VISUAL SOURCE

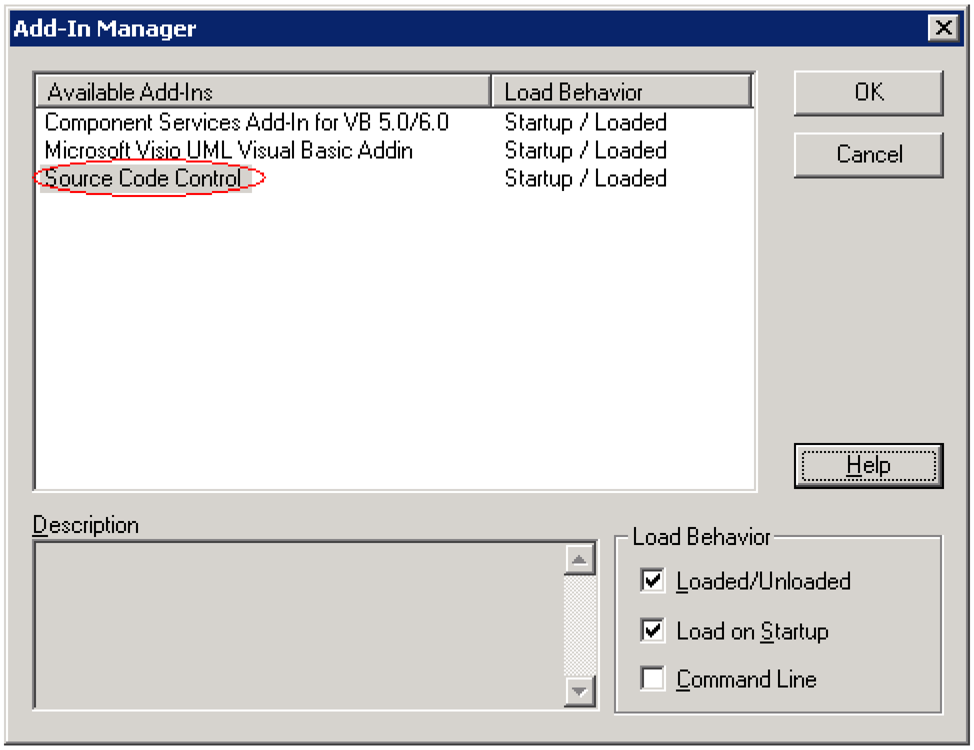
SourceSafe can be integrated into Visual Basic 6.0 to source control the VB forms, modules, class modules, etc.

To integrate SourceSafe with VB 6.0, we can do as follows:

1. Choose SourceSafe as the current source control provider.
For information on how to do it, refer to: http://www.kevingao.net/sourcesafe/microsoft-source-code-control-interface-msscci-registry-entries.html

2. Open VB 6.0 and check if the Source Code Control add-in is loaded through menu Add-Ins -> Add-In Manager.

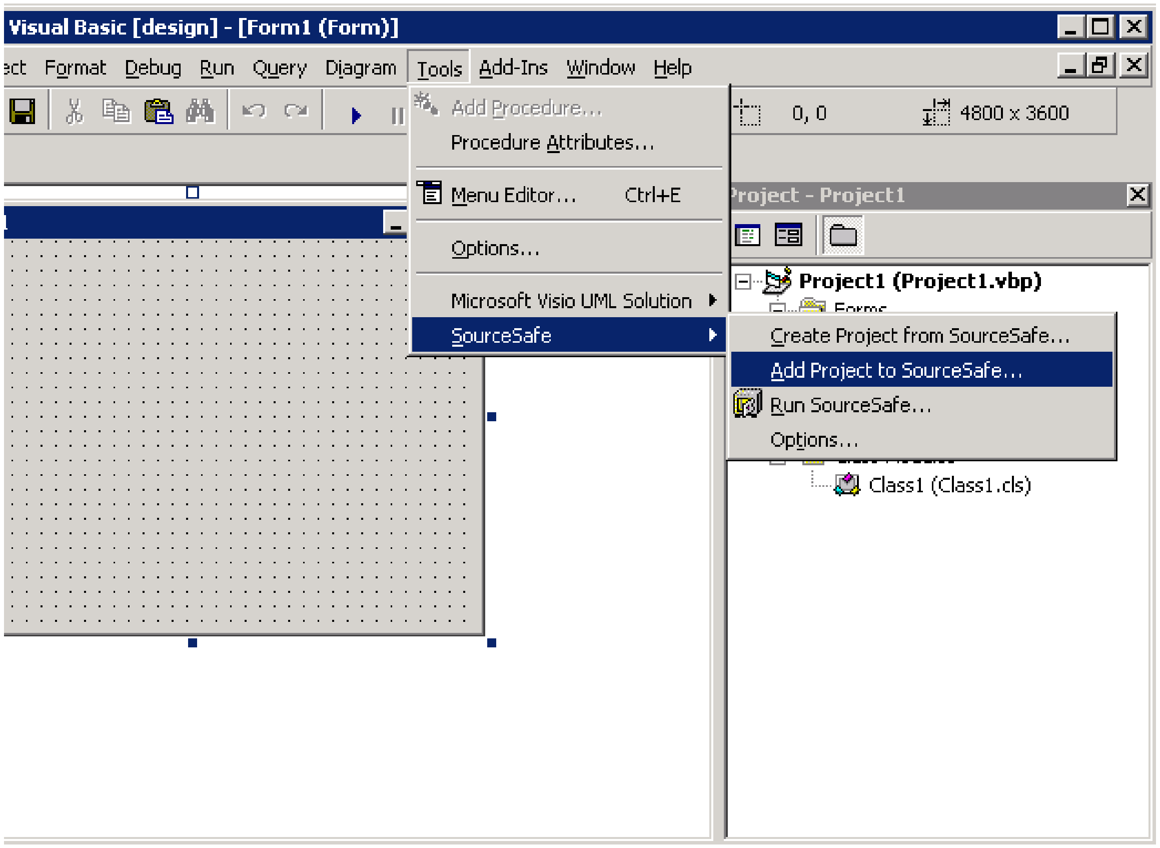
If yes, we should be able to find the SourceSafe command under Tools menu.
If no, please edit the vbaddin.ini file by going to Start -> Run: vbaddin.ini and adding the line "vbscc=3 " in the file.



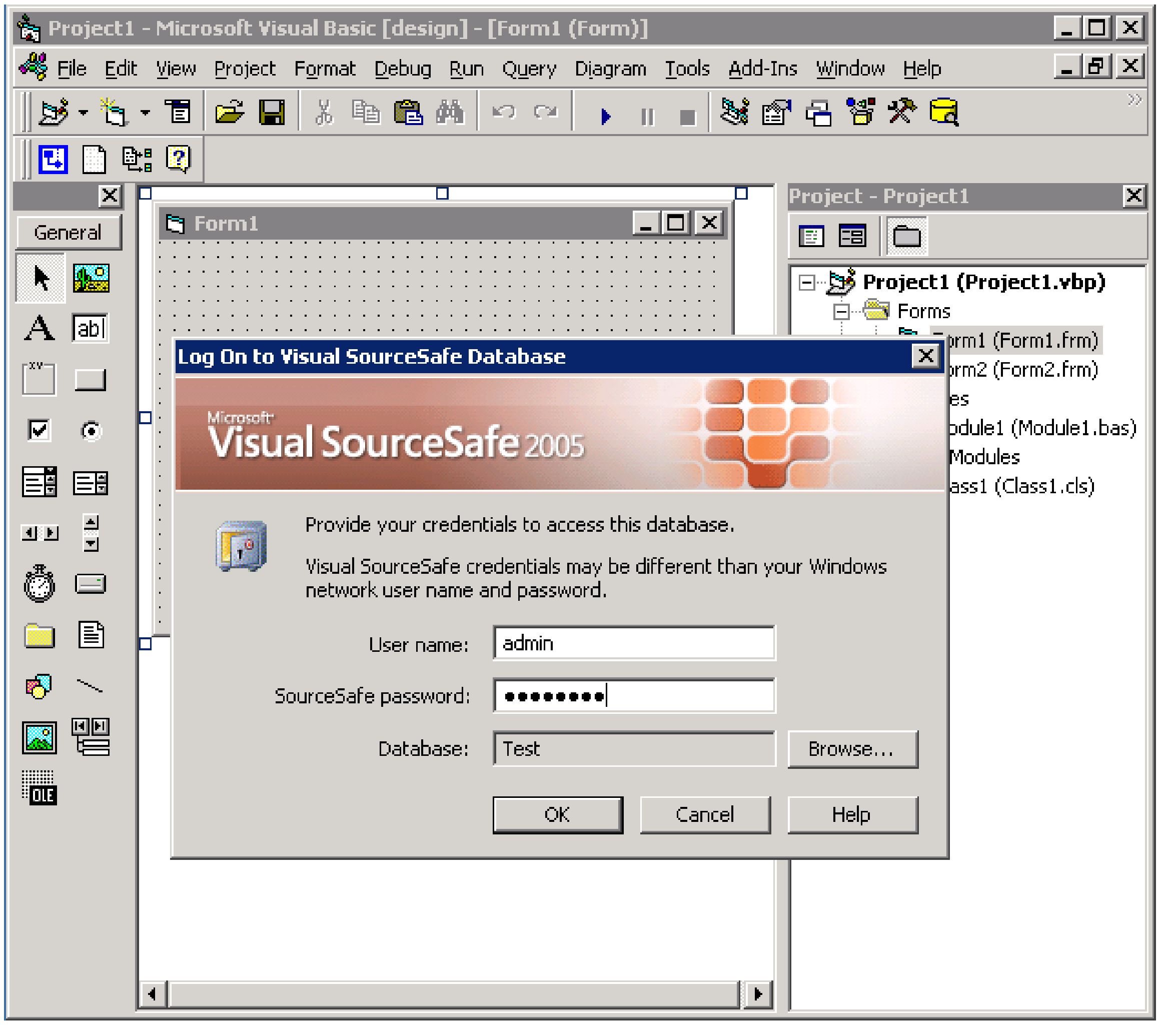
Available Add-Ins	Load Behavior
Component Services Add-In for VB 5.0/6.0	Startup / Loaded
Microsoft Visio UML Visual Basic Addin	Startup / Loaded
Source Code Control	Startup / Loaded

Load Behavior
- [✓] Loaded/Unloaded
- [✓] Load on Startup
- [ ] Command Line

3. Add the VB project into source control of SourceSafe by clicking menu Tools -> SourceSafe -> Add Project to SourceSafe.


Visual Basic [design] - [Form1 (Form)]
ect Format Debug Run Query Diagram Tools Add-Ins Window Help
Add Procedure...
Procedure Attributes...
Menu Editor... Ctrl+E
Options...
Microsoft Visio UML Solution
SourceSafe
Create Project from SourceSafe...
Add Project to SourceSafe...
Run SourceSafe...
Options...
0, 0
4800 x 3600
Project - Project1
Project1 (Project1.vbp)
Forms
Class1 (Class1.cls)
(Add VB Project to SourceSafe)

4. In the following Log On to SourceSafe Database dialog box, enter the credentials to log on a VSS DB.


Project1 - Microsoft Visual Basic [design] - [Form1 (Form)]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
General
Form1
Project - Project1
Project1 (Project1.vbp)
Forms
orm1 (Form1.frm)
orm2 (Form2.frm)
es
odule1 (Module1.bas)
Modules
ass1 (Class1.cls)
Log On to Visual SourceSafe Database
Microsoft Visual SourceSafe 2005
OK
Cancel
Help
Browse...

Provide your credentials to access this database.

Visual SourceSafe credentials may be different than your Windows network user name and password.

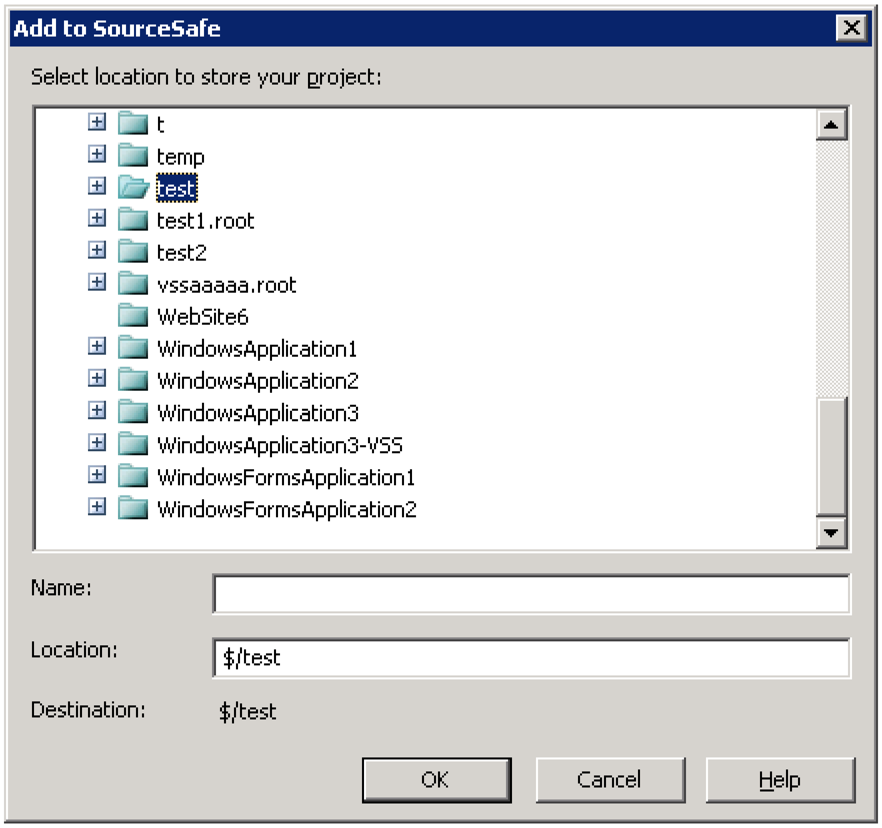
User name: admin

SourceSafe password: ●●●●●●●●●

Database: Test

(Log on to VSS Database)

5. Choose a location in the VSS project tree to store the VB project.


Add to SourceSafe

Select location to store your project:

t
temp
test
test1.root
test2
vssaaaaa.root
WebSite6
WindowsApplication1
WindowsApplication2
WindowsApplication3
WindowsApplication3-VSS
WindowsFormsApplication1
WindowsFormsApplication2

Name:

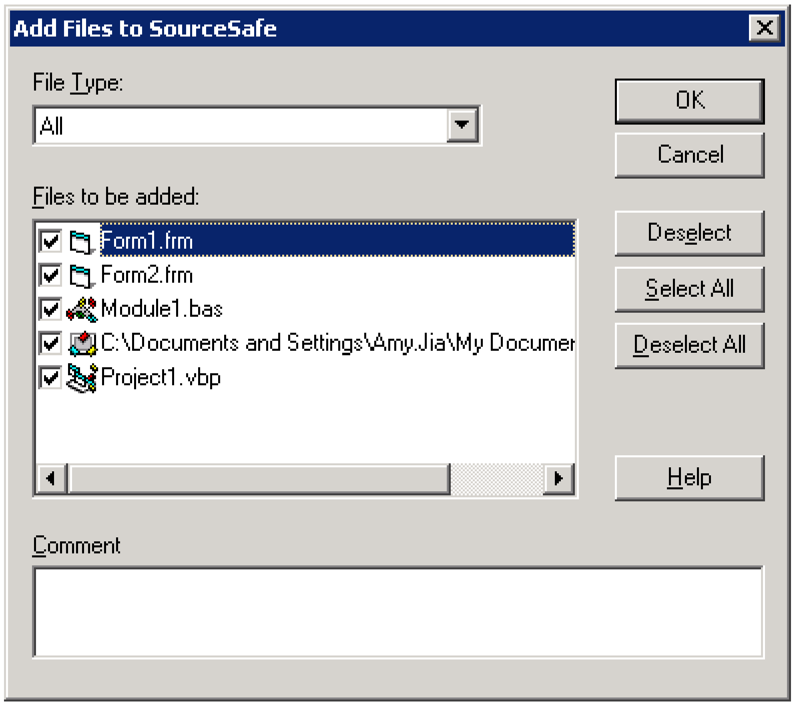
Location: $/test

Destination: $/test

OK Cancel Help

(Choose location to place the VB project)

6. Select the files we want to add into SourceSafe for source control and click OK.


Add Files to SourceSafe

File Type: All

Files to be added:

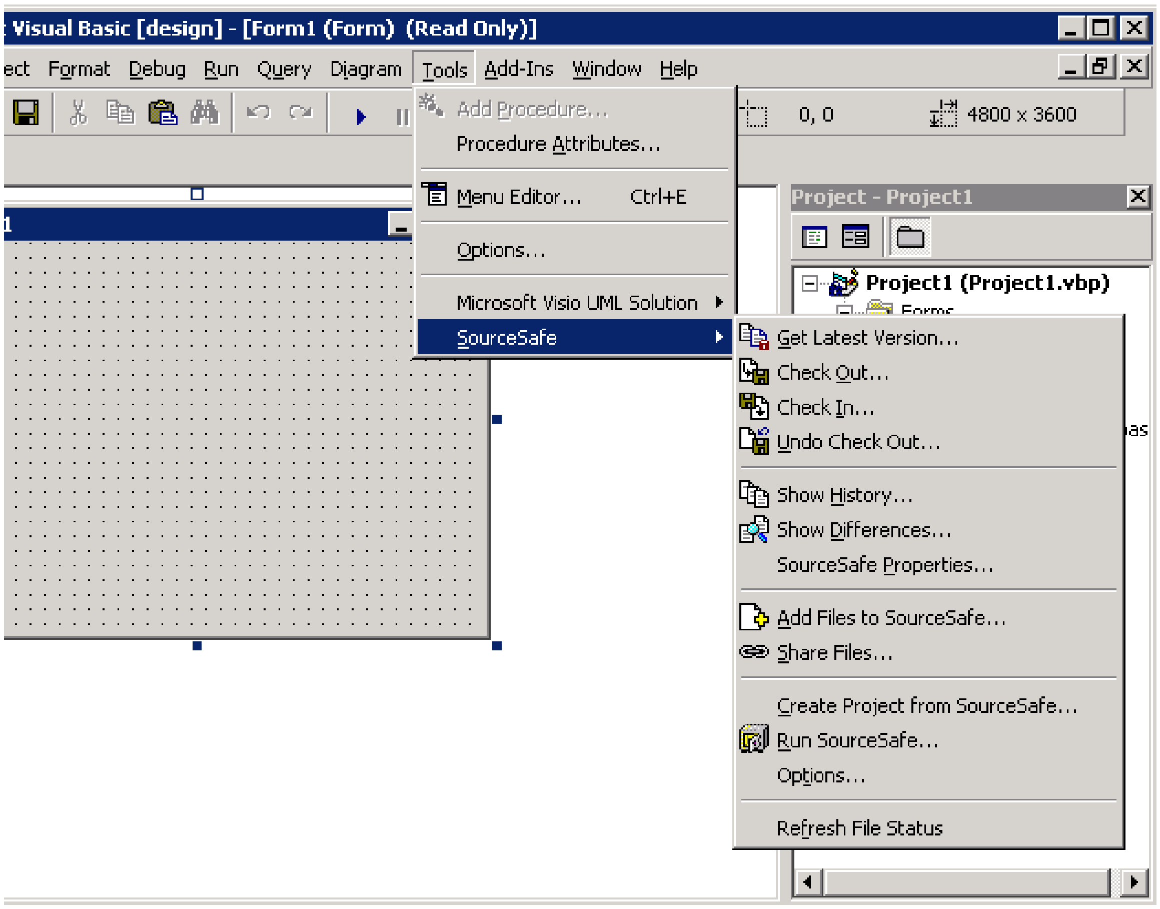
[✓] Form1.frm
[✓] Form2.frm
[✓] Module1.bas
[✓] C:\Documents and Settings\Amy.Jia\My Documer
[✓] Project1.vbp

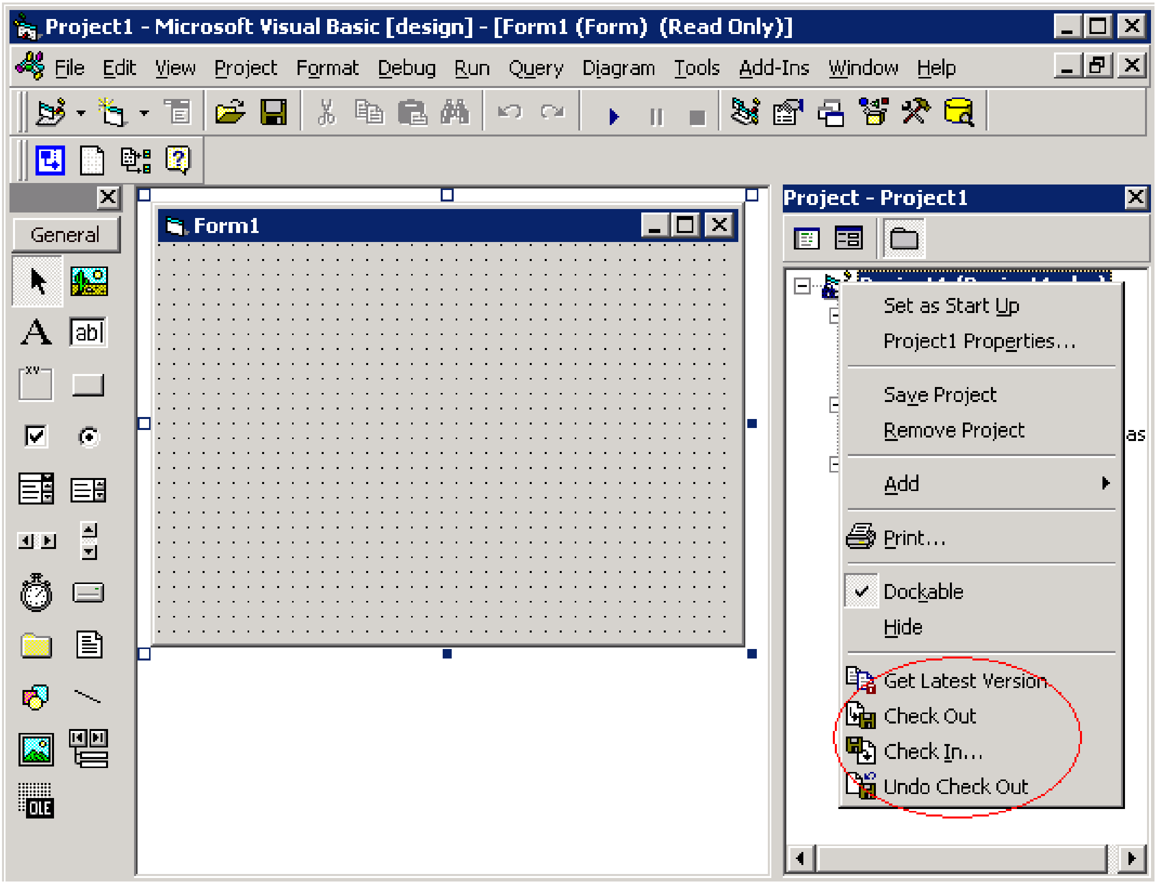
OK Cancel Deselect Select All Deselect All Help

Comment

(Add VB files to SourceSafe)

7. Now, all of the files are under source control of SourceSafe. We can find the SourceSafe functions through menu Tools -> SourceSafe. We can also access some of the functions by right-clicking the file in the Project Explorer.





