate all code. To continue with the generation process, close the [**Preserve Selected Services During Regeneration**] dialog box.

After a GDAC is generated, ActiveBlueprints EasyObjects provides the developer with the opportunity to view the source code that was just created by bringing up the dialog shown in Figure 28. To view the code, click <Yes>. A Visual Basic development environment appears with the generated source code.



Completed generation of component ABpWebsite v1.0
Would you like to open this project now?

Figure 28: View Generated Source Code Dialog Box

Component and Service Explorers

Symbol	Description
📚	Component Library
◆	Standard Component
🔧	Standard Service
➡🟩	Input Service Element
⬅🟩	Output Service Element
⇅🟩	Input/Output Service Element
🔧🛢	GDAC/GDAC Service
🔧🛢🟥	Error with GDAC/ GDAC Service
➡🟥	Error with GDAC Input Service
⬅🟥	Error with GDAC Output Service
⇅🟥	Error with GDAC Input/Output Service

Table 2: Icons used in the Component and Service Explorers

GDAC Service Wizard

Button	Control	Description
◀	Previous	Return to previous dialog
▶	Next	Move to next dialog
■	Cancel/Exit	Exit without committing changes
🏁	Finish	Exit and commit changes

Table 3: GDAC Service Wizard Navigation Controls

Appendix A: ASP Web Sites

In order to use an Active Server Pages web site created with ActiveBlueprints, the site must be registered to a web server. This appendix provides an overview of how to register and view web pages, and a troubleshooting guide to help you fix problems associated with using web pages.

Software Requirements

Web Browser

Use any web browser that supports Cascading Style Sheets Level 1 (CSS1). The following web browsers have been tested:

Microsoft Internet Explorer 4.0 or greater
Netscape Navigator 4.0 or greater

Web Server

Use any web server that supports Active Server Pages. The following web servers have been tested:

Windows 95/98

MS Personal Web Server 3.0 or greater

Active Server Pages 1.0b (for IIS 3.0)

Support for the Apache Web Server with Active Server Pages will be added in the near future for Windows 95/98/NT.

Registering Web Pages

The first time ActiveBlueprints generates an Active Server Pages web site, the site must be registered to a web server before the pages can be viewed through a browser. The following sections show how to register Active Server Pages. Register the Active Server Pages just once for each web site generated by ActiveBlueprints. The following information regarding Microsoft Personal Web Server and Internet Information Server is meant to supplement the documentation provided by Microsoft. Please refer to Microsoft’s documentation for further details about either of their servers.

Microsoft Personal Web Server

A. Select **Settings** | **Control Panel** from the <Start> button on the Windows Task Bar to bring up the Windows Control Panel.

B. Select the Personal Web Server icon in the Control Panel and the [**Personal Web Server Properties**] dialog box appears.

C. Make sure that the Personal Web Server is running. Check the <Web Server State> box in the <Startup> tab.

D. Select the Administration tab in the [**Personal Web Server Properties**] dialog box and press the <Administration> button. The Microsoft Internet Explorer browser will start with the Internet Services Administrator web page.

E. Under Select the PWS Service to view:, select the <WWW Administration> link. The <Internet Services Administrator - WWW Administration> page appears with <Service>, <Directories>, and <Logging> tabs. If a problem occurs, see *Web Page Administration Problems* on page 48.

F. Select the < Directories > tab and click <Add…> under the Action… column. The <WWW Administrator – Directory Add> page appears (see Figure 29).

Figure 29: WWW Administrator - Directory Add Web Page.

G. Fill in the page with the appropriate information.

⌨ Directory: (*path of the Active Server Pages generated by ActiveBlueprints*)

Virtual Directory

🖱 Virtual Directory: ✓

⌨ Directory Alias: (*name you will use to access the Active Server Pages*)

Access

🖱 Read: ✓

🖱 Execute: ✓

H. Press <OK> to register the virtual directory of the Active Server Pages generated by ActiveBlueprints.

Microsoft Internet Information Server

A. Log into the NT server with IIS administrative privileges.

B. Start the Internet Service Manager (the Microsoft default installation shortcut is **Start** | **Programs** | **Windows NT 4.0 Option Pack** | **Microsoft Internet Information Server**). The [**Microsoft Management Console**] appears.

C. Right-click the Default Web Site icon under the Internet Information Server node in the Management Console Explorer and select **New** | **Virtual Directory**. The [**New Virtual Directory Wizard**] appears.

D. The [**New Virtual Directory Wizard**] provides a series of screens that assists you in the registration of the web pages. Make sure that the following access permissions for the virtual directory are checked:

🖱 Allow Read Access: ✓

🖱 Allow Script Access: ✓

E: Press the <Finish> button to register the web pages to the default web site of the server.

Viewing Registered Web Pages

Once the web pages have been registered with to the web server, you can view them using one of the following methods for specifying the URL:

http://<*Name of machine*>/<*Name of Virtual Directory*>

http://localhost/<*Name of Virtual Directory*>

http://<*IP Address of machine*>/<*Name of Virtual Directory*>

The first method is preferred, but all three methods work equally well. If the web server is located on the Internet, then the web pages may be viewed from anywhere. If the web server is behind a firewall, then the web pages may not be accessible externally. In this case the pages are visible locally on an Intranet.

Troubleshooting

Web Page Administration Problems

Symptoms

An error message appears when the <WWW Administration> link is selected in the Internet Services Administrator web page for Microsoft Personal Web Server.

Possible Solution

The web browser might be using a proxy server to access the Internet. The Internet Services Administrator web page uses a local address; the browser should bypass the proxy server for local (Intranet) addresses.

For Microsoft Internet Explorer

1. Select the <Internet> icon in the Control Panel. The [**Internet Properties**] dialog box appears.
2. Select the <Connection> tab and make sure that the following is checked in the <Proxy server> box:

   🖱 Bypass proxy server for local (Intranet) addresses: ✓
3. Press the <OK> button to commit the changes to the [Internet Properties] dialog box. Try clicking the <WWW Administration> hyperlink again.

For Netscape Navigator

1. Select the **Edit** | **Preferences** from Navigator's main menu bar. The [**Preferences**] dialog box appears.
2. Expand the <Advanced> node in the <Category> tree view and select the <Proxies> node.
3. If the <Manual proxy configuration> radio button is selected, press <View…>. The [**Manual Proxy Configuration**] dialog box appears.
4. In the <Exceptions> frame of the [**Manual Proxy Configuration**] dialog box, enter the name of the machine in the edit box labeled <Do not use with proxy servers for domains beginning with:>.
5. Press <OK> to close the [**Manual Proxy Configuration**] dialog box and commit the changes.
6. Press <OK> again to close the [**Preferences**] dialog box. Try clicking the <WWW Administration> hyperlink again.

General Failures

Symptoms

After generating an application to Active Server Pages and setting up the virtual directory in the web server, blank pages appear in the application, pages with the message “Document contains no data” or “Error 404: object not found” appear in the application, or the application does not work properly.

Possible Solution

We suspect that this problem has to do with the way Microsoft’s web servers (IIS or Persona Web Server) caches ASP code. The cached code needs to be flushed from the cache and reloaded from the disk. To flush the cache, remove the reference to the virtual directory in the web server and add it back again. Do not delete the physical directory. If that does not work, delete both the physical and the virtual directory. Regenerate the code from ActiveBlueprints and reregister the virtual directory. If that fails, stop and then restart the web server.