Q. Where is the main copy of the Visual Basic project stored?

A. The main copy of the project is located in the Visual SourceSafe database. After modifying their local copies, all users should check the files into Visual SourceSafe. This will cause the latest version of the user's work to be stored in Visual SourceSafe. Every time you open the Visual Basic project, Visual Basic will use the information in the local Mssccprj.scc file to retrieve the latest version of the files from Visual SourceSafe and replace the local copies of the user's Visual Basic files.

Q. If the Allow Multiple Checkouts option is enabled, what type of files can I allow multiple users to check out simultaneously?

A. Binary files cannot be checked out by multiple users simultaneously. Forms, User Controls, and Property pages are added to Visual SourceSafe as binary files. However, Project files, Modules, Class Modules, and User Connections can be checked out by multiple users at the same time.

Q. How do I know that a file has been Checked Out by some other user?

A. From Visual Basic, select the file in the project window for which you want to know the status, then click on Tools->SourceSafe->SourceSafe Properties->Check Out Status tab. You will get the following information: Which users have the file checked out, Check Out Date and time, Computer name that the file was Checked Out to and the Visual SourceSafe project that contains this file.

Q. If two users check out a project file, how does Visual SourceSafe avoid overwriting changes made by one user when the other user checks in their copy of the file?

A. Let us assume that UserA and UserB checked out the same project file, Proj1.vbp. UserA changes the file and checks it in. This will update the file stored in the Visual SourceSafe database. UserB changes their local copy of the file and checks it in. Before updating the server copy, Visual SourceSafe will compare the files in UserB's working folder and the file stored in the Visual SourceSafe database. If UserA and UserB have made different changes to the same parts of the vbp file, Visual SourceSafe will display the Visual Merge window and give UserB the opportunity to specify which changes he or she wants to keep.
Q. If new users want to modify the project, what are the steps they need to follow?

A. Follow the steps given in the "Creating local copy of the Project from SourceSafe" section. This will create a local copy of the project from Visual SourceSafe database.
Q. While trying to modify a user control after doing a checkout, it gives an error "Can't edit module." How can I fix this error?

A. You will get this error if do not have the form that contains the user control checked out. You can avoid this error by checking out the form prior to editing the user control.
Q. Is it necessary to check out the project file when I only want to edit one of the forms contained in the project file?
A. If you have a ActiveX control embedded in your form, you have to check out the project file also.
Q. I've checked out and modified my files. At what point do the copies stored in the Visual SourceSafe database get updated?

A. The copies of the files stored in the Visual SourceSafe database only get updated when you check your files back in. This causes the changes you've made to your local copy of the file transferred to the file stored in Visual SourceSafe.
Q. When I open a project in Visual Basic that is under Source Code control, does Visual SourceSafe get the latest version of the project?

A. In order to get the latest version of your project from Visual SourceSafe, you need to set the Tools->SourceSafe->Options->'Get the latest checked-in version while opening the Visual Basic project' option to Yes.

ASSIGNMENT STATEMENTS

Assignment statements carry out assignment operations, which consist of taking the value on the right side of the assignment operator (=) and storing it in the element on the left, as in the following example.

```
v = 42
```

In the preceding example, the assignment statement stores the literal value 42 in the variable v.
Eligible Programming Elements

The programming element on the left side of the assignment operator must be able to accept and store a value. This means it must be a variable or property that is not ReadOnly (Visual Basic), or it must be an array element. In the context of an assignment statement, such an element is sometimes called an lvalue, for "left value."

The value on the right side of the assignment operator is generated by an expression, which can consist of any combination of literals, constants, variables, properties, array elements, other expressions, or function calls. The following example illustrates this.

```
x = y + z + findResult(3)
```

The preceding example adds the value held in variable y to the value held in variable z, and then adds the value returned by the call to function findResult. The total value of this expression is then stored in variable x.
Data Types in Assignment Statements

In addition to numeric values, the assignment operator can also assign String values, as the following example illustrates.

```
Dim a, b As String
a = "String variable assignment"
b = "Con" & "cat" & "enation"
' The preceding statement assigns the value "Concatenation" to b.
```

You can also assign Boolean values, using either a Boolean literal or a Boolean expression, as the following example illustrates.

```
Dim r, s, t As Boolean
r = True
s = 45 > 1003
t = 45 > 1003 Or 45 > 17
' The preceding statements assign False to s and True to t.
```

Similarly, you can assign appropriate values to programming elements of the Char, Date, or Object data type. You can also assign an object instance to an element declared to be of the class from which that instance is created.
Compound Assignment Statements

Compound assignment statements first perform an operation on an expression before assigning it to a programming element. The following example illustrates one of these operators, +=, which increments the value of the variable on the left side of the operator by the value of the expression on the right.

```
n += 1
```

The preceding example adds 1 to the value of n, and then stores that new value in n. It is a shorthand equivalent of the following statement:

```
n = n + 1
```

A variety of compound assignment operations can be performed using operators of this type. For a list of these operators and more information about them, see Assignment Operators.

The concatenation assignment operator (&=) is useful for adding a string to the end of already existing strings, as the following example illustrates.

```
Dim q As String = "Sample "
q &= "String"
' q now contains "Sample String".
```

Type Conversions in Assignment Statements

The value you assign to a variable, property, or array element must be of a data type appropriate to that destination element. In general, you should try to generate a value of the same data type as that of the destination element. However, some types can be converted to other types during assignment.

For information on converting between data types, see Type Conversions in Visual Basic. In brief, Visual Basic automatically converts a value of a given type to any other type to which it widens. A widening conversion is one in that always succeeds at run time and does not lose any data. For example, Visual Basic converts an Integer value to Double when appropriate, because Integer widens to Double. For more information, see Widening and Narrowing Conversions.

Narrowing conversions (those that are not widening) carry a risk of failure at run time, or of data loss. You can perform a narrowing conversion explicitly by using a type conversion function, or you can direct the compiler to perform all conversions implicitly by setting Option Strict Off. For more information, see Implicit and Explicit Conversions.

Data Reports

You have learned how to build a database in Visual Basic 6 in previous chapters, however you have not learned how to display the saved data in a report. Reports are important and useful in many respects because they provide useful and meaningful information concerning a set of data. In this chapter, we will show you how to create a report in Visual Basic 6.

In previous versions of Visual Basic 6, there is no primary reporting . Previous versions of Visual basic 6 uses Crystal Reports tool, a software from Seagate. Fortunately, Microsoft has integrated a good report writer into Visual Basic 6, so you no longer need to use Crystal Report.
40.2 Steps in building your report in Visual Basic 6

Visual Basic 6 provides you with a data report designer to create your report, it is somewhat similar to data report designer in Microsoft Access. The data report designer has its own set of controls which allow you to customize your report seamlessly. The steps in creating the report in VB6 are listed below:

Step 1: Adding Data Report

Start Visual Basic as a Standard EXE project. From the Project menu in the VBE, select Add Data Report in the dropdown menu. Now, you will be presented with the data report environment, as shown in Figure 40.1. The data report environment contains 6 controls, they are RptTextBox, RptLine, RptFunction, RptLabel, RptImage and RptShape.

You can customize your report here by adding a title to the page header using the report label RptLabel. Simply drag and draw the RptLabel control on the data report designer window and use the Caption property to change the text that should be displayed. You can also add graphics to the report using the RptImage control.

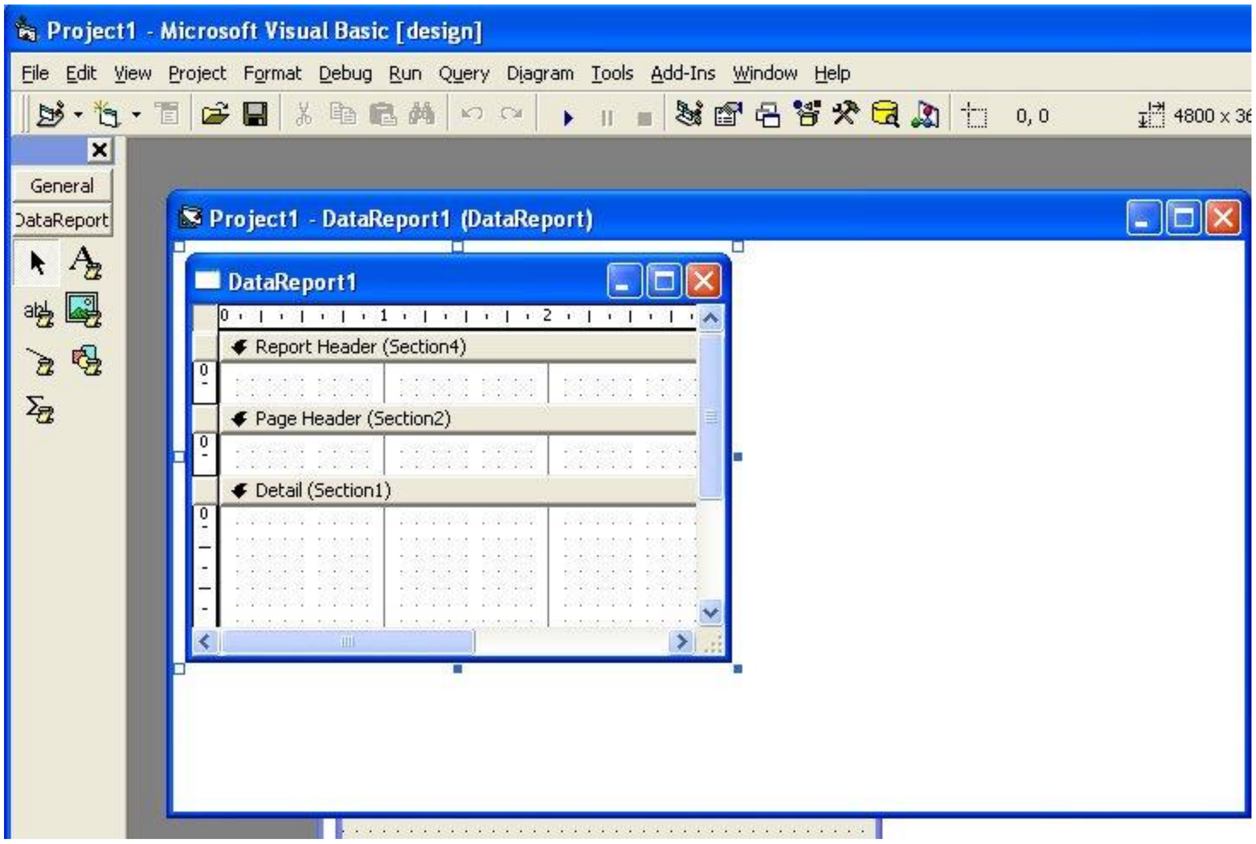

Project1 - Microsoft Visual Basic [design]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
0, 0
4800 x 36
General
DataReport
Project1 - DataReport1 (DataReport)
DataReport1
Report Header (Section4)
Page Header (Section2)
Detail (Section1)

Figure 40.1: The Data Report Environment

Step 2: Connecting the report to database using Data Environment Designer
Click the Project menu, then select Data Environment. from the drop-down menu. The default data environment will appear, as shown in figure 40.2

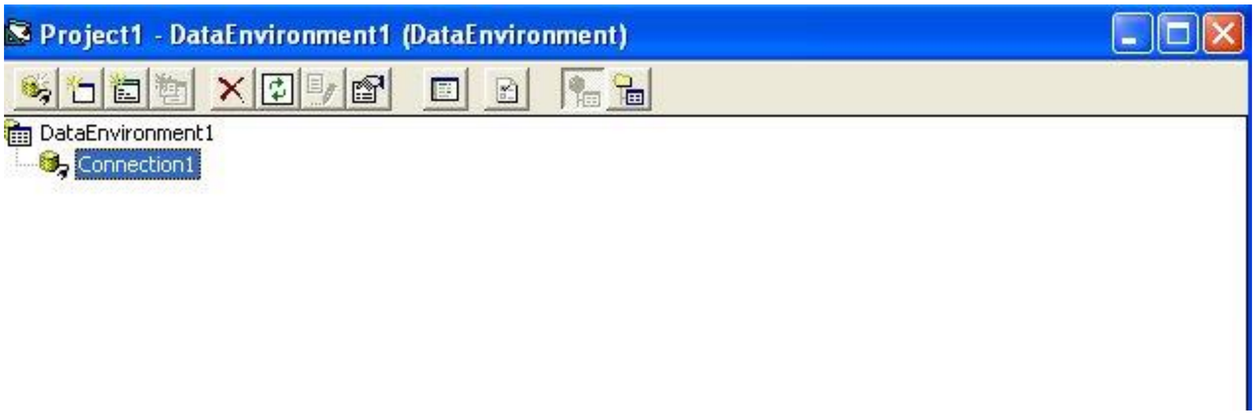
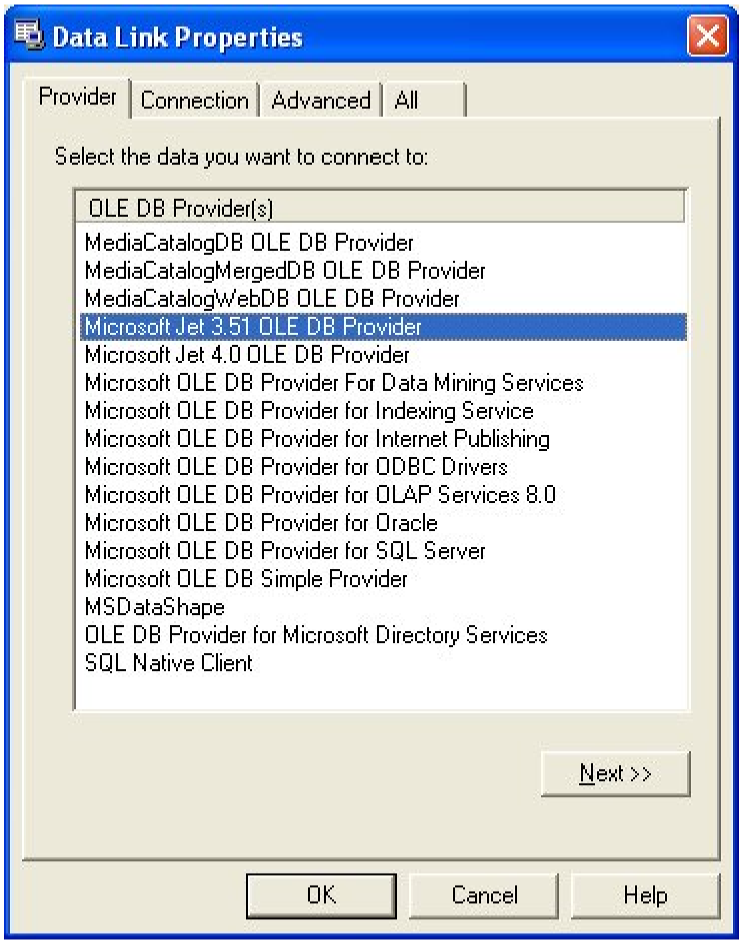


Figure 40.2: Data Environment

Now, to connect to the database, right-click connection1 and select Microsoft Jet 3.51 OLE DB Provider (as we are using MS Access database) from the Data Link Properties dialog (as shown in Figure 40.3), then click next.

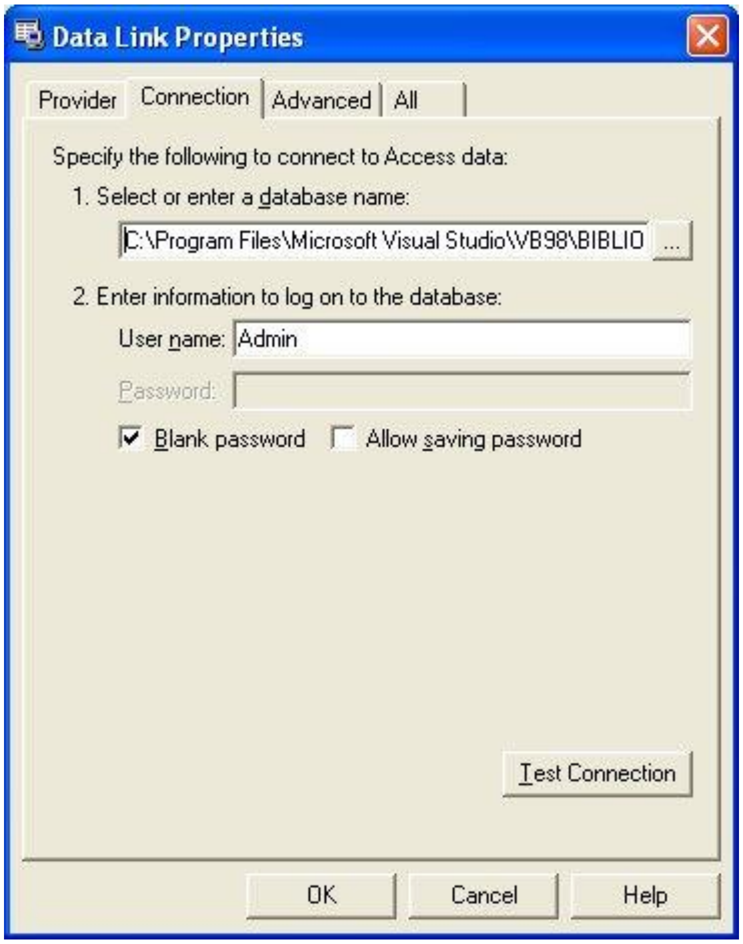


Select the data you want to connect to:

OLE DB Provider(s)
MediaCatalogDB OLE DB Provider
MediaCatalogMergedDB OLE DB Provider
MediaCatalogWebDB OLE DB Provider
Microsoft Jet 3.51 OLE DB Provider
Microsoft Jet 4.0 OLE DB Provider
Microsoft OLE DB Provider For Data Mining Services
Microsoft OLE DB Provider for Indexing Service
Microsoft OLE DB Provider for Internet Publishing
Microsoft OLE DB Provider for ODBC Drivers
Microsoft OLE DB Provider for OLAP Services 8.0
Microsoft OLE DB Provider for Oracle
Microsoft OLE DB Provider for SQL Server
Microsoft OLE DB Simple Provider
MSDataShape
OLE DB Provider for Microsoft Directory Services
SQL Native Client

Figure 40.3

Now, you need to connect to the database by selecting a database file from your hard disk. For demonstration purpose, we will use the database BIBLIO.MDB that comes with Visual Basic, as shown in Figure 40.4. The path to this database file is C:\Program Files\Microsoft Visual Studio\VB98\BIBLIO.MDB. This path varies from computers to computers, depending on where you install the file. After selecting the file, you need to test the connection by clicking the Test Connection button at the right bottom of the Data Link Properties dialog. If the connection is successful, a message that says 'Test Connection Succeeded' will appear. Click the OK button on the message box to return to the data environment. Now you can rename connection1 to any name you like by right-clicking it. For example, you can change it to MyConnection. You may also change the name of DataEnvironment1 to MyDataEnvironment using the Properties window.



Specify the following to connect to Access data:

1. Select or enter a database name:

C:\Program Files\Microsoft Visual Studio\VB98\BIBLIO

2. Enter information to log on to the database:

User name: Admin

Password:

[✓] Blank password [ ] Allow saving password

Figure 40.4

Step 3: Retrieving Information from the Database

In order to use the database in your report, you need to create query to retrieve the information from the database. Here , we will use SQL command to create the query. First of all, right click on MyConnection to add a command to the data environment. The default command is Command1, you can rename it as MyCommand, as shown in Figure 40.5.

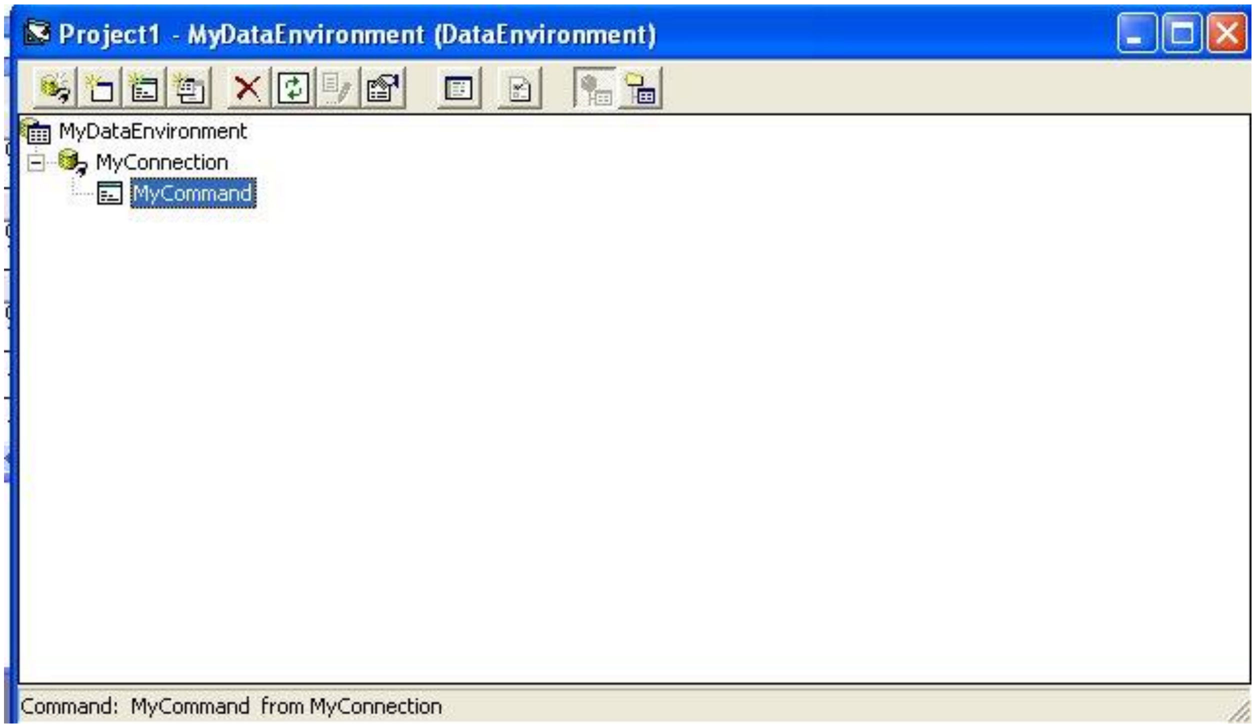

Project1 - MyDataEnvironment (DataEnvironment)
MyDataEnvironment
MyConnection
MyCommand
Command: MyCommand from MyConnection

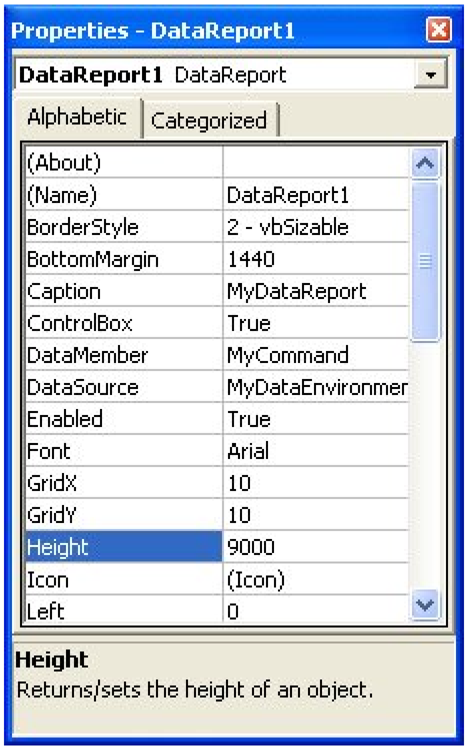
Figure 40.5: MyCommand

In order to use SQL command, right-click MyCommand and you can see its properties dialog. At the General tab, select SQL statement and key in the following SQL statement:

```
SELECT Au_ID, Author
FROM Authors ORDER BY Author
```

This command is to select all the fields from the Authors table in the Biblio.Mdb database. The command ORDER BY Author is to arrange the list in ascending order according to the Authors' Names.

Now, you need to customize a few properties of your data report so that it can connect to the database. The first property to set is the DataSource, set it to MyDataEnvironment. Next, you need to set the DataMember property to MyCommand,as shown in Figure 40.6


Properties - DataReport1

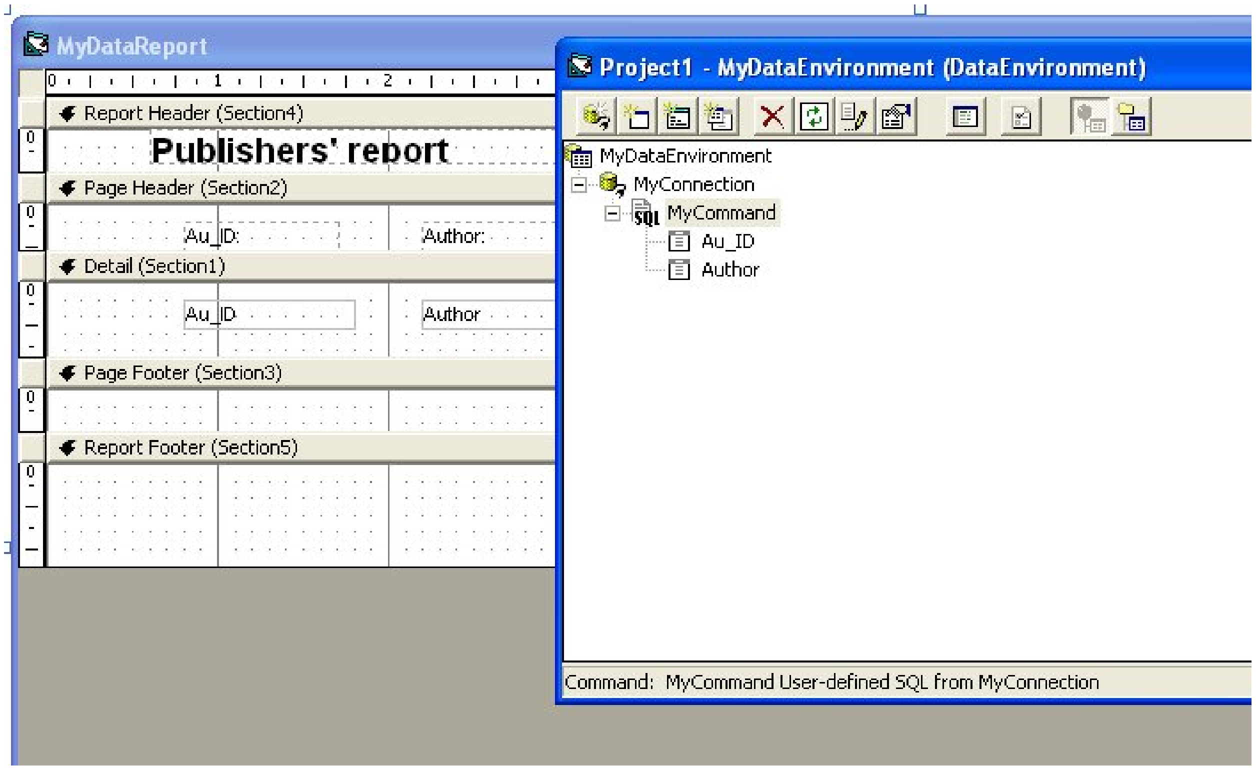
DataReport1 DataReport

(About)	
(Name)	DataReport1
BorderStyle	2 - vbSizable
BottomMargin	1440
Caption	MyDataReport
ControlBox	True
DataMember	MyCommand
DataSource	MyDataEnvironmer
Enabled	True
Font	Arial
GridX	10
GridY	10
Height	9000
Icon	(Icon)
Left	0

Height
Returns/sets the height of an object.

Figure 40.6: Properties of

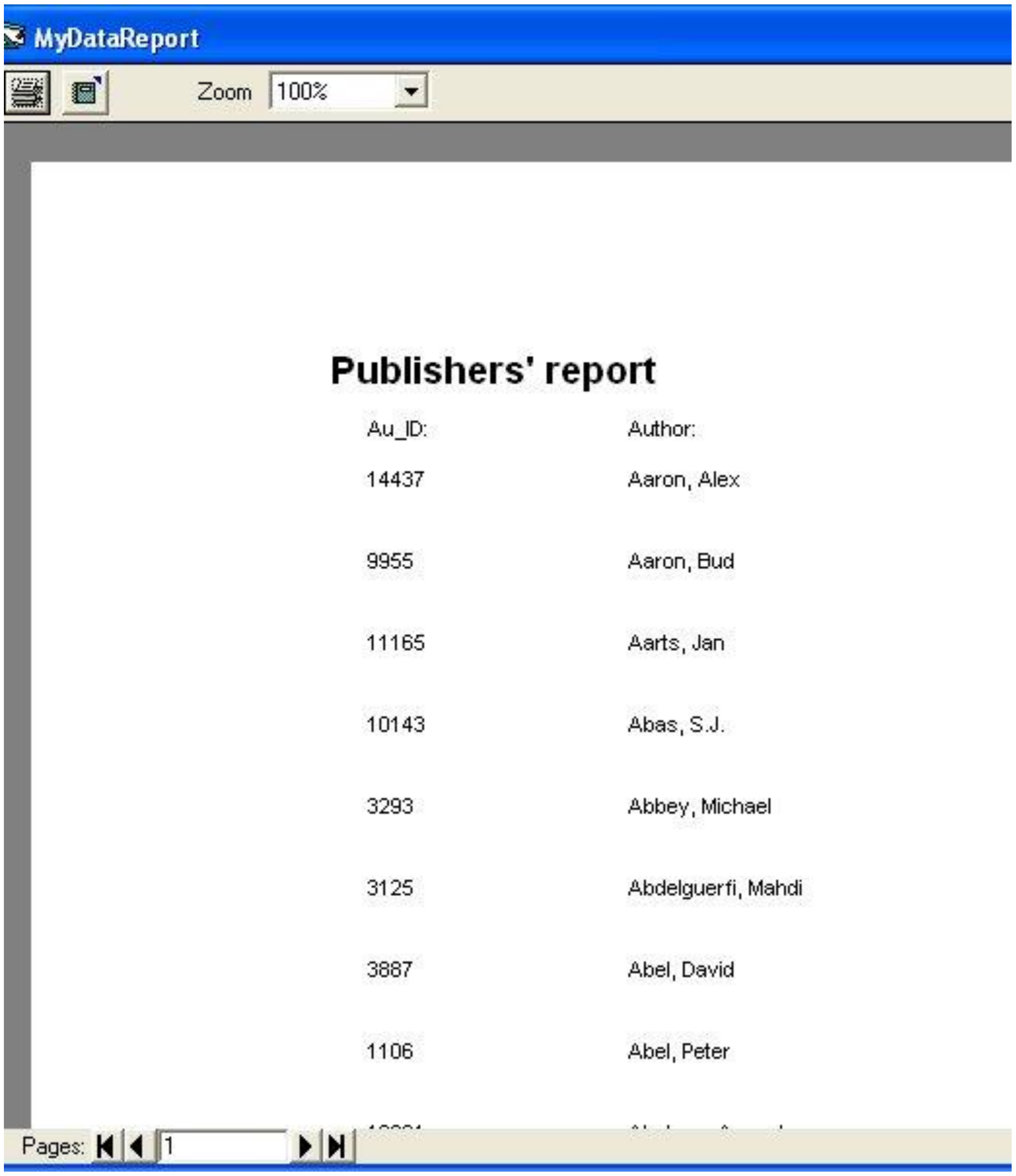
To add data to your report, you need to drag the fields from MyCommand in MyDataEnvironment into MyDataReport, as shown in Figure 40.7.Visual Basic 6 will automatically draw a RptTextBox, along with a RptLabel control for each field on the report. You can customize the look of the labels as well as the TextBoxes from the properties window of MyDataReport.



Command: MyCommand User-defined SQL from MyConnection

Figure 40.7

The Final step is to set MydataReport as the Startup form from the Project menu, then run the program. You will see your report as shown in Figure 40.8. You can print out your report.



Publishers' report

Au_ID:	Author:
14437	Aaron, Alex
9955	Aaron, Bud
11165	Aarts, Jan
10143	Abas, S.J.
3293	Abbey, Michael
3125	Abdelguerfi, Mahdi
3887	Abel, David
1106	Abel, Peter

Figure 40.8: The Final Report.

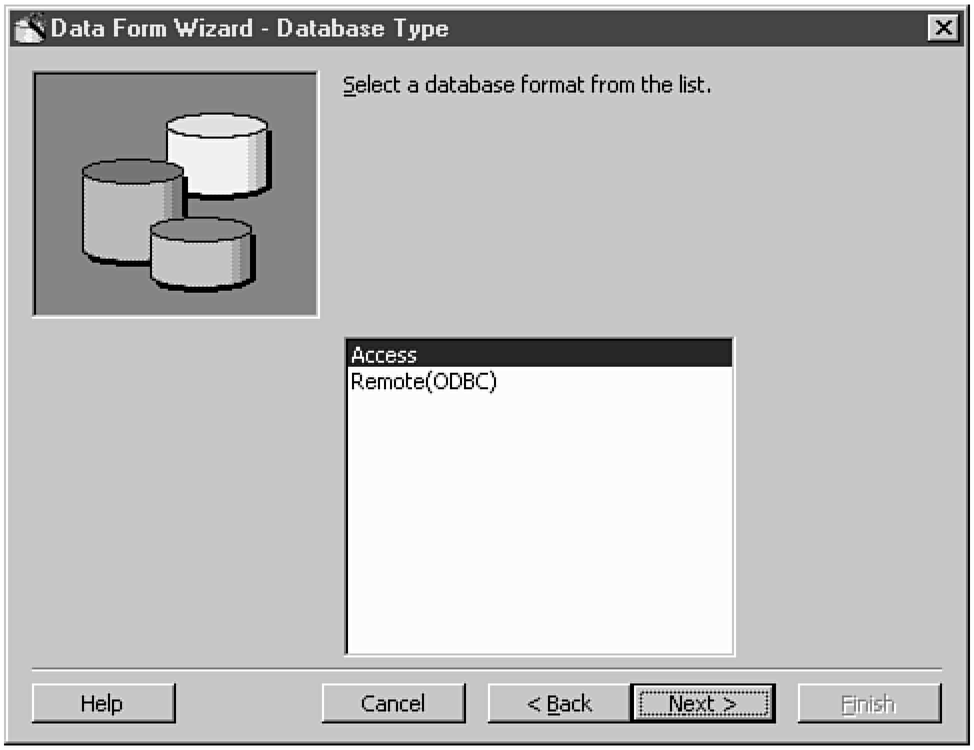
Data Form Wizard

The Data Form Wizard

The Data Form Wizard analyzes a database, locates the fields for you (you don't have to know the format of the database ahead of time), and automatically builds a form that contains an appropriate title, field names, Text Box controls for the fields, and the Data control you can use to move between the records. If you want to follow along with this exercise, start a new VB project and remove any projects currently loaded in VB.

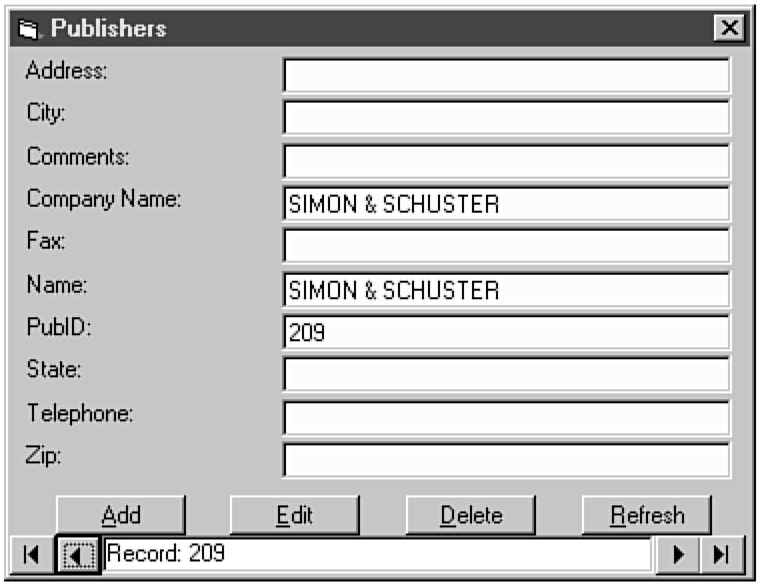
An add-in application is a tool that extends Visual Basic's development environment.

To access the Data Form Wizard, select Add-Ins | Add-In Manager. When the Add-In Manager dialog box appears, select the VB 6 Data Form Wizard and check the Loaded/Unloaded check box. Click OK to continue. Now select Add-Ins | Data Form Wizard. Visual Basic displays the Data Form Wizard's opening window. When you click Next, you see the database-selection dialog box shown in Figure 15.2.


Data Form Wizard - Database Type
Select a database format from the list.
Access
Remote(ODBC)
Help
Cancel
< Back
Next >
Finish
Figure 15.2 The Data Form Wizard's database selection tool.

Continue following the wizard's requests to create the form. For example, you will have to tell the wizard the kind of database for which you want to create a form. After you select a database, the next dialog box asks you for the database name (which you can browse for) and a data source such as a table or query. Select the kind of form and then on the Record Source dialog box, you must select a table. Copy all the fields you want from that table to the final form. You then can click the options you want and click Finish to generate the form.

The form that the Data Form Wizard generates might not be as unique as the one you create yourself, but the form does include buttons that let the user not only change the database data, but also add and delete records as well, as Figure 15.3 shows.



Address:	
City:	
Comments:	
Company Name:	SIMON & SCHUSTER
Fax:	
Name:	SIMON & SCHUSTER
PubID:	209
State:	
Telephone:	
Zip:	

Figure 15.3 The Data Form Wizard creates a nice form.

Note

You can insert the Data Form Wizard's from into another application and then display the form with the Snow method.

The Data Form Wizard is designed to automatically generate Visual Basic forms that contain individual bound controls and procedures used to manage information derived from database tables and queries. You can use the Data Form Wizard to create either single query forms to manage the data from a single table or simple query, or Master/Detail type forms used to manage more complex one-to-many data relationships. If you are using a control, you can also create a grid or datasheet type form. The Data Form Wizard is used in conjunction with only the ADO Data control.

Use the Data Form Wizard to:
Rapidly build forms with controls bound to a data source.

Create single record, grid, and Master/Detail forms.

Rapidly create prototype data-based forms.

Since you can create three types of forms, some steps of the Data Form Wizard are different for each type.
To create single table forms or forms with a grid (datasheet) layout, the Data Form Wizard follows these steps:
Introduction

Database Type

Database

Form

Record Source

Control Selection

Finished!

Note   The grid or datasheet form layout is not available when you choose a code as a binding type.
To create Master/Detail forms, the Data Form Wizard follows these steps:
Introduction

Database Type

Form

Master Record Source

Detail Record Source

Record Source Relation

Control Selection

Finished!

In addition to the previous steps, data forms that contain ADO Data controls, also have a Connect Information step.

Error Handling In Visual Basic

Despite your best efforts to cover all possible contingencies, run-time errors will occur in your applications. You can and should do all you can to prevent them, but when they happen you have to handle them.

Introduction

The various functions, statements, properties and methods available in Visual Basic and the components used in Visual Basic expect to deal with certain types of data and behavior in your applications. For example, the CDate() function can convert a value to a Date variable. The function is remarkably flexible in the type of information it can accept, but it expects to receive data that it can use to derive a date. If you provide input that it can't convert, it raises error number 13 - "Type mismatch" - essentially saying "I can't handle this input data."

In an application, this type of error may be a program logic error (you simply passed the wrong data) or it may be a data entry error on the part of the user (you asked for a date and the user typed a name). In the first case, you need to debug the program to fix the mistake. However, there is no way for you to anticipate the behavior of the end users of the application. If the user enters data you can't handle, you need to deal with the situation.

Dealing with errors at run-time is a two step process:

Trap the Error

Before you can deal with an error, you need to know about it. You use VB's On Error statement to setup an error trap.

Handle the Error

Code in your error handler may correct an error, ignore it, inform the user of the problem, or deal with it in some other way. You can examine the properties of the Err object to determine the nature of the error. Once the error has been dealt with, you use the Resume statement to return control to the regular flow of the code in the application.

In addition to dealing with run-time errors, you may at times want to generate them. This is often done in class modules built as components of ActiveX server DLLs or EXEs. It is considered good programming practice to separate the user interface from the program logic as much as possible, so if a server component cannot deal with an error, it should raise the error in its client application rather than simply display an error message for the user.

In VB5, there is an option that allows you to specify that an application has been designed for unattended execution (this is typically used for remote server applications). If you plan to allow the application to run unattended or on a remote computer, you can't simply display an error message because there will be nobody there to see it or dismiss the message box.

Trapping Errors at Run-Time

Before you can do anything to deal with a run-time error, you need to capture the error. You use the On Error statement to enable an error trap. On Error will redirect the execution in the event of a run-time error. There are several forms of the On Error statement:

On Error Goto label

This form of the On Error statement redirects program execution to the line label specified. When you use this form of On Error, a block of error handling code is constructed following the label.

On Error Resume Next

This form of the On Error statement tells VB to continue with the line of code following the line where the error occurred. With this type of error trap, you would normally test for an error at selected points in the program code where you anticipate that an error may occur.

On Error Goto 0
On Error Goto 0 disables any error handler within the current procedure. A run-time error that occurs when no error handler is enabled or after an On Error Goto 0 is encountered will be handled using VB's default error handling logic.
It's not necessary to code an error handling routine in every procedure you write in Visual Basic. If an error is raised in a procedure, VB will work its way back up through the call tree looking for an error handler.

```
Public Sub SubA()
On Error Goto ProcError

  ' other code
  MsgBox FuncA()

ProcExit:
  Exit Sub

ProcError:
  MsgBox Err.Description
  Resume ProcExit

End Sub
Private Function FuncA() As Date

  FuncA = CDate("hi there")

End Function
```

In this example, procedure SubA enables an error handler using the statement On Error Goto ProcError. When function FuncA is called in the MsgBox statement, the On Error Goto ProcError handler is still enabled. The CDate function in FuncA will generate error 13 (type mismatch) because CDate can't make a date from the input data. VB first looks in FuncA for an error handler. None was enabled, so the error is propogated back up the call tree to SubA. Since there is an error handler in SubA, program execution is redirected to the ProcError label in SubA. The MsgBox statement displays a description of the error and the Resume statement directs VB to continue execution at the ProcExit label.

There are some situations where VB cannot pass an error back up the call tree. This applies to Sub Main, most event procedures, and the Class_Terminate event procedure. Sub Main (if defined in the project property sheet) is the first code executed, so there is no procedure higher in the tree at application startup time. Most event procedures are also fired by Visual Basic when no other code is running so these are also at the top of the tree. Finally, the Class_Terminate event of class modules cannot raise an error because this event can also occur when no other code is executing in the application.

If an error is generated in one of these types of procedures and no error handler is enabled in the procedure, VB invokes its own default error handler, displays a message, and terminates the application. Because of this behavior, it is vital that you always code an error handler in Sub Main, all event procedures, and the Class_Terminate event for class modules.
Unlike the Class_Terminate event, the Class_Initialize event of a class module can raise an error or allow it to go untrapped. However, it is considered good programming practice to have classes

trap their own errors, deal with them if possible, and if necessary raise errors explicitly, providing a number and description defined within the class.

You can code your classes to map any error the class encounters to class-defined error numbers, but given the large number of potential errors that could occur in an application, that may not always be practical. An alternative is to have the class assign specific numbers and descriptions to errors that are specific to problems with the code or data in the class (such as a value that violates a rule for the data) but pass out standard VB errors (such as error 13 - type mismatch) as is. This allows applications using the class to explicitly handle the errors exclusive to the class with customized code, but handle standard VB errors with more generic code.

Regardless of the approach you take, you must always ensure that private data within the class is valid and that code within the class cleans up any local or module level object variables it creates. This may require you to setup an error handler that traps errors, cleans up local object variables, and then raises the same error again.

Building Error Handlers

Trapping an error using the On Error statement is only the first step in dealing with run-time errors in your code. You must also deal with the error in some way, even if the error handling code is as simple as ignoring the error (a perfectly valid approach in some situations) or displaying a message for the user.

The first step in handling an error is determining the nature of the error. This is accomplished by examining the properties of Visual Basic's Err object. The Err object includes the following properties:

Number

This is the error number that was raised.

Description

This contains a descriptive message about the error. Depending on the error, the description may or may not be useful. (Microsoft Access, for example, has the the infamous error message "There is no message for this error.")

Source

The Source property tells you what object generated the error.

HelpContext

If a help file has been defined for the component that raised the error, this property will give you the help context ID. You can use this property along with the HelpFile property to display context sensitive help for errors in your application or as a debugging aid.

HelpFile

This is the name of the help file that contains additional information about the error (if a help file has been provided).

It is important that you rely only on the error number to determine the nature of the error. While the Description and other properties may contain useful information, only the Number property is a reliable indicator of the exact error that occurred.

A common approach in coding an error handler is to build a Select Case block based on the Number property of the Err object:

```
Public Sub SubA()
On Error Goto ProcError

  ' code that raises an error
```

```
ProcExit:
  Exit Sub

ProcError:
  Select Case Err.Number
    Case X
      ' handle X
    Case Y
      ' handle Y
    Case Z
      ' handle Z
    Case Else
      ' default
      MsgBox Err.Description
      Resume ProcExit
  End Select

End Sub
```

X, Y, and Z may be literal numbers (Case 13 ' type mismatch) or, if they are available, symbolic constants representing the numbers.

If you are building a class module that will raise class-defined errors, you should provide a public enumeration in the class that defines constants for any errors raised by the class. By providing constants, code that creates objects defined by the class can use the constants instead of the literal numbers and protect itself from changes in the actual numbers.

Once you have trapped and handled the error, you need to tell Visual Basic where to continue with program execution. There are several options available when an error handling block is entered using On Error Goto label:

Resume

The Resume statement tells VB to continue execution with the line that generated the error.

Resume Next

Resume Next instructs Visual Basic to continue execution with the line following the line that generated the error. This allows you to skip the offending code.

Resume label

This allows you to redirect execution to any label within the current procedure. The label may be a location that contains special code to handle the error, an exit point that performs clean up operations, or any other point you choose.

Exit

You can use Exit Sub, Exit Function, or Exit Property to break out of the current procedure and continue execution at whatever point you were at when the procedure was called.

End

This is not recommended, but you can use the End statement to immediately terminate your application. Remember that if you use End, your application is forcibly terminated. No Unload, QueryUnload, or Terminate event procedures will be fired. This is the coding equivalent of a gunshot to the head for your application.

In addition to these statements, you can also call the Clear method of the Err object to clear the current error. This is most often used with inline error handling, as shown below:

```
Public Sub CreateFile(sFilename As String)

  On Error Resume Next
  ' the next line will raise an error if the file
  ' doesn't exist, but we don't care because the
  ' point is to kill it if it does anyway.
  Kill sFilename
  Err.Clear

  ' code to create a file

End Sub
```

This isn't a very robust example. There are many other things besides a file that doesn't exist that could cause the Kill statement to fail. The file may be read-only, there may be a network permissions error, or some other problem.

Handling Errors You Can't Handle

In most cases you can anticipate the most common errors and build code to deal with them. The error handling code might be as simple as a message to the user such as "This field requires a valid date." In some cases, however, you will encounter errors you didn't anticipate and you must find a way to deal with them.

There are two general approaches you can take to handling unanticipated errors:

Assume that the error is not fatal to the application.

Most errors will not be fatal to an application. The error may have been bad data provided by a user, a file that was not found, etc. Normally these kinds of errors can be corrected by the user and the application can continue. A default case in an error handler can simply display a message and exit the current procedure or continue.

Assume that the error is fatal and the application must be terminated.

In some cases, any error may be an application killer. This should be rare because this kind of error should be explicitly handled, if necessary by providing the user with the tools or information necessary to correct the situation. However, if a situation occurs where an unanticipated error is fatal, you must be sure to clean up after yourself before you shut down the application by unloading all forms and releasing any object variables you have created.

You should try to avoid the latter situation at all times. Displaying a message and shutting down or - worse yet - just pulling the application out from under the user will not be well received. If you must terminate an application due to some disastrous situation, be sure to provide as much information to the user as you can so that the situation can be resolved. An even better option is to code your error handlers to call code that corrects severe problems. For example, if you are designing a database application and encounter a corrupted database file, the error handling code could give the user the option of attempting to repair the damaged file. If a file cannot be found where it should be, write code to either look for it or give the user a file open dialog box so they can tell you where it went.

Raising Your Own Errors

There may be times when you need to generate errors in your code. This happens most often in class modules, but you can raise an error anywhere in a Visual Basic application. You raise an error by calling the Raise method of the Err object. Not surprisingly, the parameters of the Raise method are the same as the properties of the Err object: Number, Description, Source, HelpContext, and HelpFile. The values you provide for these parameters are available to error handling code that deals with the error you generate.

When you raise an error, you should make the information you provide via the Err object as informative as possible so that error handling code that deals with the error has the best possible opportunity to handle it.

Number

You can raise any of the standard VB error numbers or provide your own number. If you are raising application-defined errors, you need to add the intrinsic constant vbObjectError to the number you raise so that your number does not conflict with built in error numbers.

Description

Make the description as informative as possible. If invalid data is provided, it may be helpful to make that data part of the error message.

Source

The Source provides the name of the object that generated the error. For example, if a Jet Database object raises an error, the Source property is "DAO.Database".

HelpContext

If you provide a help file with the component or application, use the HelpContext parameter to provide a context ID. This can then be passed on to the MsgBox statement so that context sensitive help about the error is available.

HelpFile

This is the name of the help file and is used in conjunction with the HelpContext parameter.

The following example is a hypothetical property procedure for a class module:

```
' in the declarations section
Private mDate As Date

Public Enum MyClassErrors
  errInvalidDate
  ' other errors
End Enum

' a property procedure
Public Property Let Date(vntDate As Variant)
' a variant is used to allow the property
' procedure to attempt to convert the value
' provided to a date

  If IsDate(vntDate) Then
    mDate = CDate(vntDate)
  Else
    ' invalid data
    Err.Raise _
        Number:=errInvalidDate, _
        Description:=CStr(vntDate) & " is not a valid date.", _
        Source:="Foo.MyClass"
        ' help context and file go here if a help file is available
  End If

End Property
```

In this example, the property procedure tests for a valid date using the IsDate function. If the data provided is not a date, an error is raised using the constant from the error enumeration in the

declarations section of the class module and a description that includes the bad data and the nature of the error.
Summary
Handling run-time errors is something all applications must do if they are to be robust and reliable.

The key points for error handling are:
There are two steps to handling run-time errors:
Trap the error by enabling an error handler using the On Error statement.
Handle the error by examining the properties of the Err object and writing code to deal with the problem.
Error handlers can be dedicated blocks of code enabled by using On Error Goto label or can be inline handlers enabled by using On Error Resume Next.
You can raise your own errors by calling the Raise method of the Err object.
Do your best to handle run-time errors rather than just inform the user of the problem, but if you can't do anything but display a message, make it as informative as possible.
Avoid terminating the application if at all possible.
The Err object was introduced in Visual Basic 4.0. For backward compatibility, VB continues to support the Err and Error statements and functions. Any new code should be using the Err object and legacy code should be converted to use the Err object.
The following table contains a list of the trappable error codes you may encounter when you use the Err function.

Error code	Error message
3	Return without GoSub
5	Invalid procedure call
6	Overflow
7	Out of memory
9	Subscript out of range
10	Duplicate definition (versions 5.0 and 7.0)
10	This array is fixed or temporarily locked (version97)
11	Division by zero
13	Type mismatch
14	Out of string space
16	String formula too complex (versions 5.0 and 7.0)
16	Expression too complex (version 97)
17	Can't perform requested operation
18	User interrupt occurred
20	Resume without error
28	Out of stack space
35	Sub or function not defined (versions 5.0 and 7.0)
35	Sub, function, or property not defined (version 97)
47	Too many DLL application clients (version 97)
48	Error in loading DLL
49	Bad DLL calling convention
51	Internal error
52	Bad file name or number
53	File not found
54	Bad file mode

55	File already open
57	Device I/O error
58	File already exists
59	Bad record length
61	Disk full
62	Input past end of line
63	Bad record number
67	Too many files
68	Device unavailable
70	Permission denied
71	Disk not ready
74	Can't rename with different drive
75	Path/File access error
76	Path not found
91	Object variable not set (versions 5.0 and 7.0)
91	Object variable or With block variable not set (version 97)
92	For Loop not initialized
93	Invalid pattern string
94	Invalid use of Null
95	User-defined error (versions 5.0 and 7.0 only)
298	System DLL could not be loaded (version 97)
320	Can't use character device names in specified file names (version 97)
321	Invalid file format (version 97)
322	Can't create necessary temporary file (version 97)
323	Can't load module; invalid format (versions 5.0 and 7.0)
325	Invalid format in resource file (version 97)
327	Data value named was not found (version 97)
328	Illegal parameter; can't write arrays (version 97)
335	Could not access system registry (version 97)
336	ActiveX component not correctly registered (version 97)
337	ActiveX component not found (version 97)
338	ActiveX component did not correctly run (version 97)
360	Object already loaded (version 97)
361	Can't load or unload this object (version 97)
363	Specified ActiveX control not found (version 97)
364	Object was unloaded (version 97)
365	Unable to unload within this context (version 97)
368	The specified file is out of date. This program requires a newer version (version 97)
371	The specified object can't be used as an owner form for Show (version 97)
380	Invalid property value (version 97)
381	Invalid property-array index (version 97)
382	Property Set can't be executed at run time (version 97)
383	Property Set can't be used with a read-only property (version 97)
385	Need property-array index (version 97)
387	Property Set not permitted (version 97)

393	Property Get can't be executed at run time (version 97)
394	Property Get can't be executed on write-only property (version 97)
400	Form already displayed; can't show modally (version 97)
402	Code must close topmost modal form first (version 97)
419	Permission to use object denied (version 97)
422	Property not found (version 97)
423	Property or method not found
424	Object required
425	Invalid object use (version 97)
429	ActiveX component can't create object or return reference to this object (version 97)
430	Class doesn't support OLE Automation
430	Class doesn't support Automation (version 97)
432	File name or class name not found during Automation operation (version 97)
438	Object doesn't support this property or method
440	OLE Automation error
440	Automation error (version 97)
442	Connection to type library or object library for remote process has been lost (version 97)
443	Automation object doesn't have a default value (version 97)
445	Object doesn't support this action
446	Object doesn't support named arguments
447	Object doesn't support current locale settings
448	Named argument not found
449	Argument not optional
449	Argument not optional or invalid property assignment (version 97)
450	Wrong number of arguments
450	Wrong number of arguments or invalid property assignment (version 97)
451	Object not a collection
452	Invalid ordinal
453	Specified DLL function not found
454	Code resource not found
455	Code resource lock error
457	This key is already associated with an element of this collection (version 97)
458	Variable uses a type not supported in Visual Basic (version 97)
459	This component doesn't support events (version 97)
460	Invalid clipboard format (version 97)
461	Specified format doesn't match format of data (version 97)
480	Can't create AutoRedraw image (version 97)
481	Invalid picture (version 97)
482	Printer error (version 97)

483	Printer driver does not support specified property (version 97)
484	Problem getting printer information from the system. Make sure the printer is set up correctly (version 97)
485	Invalid picture type (version 97)
486	Can't print form image to this type of printer (version 97)
735	Can't save file to Temp directory (version 97)
744	Search text not found (version 97)
746	Replacements too long (version 97)
1000	Classname does not have propertyname property (versions 5.0 and 7.0)
1001	Classname does not have methodname method (versions 5.0 and 7.0)
1002	Missing required argument argumentname (versions 5.0 and 7.0)
1003	Invalid number of arguments (versions 5.0 and 7.0)
1004	Methodname method of classname class failed (versions 5.0 and 7.0)
1005	Unable to set the propertyname property of the classname class (versions 5.0 and 7.0)
1006	Unable to get the propertyname property of the classname class (versions 5.0 and 7.0)
31001	Out of memory (version 97)
31004	No object (version 97)
31018	Class is not set (version 97)
31027	Unable to activate object (version 97)
31032	Unable to create embedded object (version 97)
31036	Error saving to file (version 97)
31037	Error loading from file (version 97)

Option Explicit Statement

Used at module level to force explicit declaration of all variables in that module.

Syntax
Option Explicit

If used, the Option Explicit statement must appear in a module before any procedures.

When Option Explicit appears in a module, you must explicitly declare all variables using the Dim, Private, Public, ReDim, or Static statements. If you attempt to use an undeclared variable name, an error occurs at compile time.

If you don't use the Option Explicit statement, all undeclared variables are of Variant type unless the default type is otherwise specified with a Deftype statement.

Note   Use Option Explicit to avoid incorrectly typing the name of an existing variable or to avoid confusion in code where the scope of the variable is not clear.

SmartVB6 - useful Visual Basic 6.0 development tools.

OLSOFT has released SmartVB6 - useful Visual Basic 6.0 development tools to build fast and robust VB6 code. It includes intuitively understandable modules that help in programming such as powerful project explorer, snippets that is a library of useful codes, a number of builders: procedures, classes, properties, structures, collections, enum and error handlers.

Project Explorer is intended to expand standard Visual Basic Explorer functionality. Easy to use and navigate diagram-tree helps to manage objects and elements.

Procedure Builder is intended to generate text of the Sub or Function. Collections Builder helps to include a new collection class into the user's project based on the existing class. Enum Builder is used to create new Enum type in the project. Error Handler Manager helps to insert, edit or remove error handler codes and schemes.

Code Snippets module is used to work with codes library. Snippets is the library of useful codes. Snippet Manager, Insert Snippet and Store to Snippets modules are used to work with codes library, to insert a fragment from snippets to the project or to record interesting fragment to the library.

Settings form is also included into the program to easily adjust necessary parameters such as author, procedure, property, class, enum, snippet manager, exceptions, control type and explorer.

SmartVB6 was created to easily build fast and robust VB6 code.

What is the difference between Two Tier Architecture and Three Tier Architecture?

In Two Tier Architecture or Client/Server Architecture two layers like Client and Server is involved. The Client sends request to Server and the Server responds to the request by fetching the data from it. The problem with the Two Tier Architecture is the server cannot respond to multiple requests at the same time which causes data integrity issues.

The Client/Server Testing involves testing the Two Tier Architecture of user interface in the front end and database as backend with dependencies on Client, Hardware and Servers.

In Three Tier Architecture or Multi Tier Architecture three layers like Client, Server and Database are involved. In this the Client sends a request to Server, where the Server sends the request to Database for data, based on that request the Database sends back the data to Server and from Server the data is forwarded to Client.
The Web Application Testing involves testing the Three Tier Architecture including the User interface, Functionality, Performance, Compatibility, Security and Database testing.
2-tier means 1) Design layer/Presentation Layer 2) Data layer
3-tier means 1) Design layer/Presentation Layer 2) Business layer or Logic layer 3) Data layer

Key Differences Between VB and VB Script

You didn't run out to the computer store and buy a copy of VBScript. You didn't install a VBScript disk on your computer, either. All you did was install the Internet Explorer browser, which supports VBScript, on your computer--just like millions of other folks. Everyone of them has the VBScript engine on their computer, and everyone of them has the ability to create Web pages with VBScript.

So where's the integrated development environment that you're used to using in Visual Basic? Keep looking, because there isn't one. All you have is your favorite text editor, the ActiveX Control Pad, and a Web browser. That in itself is the single largest difference between Visual Basic and VBScript. It leads to some specific differences, too. Here's what they are:
Debugging VBScript doesn't have a debugger like Visual Basic. You'll resort to using lots of message boxes, instead.

Event-handlers You don't have an editor in which you select an object and event to edit an event-procedure. You have to name event-procedures in your scripts so that the scripting engine can find the appropriate handler when an object fires an event.

Forms VBScript doesn't have a forms editor. It doesn't need one, because you can't display forms anyway. You put forms and controls on the Web page, instead. You can use the ActiveX Control pad to insert all those nasty <OBJECT> tags in your Web page, however.

You don't compile a VBScript program into an EXE file like you do with a Visual Basic program. You distribute your scripts as plain, old text embedded in HTML files. Everyone and his uncle can read your scripts. The script engine interprets this text into intermediate code when it loads the Web page. It also creates a symbol table so that it can quickly look up things such as event-procedures and variable names. The scripting engine uses the ActiveX Scripting technology to interact with the browser.

[BEG]NOTE:[END] You'll find a plethora of nit-picky differences between Visual Basic and VBScript, too. You have to use the value property to query an objects value, for example. Thus, instead of reading a text box's value using form.text, you have to read it using form.text.value. These subtle differences are too numerous to document in this appendix. Go to Microsoft's Web site (www.microsoft.com) and their knowledge base for further explanation of these differences.

Another significant difference between Visual Basic and VBScript is the keywords that Microsoft omitted from VBScript. You'll learn more about the keywords included in VBScript in the next section. You'll learn about the keywords that Microsoft omitted from VBScript in "Visual Basic Keywords Omitted from VBScript," later in this appendix.

Key Differences Between VB and VBScript
Visual Basic Keywords Included in VBScript
Visual Basic Keywords Omitted from VBScript

DIFFERENCE BETWEEN VBSCRIPT & ASP

VBScript is one of the sripting languages you can choose for ASP.
It's the most frequently used one. others are Perl or JScript.
ASP can best be thought of as a framework with some (6) objects.

VBSript is the default scripting language for IIS. Only the VBscript and Jscript-engines are shipped with IIS so you best choose one of these languages to write your asp code in. (Unless you're planning on running asp on an Apacht, then choose Perl).

Using Crystal Reports

This tutorial shows you how to use Crystal Reports 4.6, which is the version of Crystal Reports that shipped with Visual Studio / Visual Basic 6.0. Crystal Reports provides a relatively easy way to incorporate reporting into your application. The basis of a report is a database query.

It should be pointed out that this version of Crystal Reports is quite old, and that this version and the past versions of Crystal Reports that have shipped with VB are generally "watered down" versions that do not have all the bells and whistles of the standalone product that is sold by Seagate Software. Furthermore, CR 4.6 will not work with MS-Access databases higher than Access 97. That said, what you learn about designing reports with CR 4.6 will help when working with later versions. Regarding the MS-Access issue, later versions of Access (2000, 2002, etc.) can read and save in Access 97 format.

Installing Crystal Reports 4.6

Crystal Reports 4.6 is bundled with Visual Studio/Visual Basic 6.0, but it is not installed automatically. To install it manually, locate the CRYSREPT folder on your installation CD â€“ for Visual Studio 6.0, the path is COMMON\TOOLS\VB\CRYSREPT on the third CD. In that folder, double-click the file CRYSTL32.EXE. You will be asked if you want to install Crystal Reports. Respond Yes. It will then tell you where it is going to install CR; you can override the location if desired. Following that, CR will be installed, and a few moments later you should get a message indicating that installation was successful.

The Sample Database

The sample database used for this tutorial (as well as others to follow) is an Access 97 format database named EMPLOYEE.MDB.

EMPLOYEE.MDB contains three tables: EmpMast, DeptMast, and JobMast. The tables are structured as follows:

Designing the Reports

Two reports will be developed from this database: "Annual Salary Expenses by Department" and "Annual Salary Expenses by Job".

For the Annual Salary Expenses by Department report, you want to show various fields from the employee database tables grouped and subtotaled by department. You also want to show a grand total at the end of the report. A sketch of the design might look something like the following:

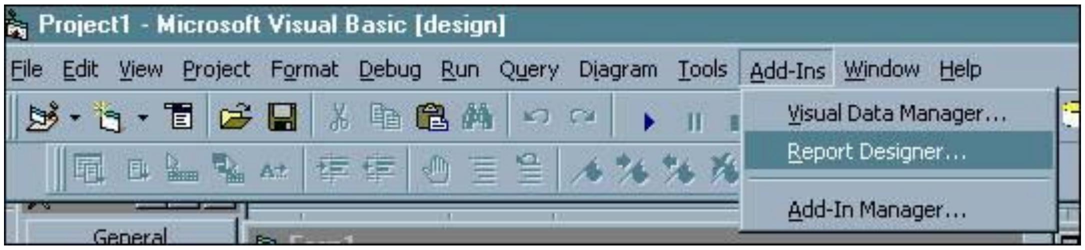
```
Annual Salary Expenses by Department

JOB WKLY HRLY

EMP # EMP NAME # JOB TITLE HIRE DATE HOURS RATE ANN SALARY

----- -------- --- --------- --------- ----- ---- ----------

DEPT XXXX XXXXXXXXXXXXXXXX

XXX XXXXXXXXXXXXXXXX XXX XXXXXXXXXX XX/XX/XX XX.XX XX.XX
$XXX,XXX.XX

XXX XXXXXXXXXXXXXXXX XXX XXXXXXXXXX XX/XX/XX XX.XX XX.XX
$XXX,XXX.XX

DEPT XXXX XXXXXXXXXXXXXXXXXXXXXXXX TOTALS: $XXX,XXX.XX

.

.

GRAND TOTALS: $XXX,XXX.XX
```

The design of the Annual Salary Expenses by Job report is similar, except that you want to show various fields from the employee database tables grouped and subtotaled by job. You might sketch the design as follows:

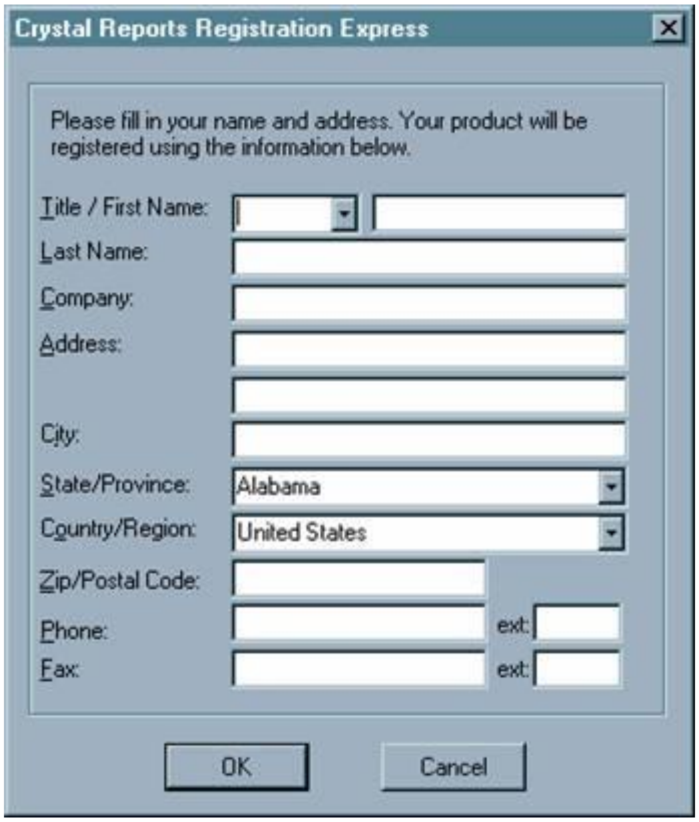
```
Annual Salary Expenses by Job

DEPT WKLY HRLY

EMP # EMP NAME # DEPT NAME HIRE DATE HOURS RATE ANN SALARY

----- -------- --- --------- --------- ----- ---- ----------


JOB XXX XXXXXXXXXXXXXXXX



XXX XXXXXXXXXXXXXXXX XXXX XXXXXXXXXX XX/XX/XX XX.XX XX.XX
$XXX,XXX.XX

XXX XXXXXXXXXXXXXXXX XXXX XXXXXXXXXX XX/XX/XX XX.XX XX.XX
$XXX,XXX.XX


JOB XXX XXXXXXXXXXXXXXXXXXXXXXXX TOTALS: $XXX,XXX.XX



.

.

.



GRAND TOTALS: $XXX,XXX.XX
```

This tutorial will start off by showing the step-by-step process for designing the first report, "Annual Salary Expenses by Department". Once that is done, it will be a simple matter to copy that report and modify it to create the Annual Salary Expenses by Job report.

With the report designs in mind, open Crystal Reports (Report Designer) from the VB Add-Ins menu:


Project1 - Microsoft Visual Basic [design]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
Visual Data Manager...
Report Designer...
Add-In Manager...
General

A registration form appears. Click the Cancel button.

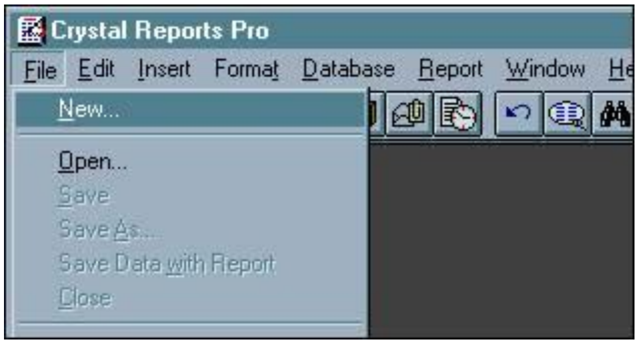

Crystal Reports Registration Express

Please fill in your name and address. Your product will be registered using the information below.

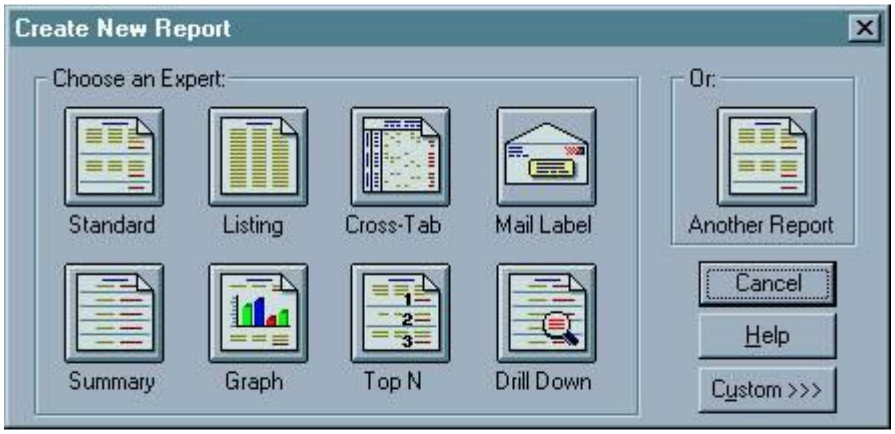
Title / First Name:
Last Name:
Company:
Address:
City:
State/Province: Alabama
Country/Region: United States
Zip/Postal Code:
Phone: ext:
Fax: ext:

OK Cancel

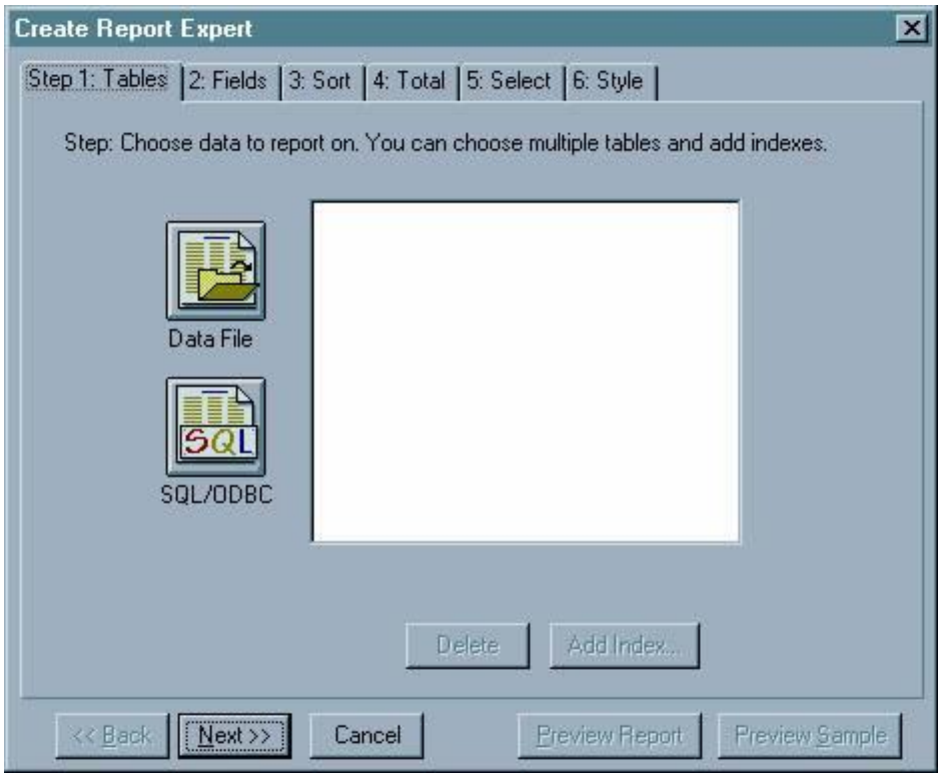
From the Crystal Reports File menu, select New:



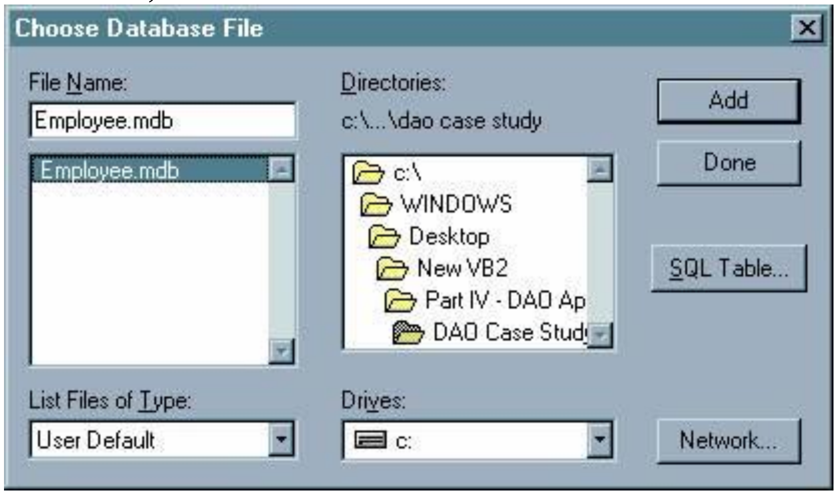
The Create New Report dialog box appears. Click the Standard button.



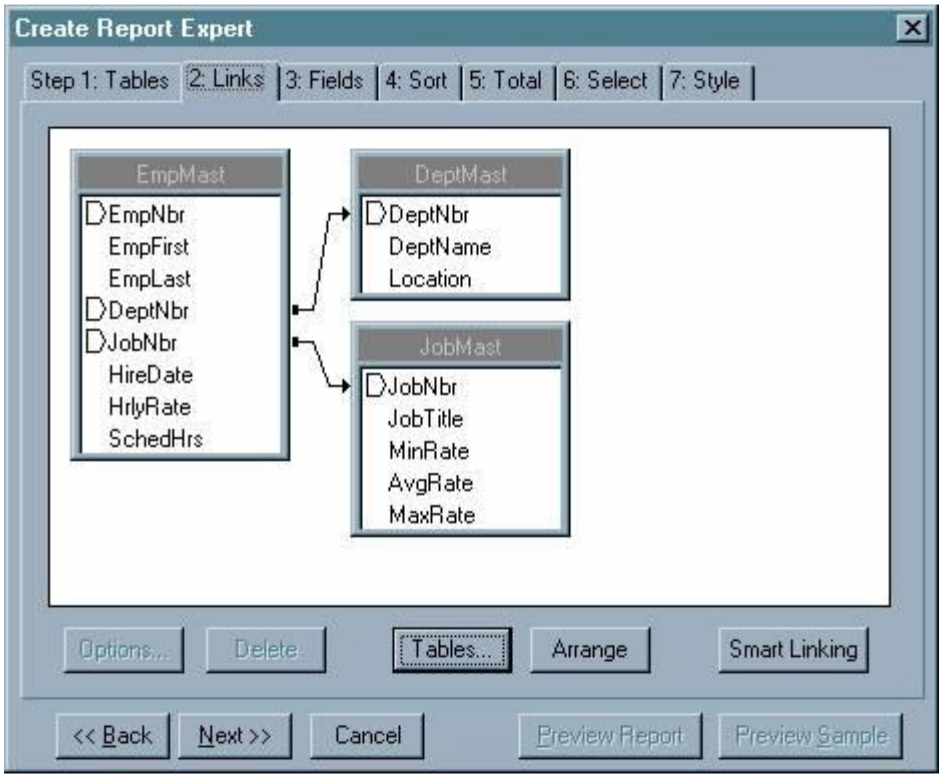
The Create Report Expert dialog box appears with the Step 1: Tables tab open. Click the Data File button:



The Choose Database File dialog box appears. Navigate to the directory where your database file resides, then click the name of the database file so that it appears under "File Name:". Click the Add button, then click the Done button.

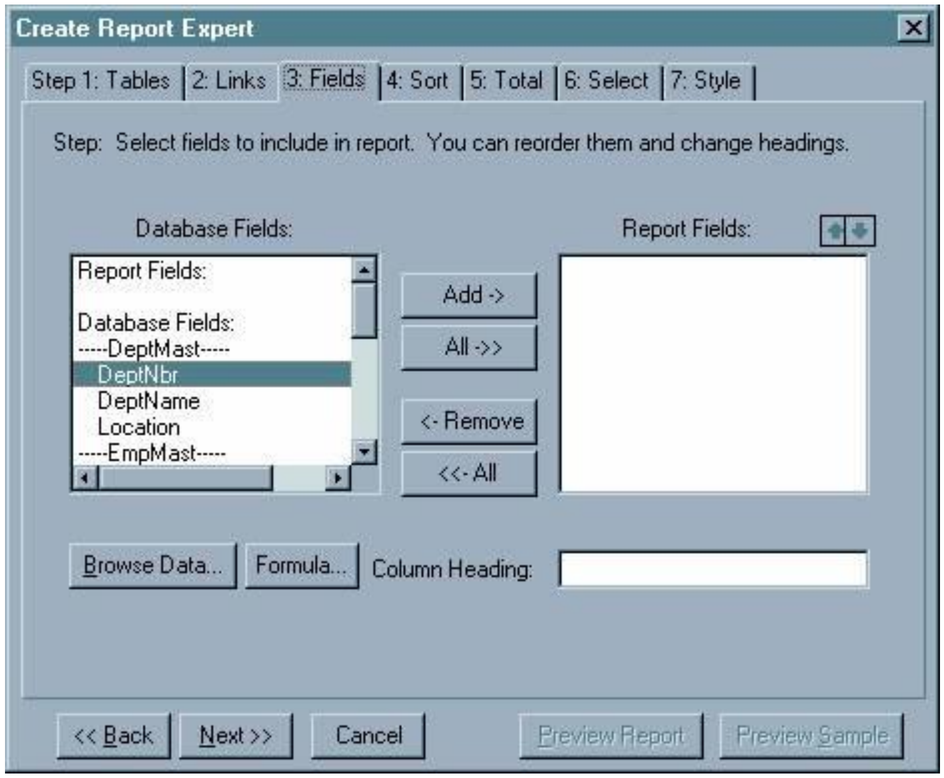


The 2: Links tab then appears, showing you a diagram similar to that of Access' Relationships diagram.



Move on by clicking the 3: Fields tab.

The "3: Fields" tab initially looks like this:



Using the "Add ->" button, select the desired fields from the "Database Fields" listbox so that they appear in the "Report Fields" listbox. Select the fields based on the initial design. For fields that are involved in a primary key â€“ foreign key relationship, only select one of those fields from either table (for example, select the DeptNbr field from either the DeptMast table OR the EmpMast table, but not both).

Select the following Database Fields:

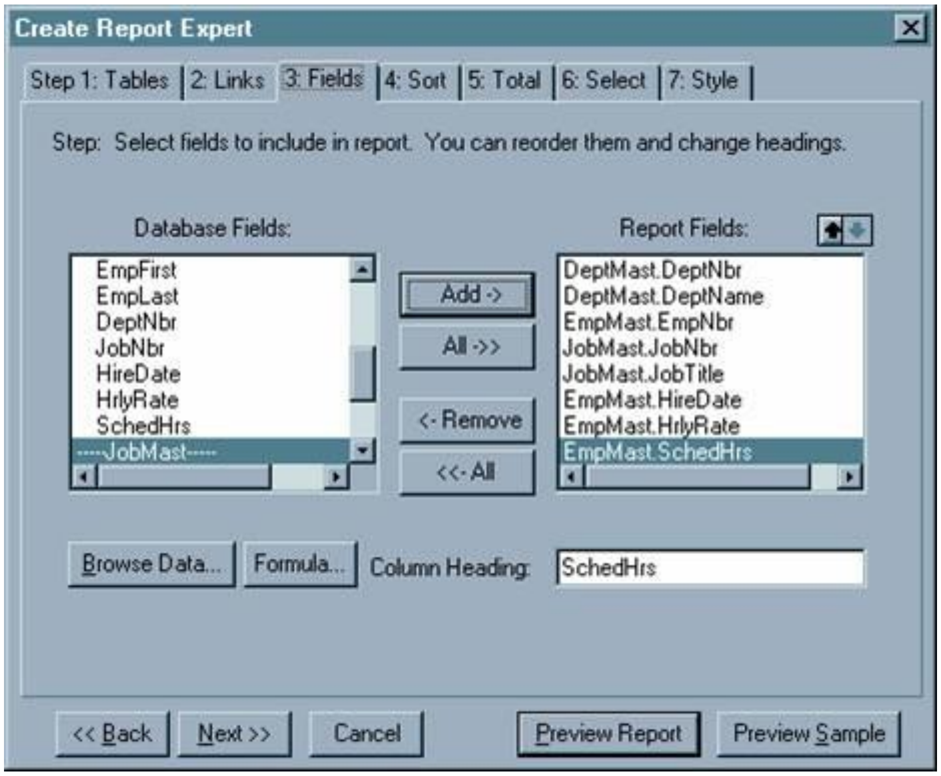
Select DeptNbr and DeptName from the DeptMast table.

Select EmpNbr From the EmpMast table.

Skip down to the JobMast table and select the JobNbr and JobTitle.

Jump back up to the EmpMast table and select the HireDate, HrlyRate, and SchedHrs.

Your screen should look like this:



Step: Select fields to include in report. You can reorder them and change headings.

Database Fields: EmpFirst, EmpLast, DeptNbr, JobNbr, HireDate, HrlyRate, SchedHrs, ----JobMast-----

Report Fields: DeptMast.DeptNbr, DeptMast.DeptName, EmpMast.EmpNbr, JobMast.JobNbr, JobMast.JobTitle, EmpMast.HireDate, EmpMast.HrlyRate, EmpMast.SchedHrs

Column Heading: SchedHrs

Still sticking with Tab 3, you can specify column headings by selecting each of the Report Fields in turn, and giving them a heading by typing the desired text in the "Column Heading:" textbox (by default, the column heading is the same as the field name).

Specify the column headings as follows:

Report Field
Column Heading

DeptMast.DeptNbr
Dept #

DeptMast.DeptName
Dept Name

EmpMast.EmpNbr
Emp #

JobMast.JobNbr
Job #

JobMast.JobTitle

Job Title

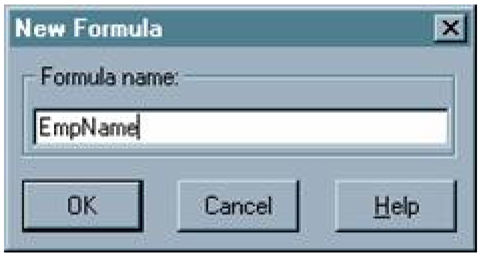
EmpMast.HireDate
Hire Date

EmpMast.HrlyRate
Hrly Rate

EmpMast.SchedHrs
Wkly Hours

We're still not done with Tab 3. We need two computed columns, one for the employee name (which will be a concatenation of the EmpLast and EmpFirst fields) and one for the annual salary (which will be the employee's hourly rate multiplied by their weekly hours multiplied by 52).

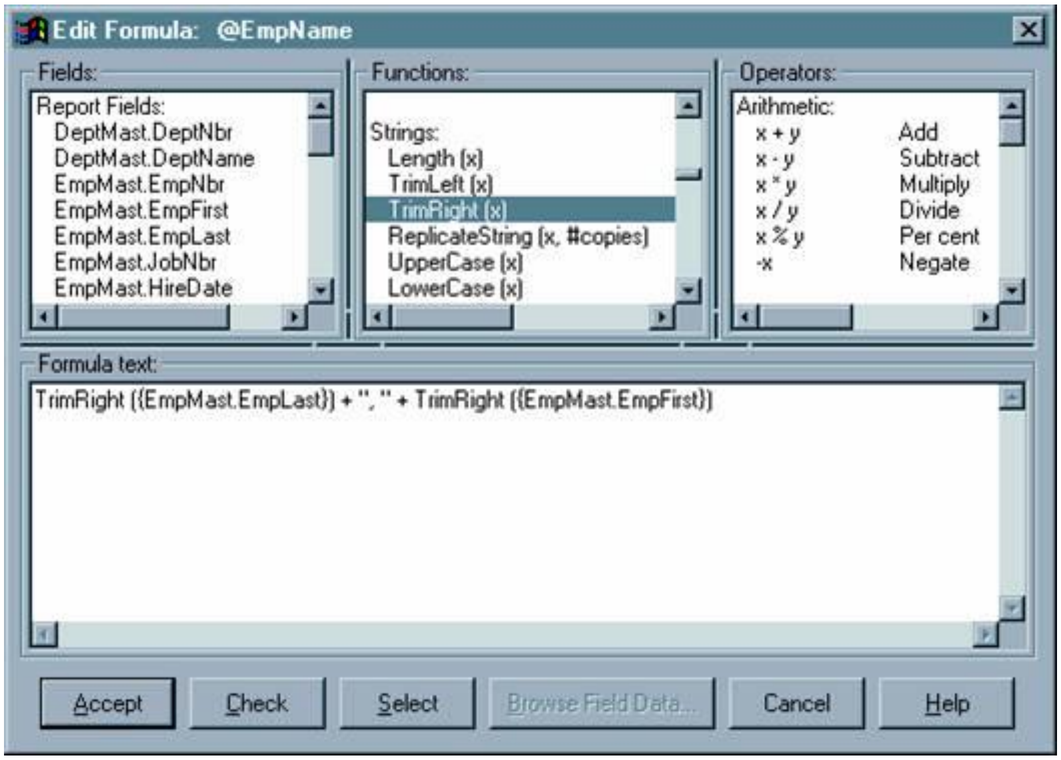
Click the Formula button. The "New Formula" dialog box appears. Type EmpName in the textbox and click OK



EmpName

The "Edit Formula" dialog box appears. In the "Formula text" area, type:

TrimRight ({EmpMast.EmpLast}) + ", " + TrimRight ({EmpMast.EmpFirst})



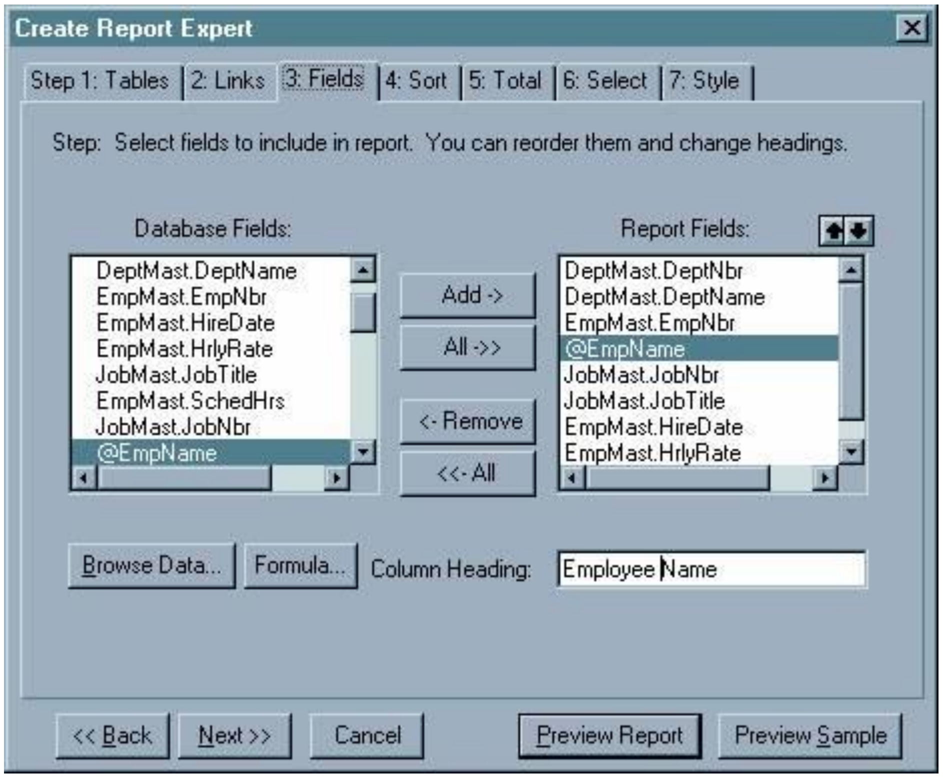
```
TrimRight ({EmpMast.EmpLast}) + ", " + TrimRight ({EmpMast.EmpFirst})
```

Your screen should look like this:

Note: Crystal Reports has its own formula syntax, which differs from the syntax of VB and Access expressions. You can scroll the "Fields", "Functions", and "Operators" listboxes above the Formula text entry area to see what's available. Also, instead of typing everything directly into the text entry area, you can double-click on a listbox selection and the text of that selection will appear in the Formula text box.

Click the Check button. If you entered the formula correctly, the message "No errors found" will pop up. Passing that, click the Accept button.

The formula will then appear in the Database Fields listbox (as "@EmpName"). With @EmpName highlighted, click the "Add->" button to add it to the Report Fields list. In the Report Fields list, drag and drop "@EmpName" so that it appears under "EmpMast.EmpNbr". Give @EmpName a column heading of "Employee Name".



Step: Select fields to include in report. You can reorder them and change headings.

Database Fields:

- DeptMast.DeptName
- EmpMast.EmpNbr
- EmpMast.HireDate
- EmpMast.HrlyRate
- JobMast.JobTitle
- EmpMast.SchedHrs
- JobMast.JobNbr
- @EmpName

Report Fields:

- DeptMast.DeptNbr
- DeptMast.DeptName
- EmpMast.EmpNbr
- @EmpName
- JobMast.JobNbr
- JobMast.JobTitle
- EmpMast.HireDate
- EmpMast.HrlyRate

Column Heading: Employee Name

Now we must create the annual salary formula. To do so, follow these steps:

Â· Make sure anything OTHER than @EmpName is selected in the Database Fields listbox.

Â· Click the Formula button.

Â· In the "New Formula" dialog box, type "AnnSal" and click OK.

Â· In the "Edit Formula" dialog box, type

{EmpMast.HrlyRate} * {EmpMast.SchedHrs} * 52

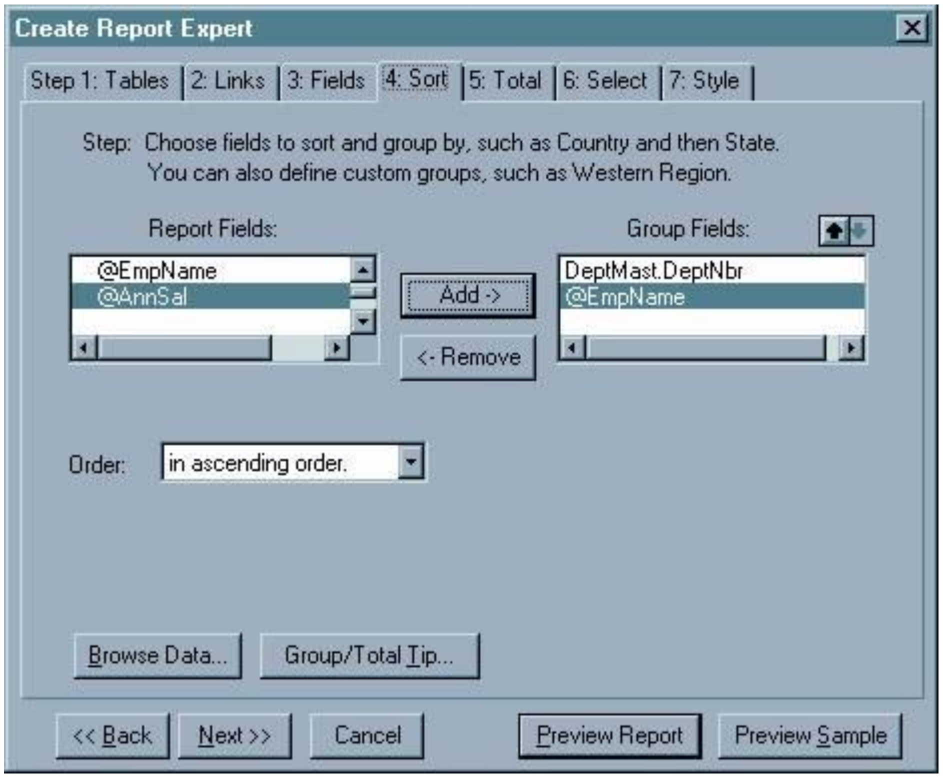
click Check, then Accept.

Â· Use the "Add->" button to bring @AnnSal over from the Database Fields list to the Report Fields list.

Â· In the Report Fields listbox, drag and drop the @AnnSal formula so that it is the last field in the list.

Â· Give @AnnSal a column heading of "Ann Salary".

Click the 4: Sort tab. Select DeptMast.DeptNbr from the "Report Fields" list and click the "Add->" button. DeptMast.DeptNbr then appears in the "Group Fields" list. Repeat this process for @EmpName. Your screen should look like this:



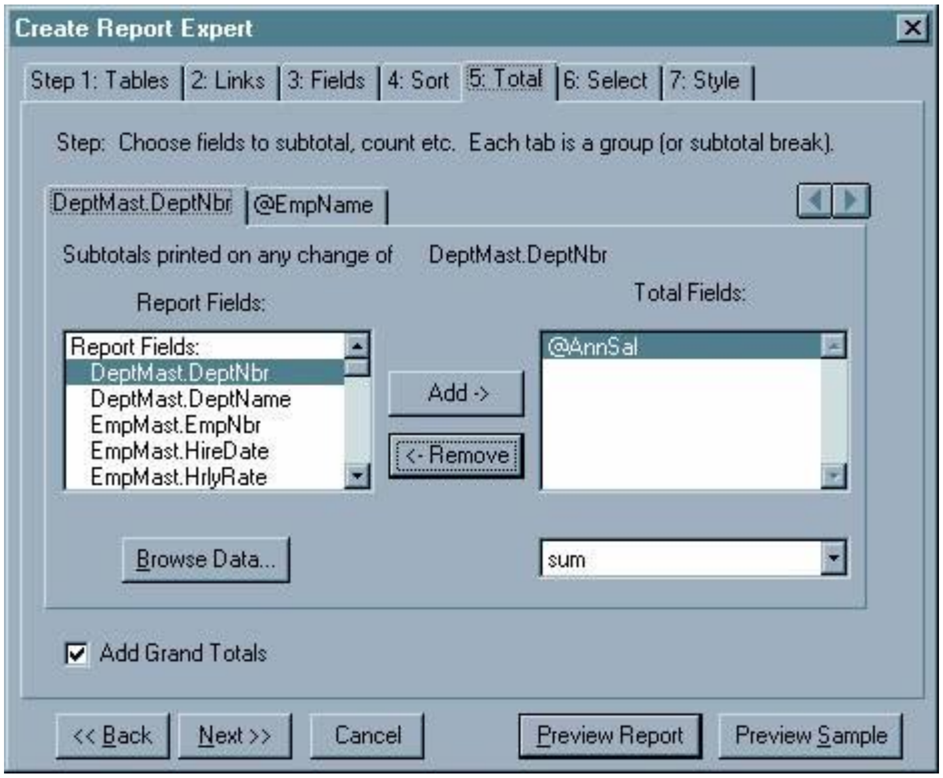
Step: Choose fields to sort and group by, such as Country and then State.
You can also define custom groups, such as Western Region.

Report Fields: @EmpName, @AnnSal

Group Fields: DeptMast.DeptNbr, @EmpName

Order: in ascending order.

Click the 5: Total tab. Within this Total tab, an inner tabbed dialog appears, with one tab for each field selected in the sort. On the "DeptMast.DeptNbr" tab, remove all items except "@AnnSal" from the Total Fields list, as shown below. What we are saying is that we want to print a subtotal for the annual salary every time there is a change, or break, in the department number.



Step: Choose fields to subtotal, count etc. Each tab is a group (or subtotal break).

Subtotals printed on any change of DeptMast.DeptNbr

Report Fields:

Report Fields:
DeptMast.DeptNbr
DeptMast.DeptName
EmpMast.EmpNbr
EmpMast.HireDate
EmpMast.HrlyRate

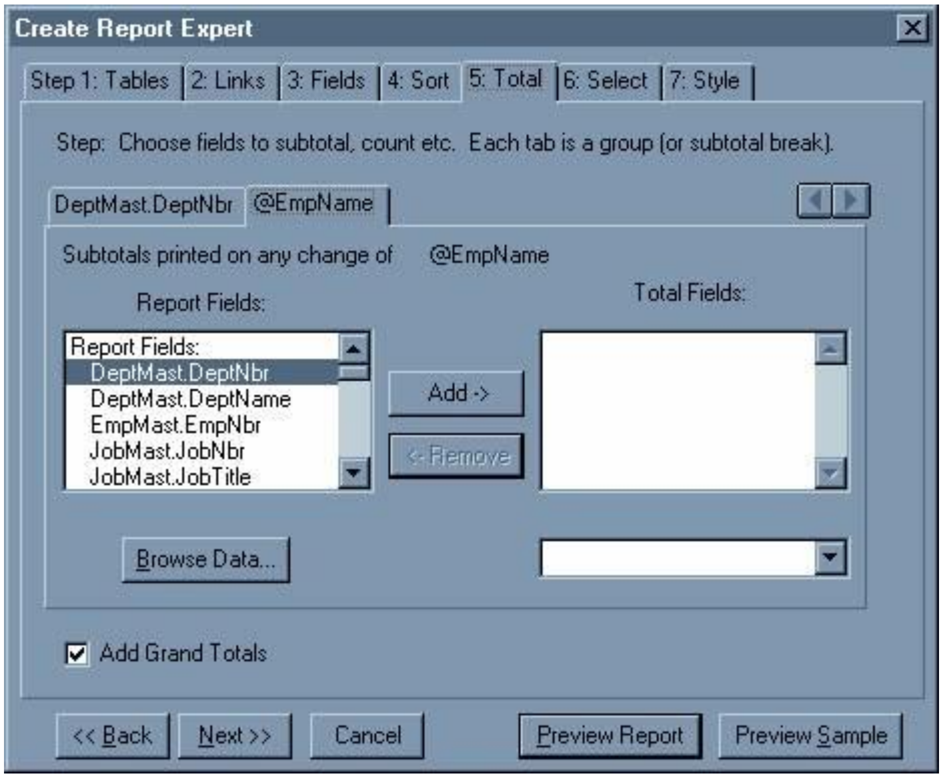
Total Fields:

@AnnSal

sum

☑ Add Grand Totals

Still in the "5: Total" tab, click the "@EmpName" tab and remove all items from the Total Fields list, as shown below. (We don't want to print subtotals after every employee name.)



Step: Choose fields to subtotal, count etc. Each tab is a group (or subtotal break).

Subtotals printed on any change of @EmpName

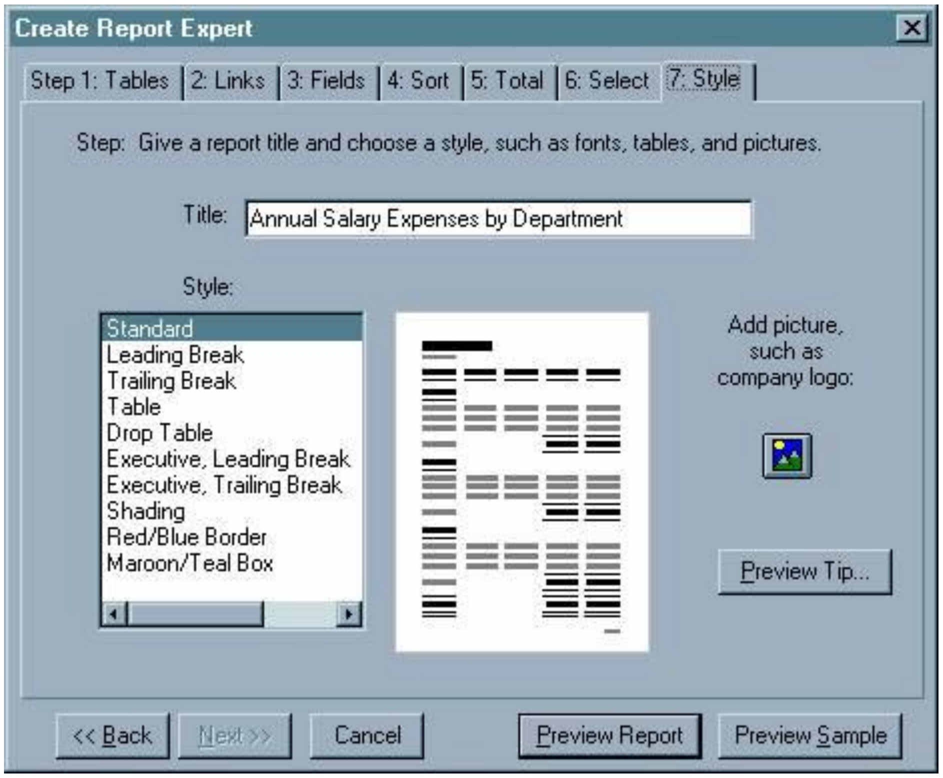
Report Fields:

Report Fields:
DeptMast.DeptNbr
DeptMast.DeptName
EmpMast.EmpNbr
JobMast.JobNbr
JobMast.JobTitle

Total Fields:

☑ Add Grand Totals

We don't need to do anything in tab 6, so click the 7: Style tab. For the title, type "Annual Salary Expenses by Department".

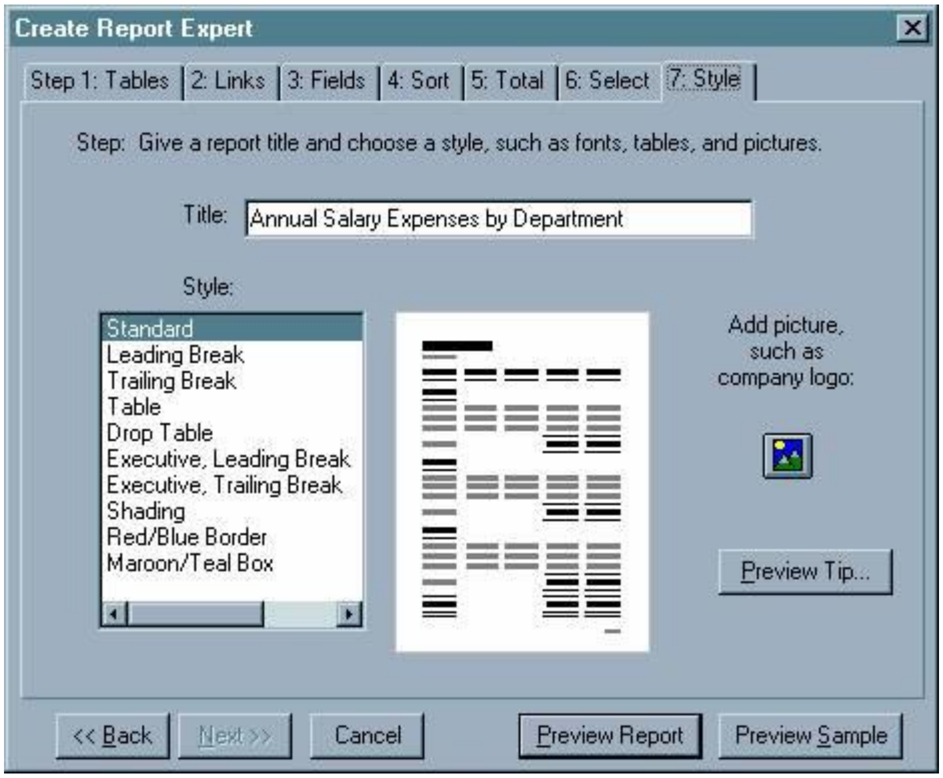


Step: Give a report title and choose a style, such as fonts, tables, and pictures.

Title: Annual Salary Expenses by Department

Style: Standard, Leading Break, Trailing Break, Table, Drop Table, Executive, Leading Break, Executive, Trailing Break, Shading, Red/Blue Border, Maroon/Teal Box

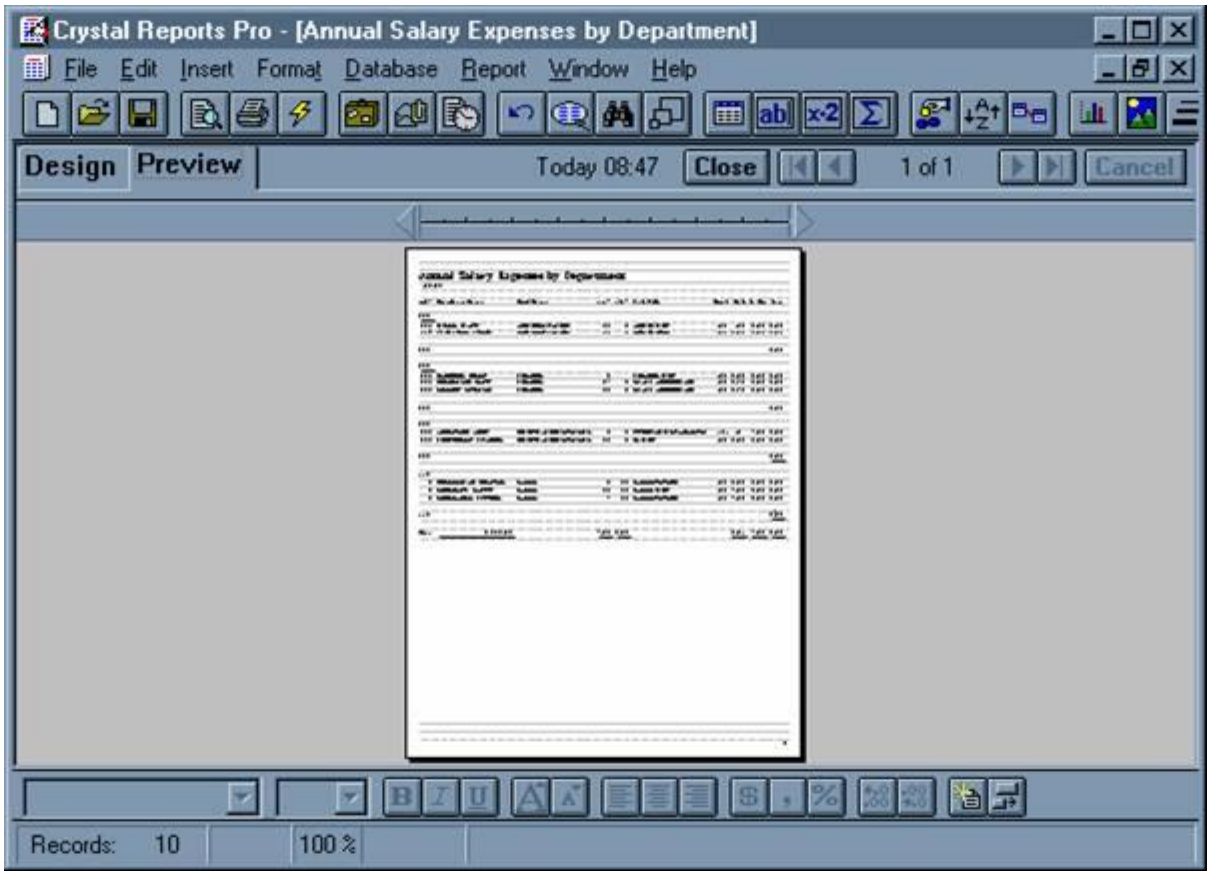
Click the Preview Report button. At this time, the "Create Report Expert" is finished and you can't go back to it, but you can make any desired changes in the Crystal Reports interface. Following is the screen that is initially displayed after you click the Preview Report button from Step 7 of the Expert:



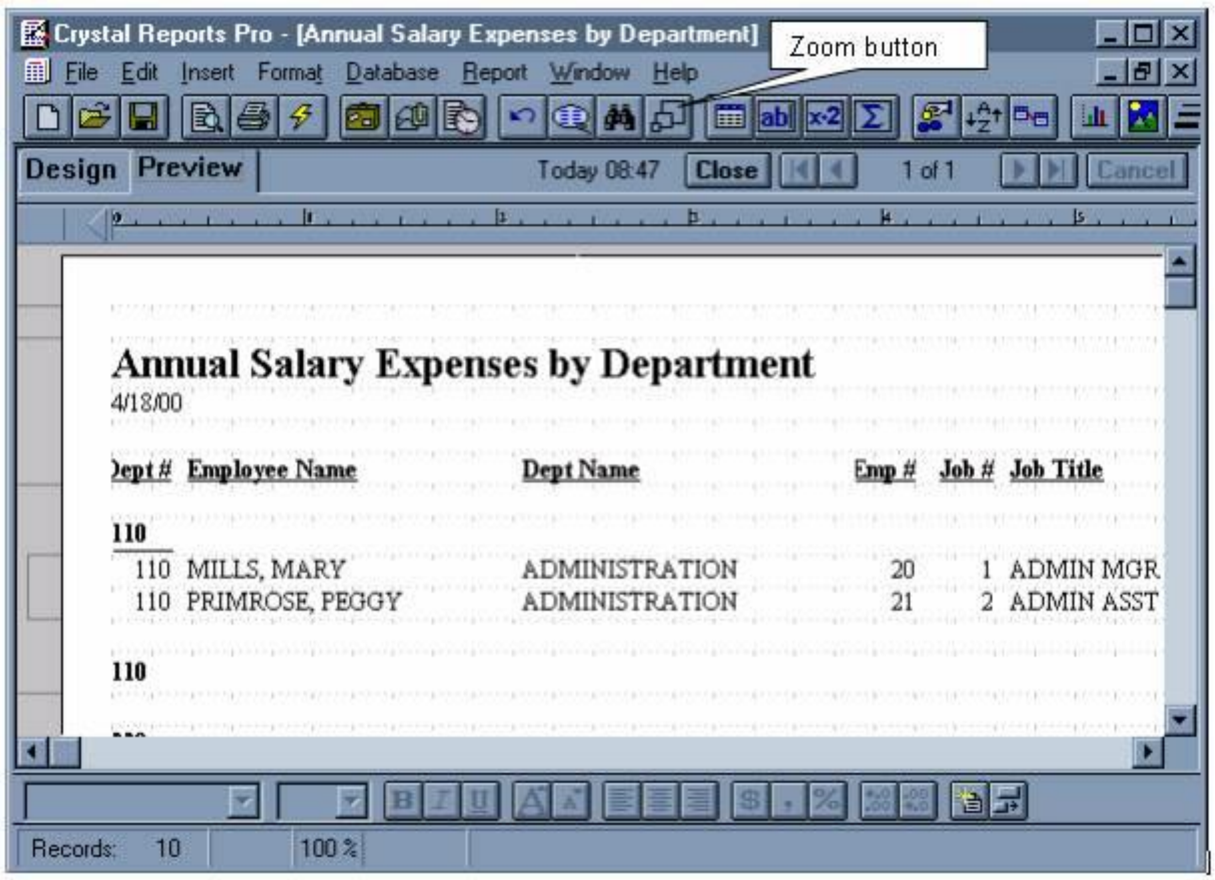
Step: Give a report title and choose a style, such as fonts, tables, and pictures.

Title: Annual Salary Expenses by Department

Style:
- Standard
- Leading Break
- Trailing Break
- Table
- Drop Table
- Executive, Leading Break
- Executive, Trailing Break
- Shading
- Red/Blue Border
- Maroon/Teal Box



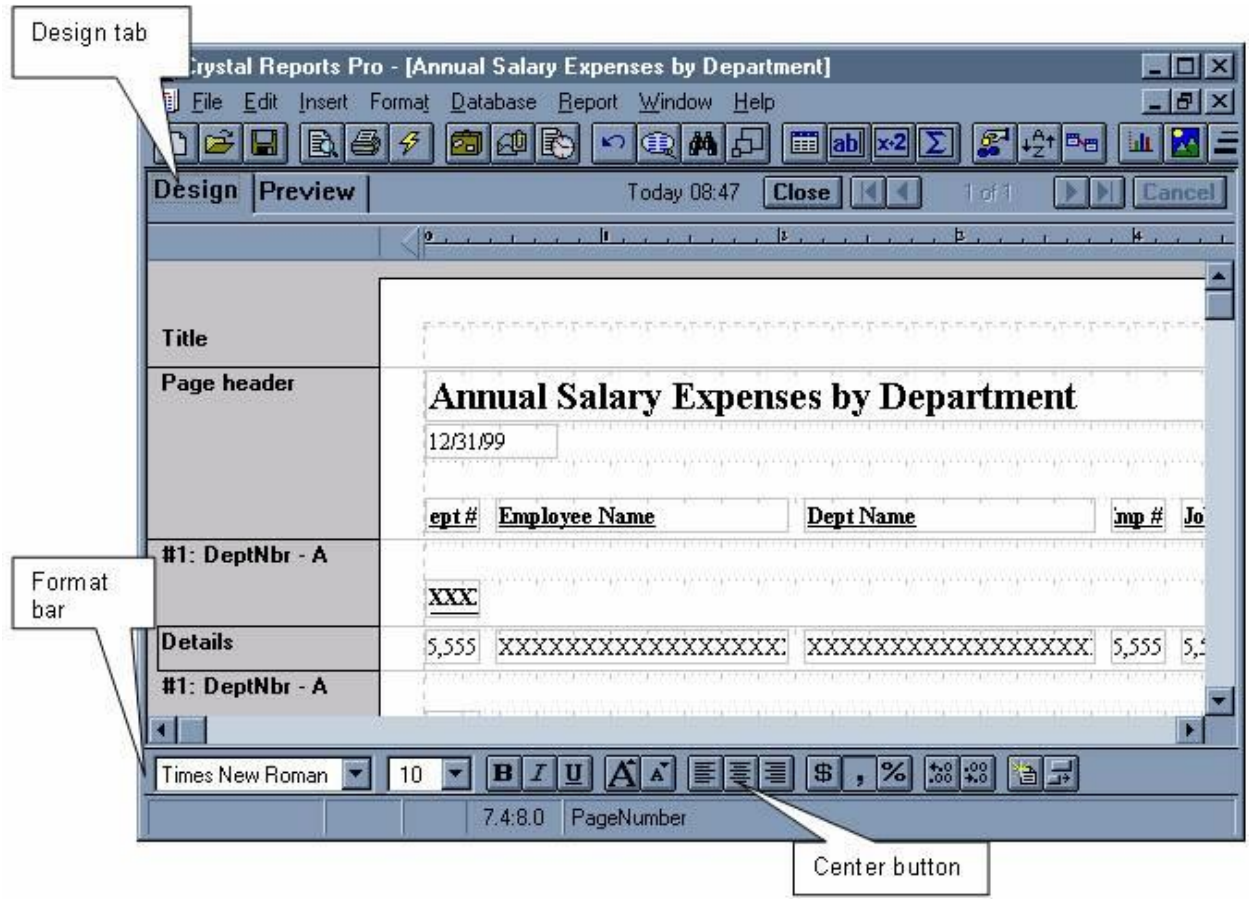
On the Crystal Reports toolbar, click the Zoom button, so you can see what the Expert did for you (it gives you a start, but it needs some work):


Crystal Reports Pro - [Annual Salary Expenses by Department]
Zoom button
File Edit Insert Format Database Report Window Help
Design Preview
Today 08:47 Close 1 of 1 Cancel
Records: 10
100 %

Annual Salary Expenses by Department
4/18/00

Dept #	Employee Name	Dept Name	Emp #	Job #	Job Title
110					
110	MILLS, MARY	ADMINISTRATION	20	1	ADMIN MGR
110	PRIMROSE, PEGGY	ADMINISTRATION	21	2	ADMIN ASST
110					

Click the Design tab:

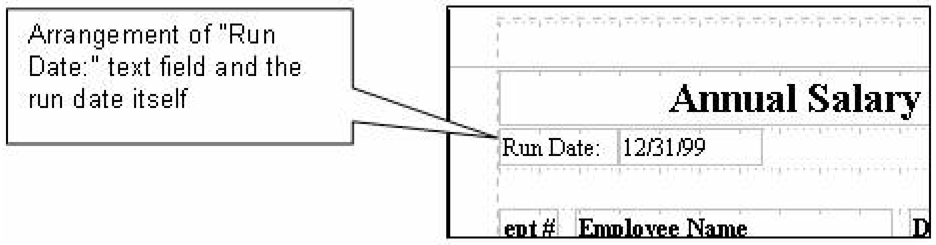

Design tab
Crystal Reports Pro - [Annual Salary Expenses by Department]
File Edit Insert Format Database Report Window Help
Design Preview
Today 08:47
Close
1 of 1
Cancel
Title
Page header
Annual Salary Expenses by Department
12/31/99
ept # Employee Name Dept Name mp # Jo
#1: DeptNbr - A
XXX
Details
5,555 XXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXX 5,555 5,
#1: DeptNbr - A
Format bar
Times New Roman
10
7.4:8.0 PageNumber
Center button

Perform the following steps to fix up the report:

Â· Go to the File menu, select Printer Setup, and change the Orientation to Landscape.

Â· In the Page header area, click the title ("Annual Salary Expenses by Department") to select it. Resize the title so that it spans the width of the entire report. Go to the format bar and click the center button to center the title.

Â· From the Insert menu, select Text Field. In the dialog box that appears, type Run Date: and click the Accept button. At that point, your mouse pointer will also have box representing the text field you just entered. Drag this box to the line where the date is. Use your mouse to arrange the items so that they look like this:

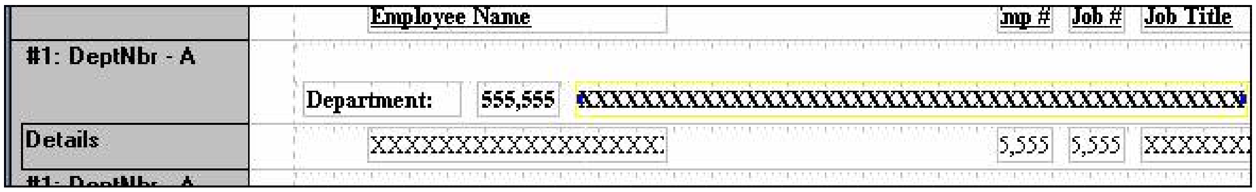

Arrangement of "Run Date:" text field and the run date itself
Annual Salary
Run Date:
12/31/99
ept #
Employee Name

Â·	Still in the Page Header area, remove the column headings for Dept # and Dept Name. (Do this by selecting each item with the mouse and pressing the Delete key.)

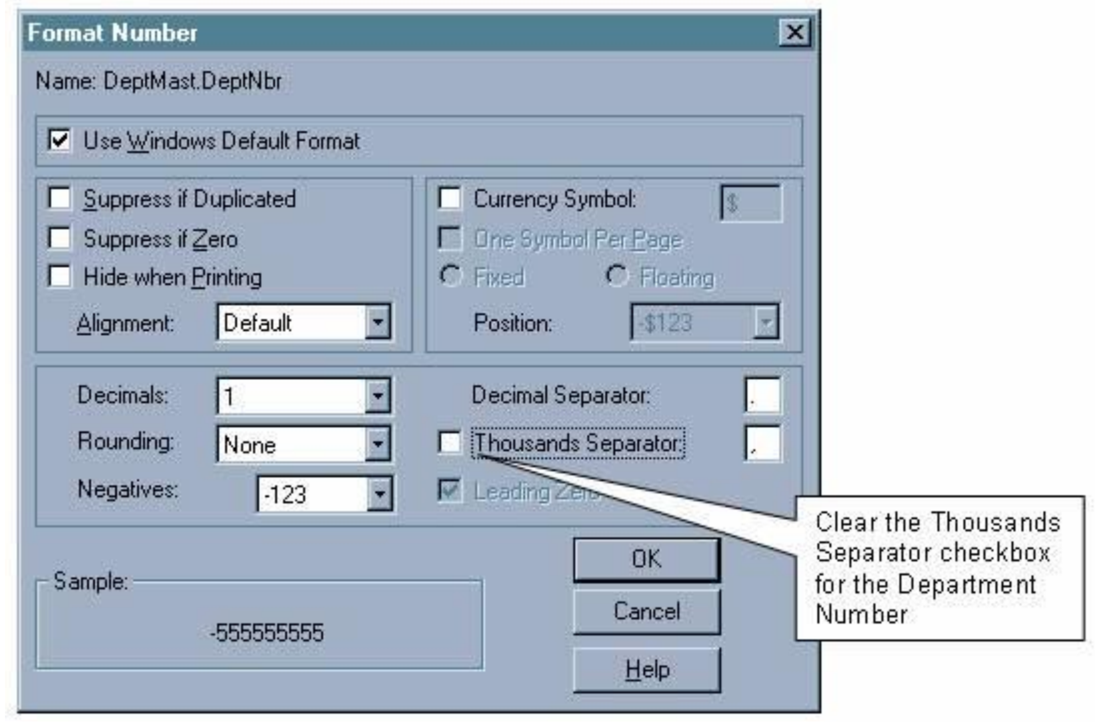
Â·	In the first area labeled #1: DeptNbr â€“ A (the one above Details), select the item there and delete it.

Â·	In the Details area, select the DeptNbr and DeptName fields, and drag them with the mouse to the first #1: DeptNbr -A area.

Â·	Once in the new area, select these two fields and click the Bold button. Insert a text field in this area (using the same technique as you did with "Run Date:") with the text "Department:". Make this text field bold as well. Arrange the fields so that they look like the following (you can resize a field by selecting it and dragging on the handles, just like resizing a control on a VB form):



Â·	Note that the default format for the department number contains a comma. We don't want that. Right click the department number, and select Change Format â€¦ from the context menu. The Format Number dialog appears, as shown below. Clear the "Thousands Separator" checkbox and click OK. (Note: You can also add or remove comma formatting by selecting the field and clicking the comma button on the formatting toolbar.)


Format Number

Name: DeptMast.DeptNbr

[✓] Use Windows Default Format

[ ] Suppress if Duplicated
[ ] Suppress if Zero
[ ] Hide when Printing
Alignment: Default

[ ] Currency Symbol: $
[ ] One Symbol Per Page
( ) Fixed ( ) Floating
Position: -$123

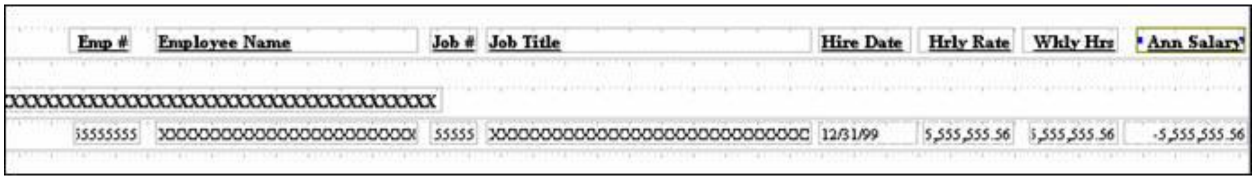
Decimals: 1
Rounding: None
Negatives: -123

Decimal Separator: .
[ ] Thousands Separator: ,
[✓] Leading Zero

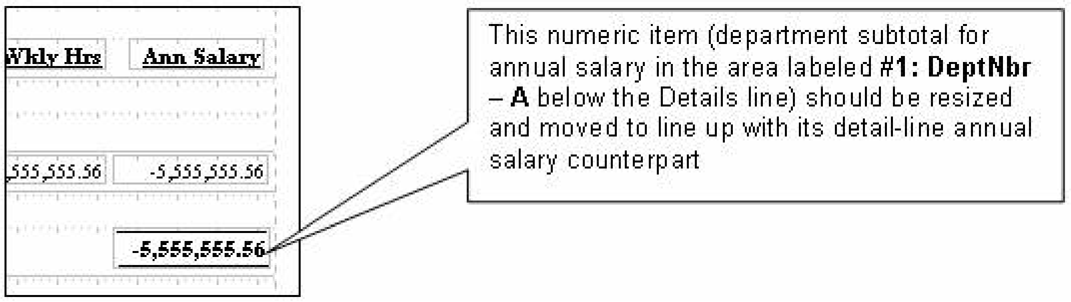
Sample: -555555555

OK
Cancel
Help
Clear the Thousands Separator checkbox for the Department Number

Â· Remove the commas from the formats for the employee number and the job number. Resize the fields on the detail line and resize their corresponding column headings so that the column headings can be fully read and the field data is lined up beneath them. Use the screen shot below as a guide:

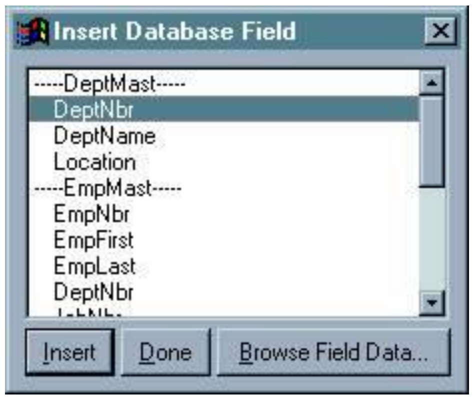


Â· In the second area labeled #1: DeptNbr â€“ A (the one below Details), you will see a text item (denoted by X's) on the far left of the line. Delete that text item. In the same area, toward the right-hand side of the line, you will see a numeric item (denoted by "55.56"). This is the department subtotal. Resize this item to make it larger, and move it to the right so that it lines up with the detail annual salary field, as shown below:

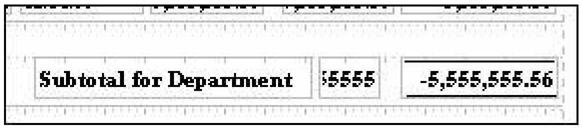

Wkly Hrs
Ann Salary
555,555.56
-5,555,555.56
-5,555,555.56
This numeric item (department subtotal for annual salary in the area labeled **#1: DeptNbr – A** below the Details line) should be resized and moved to line up with its detail-line annual salary counterpart

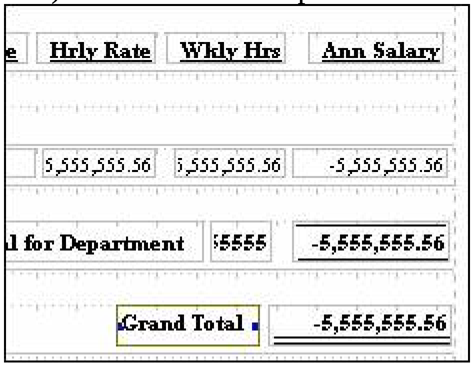
Â·	Insert a text field with the text "Subtotal for Department" to the left of the subtotal (not to the immediate left, because we are going to insert another field between these two). Click the Bold button to make it bold.

Â·	On the Insert menu, select Database Field â€¦ The "Insert Database Field" dialog box comes up (shown below). From it, select DeptNbr and drag the DeptNbr field to the subtotal line, between the text "Subtotal for Department" and the numeric subtotal.

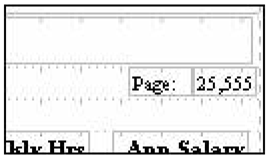


Â·	The second area labeled #1: DeptNbr â€“ A should now look like this:



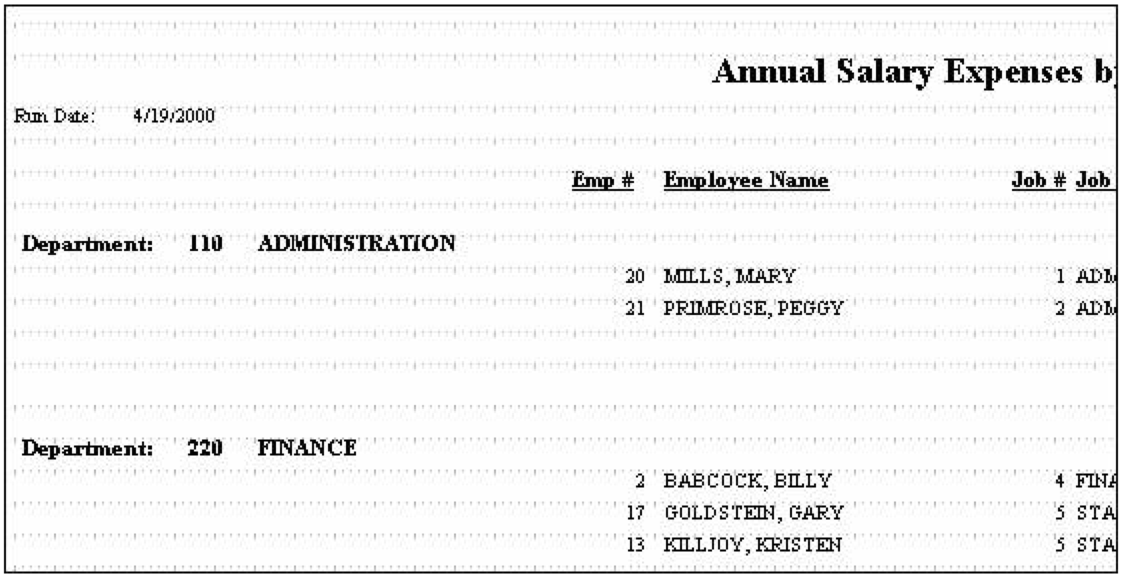
Â·	In the area of the report labeled Grand Total, delete all fields except the first one (a text item with the text "Grand Total") and the last one (a numeric item that is the grand total of the annual salaries). Resize and line up the remaining two items so that the report looks like the following: 

Â·	In the Page Footer area of the report, there is a field for the page number. With your mouse, move this field up to the right-hand side of the page heading area. Add a text field that says "Page:" and place it adjacent to the page number, as in the following screen-shot:



That's just about it. Click the Preview tab to check out the finished product.

Due to the fact this was set up to print in landscape orientation, a few screen shots are presented to show the final report. Below is the left-hand side of the report:

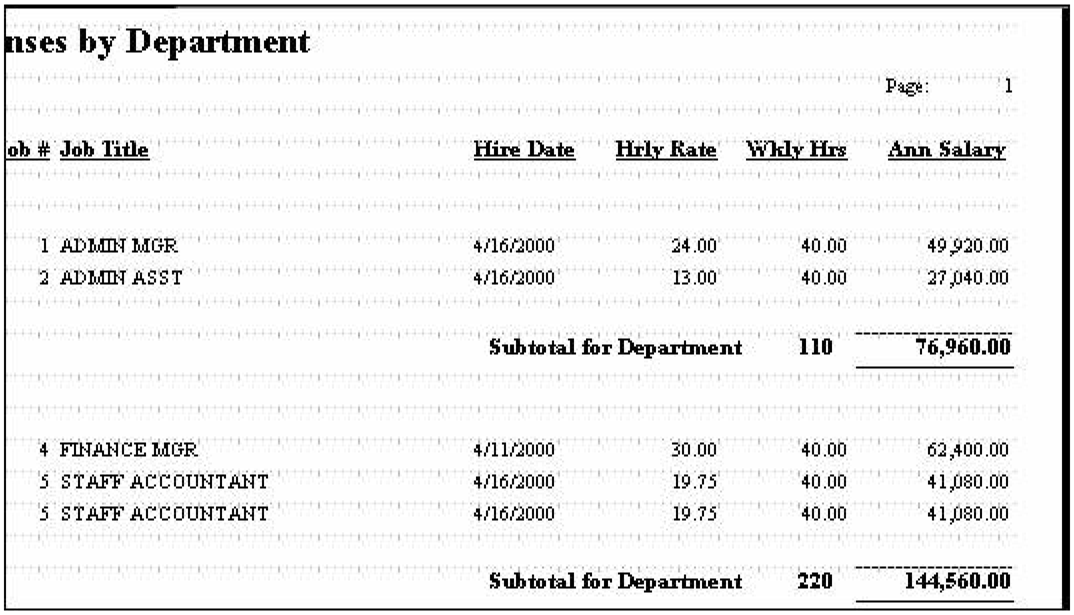


Annual Salary Expenses b

Run Date: 4/19/2000

```
                                   Emp #  Employee Name          Job #  Job
Department:  110  ADMINISTRATION
                                      20  MILLS, MARY                1  ADM
                                      21  PRIMROSE, PEGGY            2  ADM

Department:  220  FINANCE
                                       2  BABCOCK, BILLY             4  FINA
                                      17  GOLDSTEIN, GARY            5  STA
                                      13  KILLJOY, KRISTEN           5  STA
```

Scrolling to the right, we see the following:

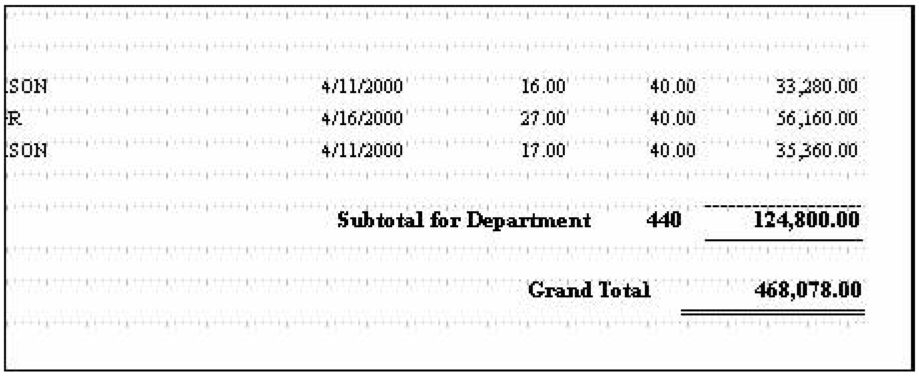


nses by Department

Page: 1

ob #	Job Title	Hire Date	Hrly Rate	Wkly Hrs	Ann Salary
1	ADMIN MGR	4/16/2000	24.00	40.00	49,920.00
2	ADMIN ASST	4/16/2000	13.00	40.00	27,040.00
		Subtotal for Department		110	76,960.00
4	FINANCE MGR	4/11/2000	30.00	40.00	62,400.00
5	STAFF ACCOUNTANT	4/16/2000	19.75	40.00	41,080.00
5	STAFF ACCOUNTANT	4/16/2000	19.75	40.00	41,080.00
		Subtotal for Department		220	144,560.00

Scrolling down, we see the following:



SON	4/11/2000	16.00	40.00	33,280.00
R	4/16/2000	27.00	40.00	56,160.00
SON	4/11/2000	17.00	40.00	35,360.00
	Subtotal for Department		440	124,800.00
		Grand Total		468,078.00

At this point, if you want to do any further tweaking, you can click the Design tab and do so. You can also print the report at this time. Before exiting Crystal Reports, save the report in the same directory as your VB project, under the name SALDEPT.RPT (Crystal Reports automatically appends the .RPT extension).

The real objective is to be able to print the report from the VB program. The how-to for that is coming up shortly, but first, there is another report to create. The second report is similar to this one, except that it will be sorted and subtotaled by job, not department. Fortunately, this second report does not have to be built from scratch. We can modify the first report to create the second report.

To create the second report, follow these steps:

1. Copy the SALDEPT.RPT file and name the new file SALJOB.RPT.

2. Double-click SALJOB.RPT to open it in Crystal Reports.

3. Dismiss the registration screen by clicking Cancel.

4. Click the Design tab.

5. From the Report menu, select Report Title ... The Edit Report Title dialog box appears. Change the title to Annual Salary Expenses by Job and click the Accept button.

6. From the Report menu, select Change Group Expert. The Edit Group Section dialog box appears. There should only be one item in the listbox, reading Group #1: DeptMast.DeptNbr â€“ A. Select that item and click OK. Another Edit Group Section dialog box appears. The first combo box should have "DeptMast.DeptNbr" selected. Click the drop down arrow and select JobMast.JobNbr and click OK.

7. In the Page header area, right-click the title, select Edit Text Field, change the word "Department" to "Job" and click Accept.

8. The next objective is to switch the department fields in the first #1: JobNbr â€“ A area with the job fields in the Details area. To do this, perform the following steps:

a. Move the job number and job title fields from the Details line to an open area of the #1: JobNbr â€“ A line.

b Move the department number and department name fields from the first #1: JobNbr â€“ A line to the space formerly occupied by the job fields in the Details line. Resize the department name field so that it fits.

c. In the first #1: JobNbr â€“ A line, edit the text of the "Department:" text field so that it says "Job:". Move the job number and job title fields close to the "Job:" text field and make these two items bold.

d. In the Details line, remove the bold formatting from the department number and department name fields.

e. Back in the Page header area, change the text of the "Job #" and "Job Title" column headings to "Dept #" and "Dept Name", respectively.

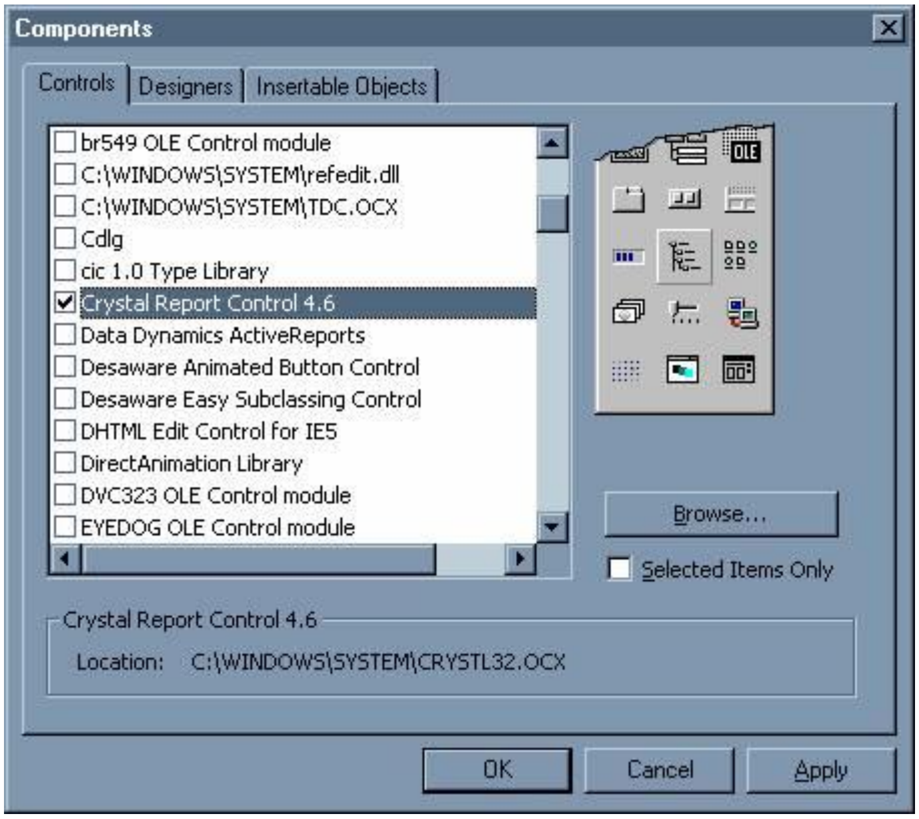
f. Make cosmetic adjustments as necessary.

9. In the second #1: JobNbr â€“ A area, delete the department number field. Replace it with the job number field. To do this, go to the Insert menu, select Database field, and select JobNbr. Resize the JobNbr field so that its small enough to fit where the department number used to be. Remove the commas from the formatting and make it bold. Edit the text field "Subtotal for Department", changing the word "Department" to "Job". Make cosmetic adjustments as necessary.

10. That's just about it. Click the Preview tab to view the fruits of your labor. If you wish, go back and make any adjustments you deem necessary and print out the report. When you are done, save the report and exit Crystal Reports.

How to Print a Crystal Report from a VB Program

First, you must add the Crystal Report control to your VB toolbox. To do so, go to Project Ã Components and check Crystal Report Control 4.6 from the Components dialog box, as shown below:


Components

Controls | Designers | Insertable Objects

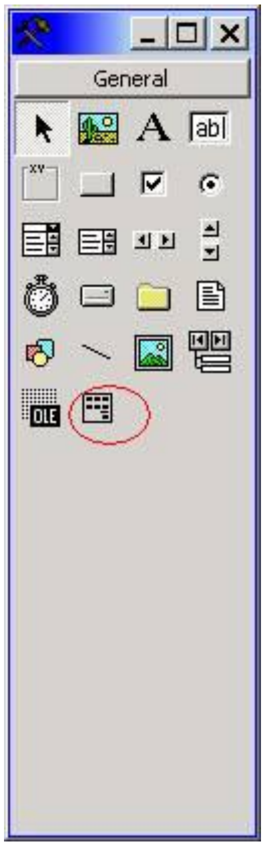
- [ ] br549 OLE Control module
- [ ] C:\WINDOWS\SYSTEM\refedit.dll
- [ ] C:\WINDOWS\SYSTEM\TDC.OCX
- [ ] Cdlg
- [ ] cic 1.0 Type Library
- [✓] Crystal Report Control 4.6
- [ ] Data Dynamics ActiveReports
- [ ] Desaware Animated Button Control
- [ ] Desaware Easy Subclassing Control
- [ ] DHTML Edit Control for IE5
- [ ] DirectAnimation Library
- [ ] DVC323 OLE Control module
- [ ] EYEDOG OLE Control module

- [ ] Selected Items Only

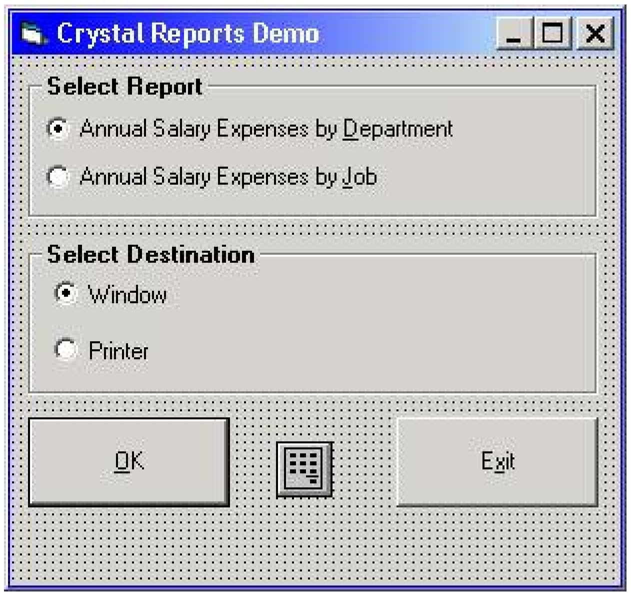
Crystal Report Control 4.6

Location: C:\WINDOWS\SYSTEM\CRYSTL32.OCX

The Crystal Reports control will then appear in the VB toolbox (it is circled in the screen shot below):



The form used in the demo application associated with this topic, named frmCRDemo, is shown below. The form contains two sets of option button control arrays (each contain two option buttons indexed 0 and 1). The first option button control array, named optReport, appears in the frame labelled "Select Report"; the second option button control array, named optDestination, appears in the frame labelled "Select Destination". The Crystal Report control was renamed rptAnnSalExp. No other properties of the Crystal Report control were set at design-time; all necessary properties are set in code at run-time. Only one Crystal Report control is necessary to print any number of reports off of a form. The Crystal Report control is not visible at run-time. Finally, there are two command buttons on the form; the OK button and the Exit button. The OK button runs the report based on the options selected by the user in the frames above; the Exit button ends the application.



Following is the code for both the frmCRDemo form and a standard module called modCommon. The Crystal Reports-related code will be explained following the code listings.

Code for frmDemo:

```
Option Explicit


'-----------------------------------------------------------------------
Private Sub Form_Load()
'-----------------------------------------------------------------------
CenterForm Me

End Sub
```

```
'-----------------------------------------------------------------------
Private Sub cmdOK_Click()
'-----------------------------------------------------------------------

On Error GoTo cmdOK_Click_Error

Dim strReportName As String
Dim intReportDestination As Integer

If optReport(0).Value = True Then
strReportName = "SALDEPT.RPT"
Else
strReportName = "SALJOB.RPT"
End If

If optDestination(0).Value = True Then
intReportDestination = crptToWindow
Else
intReportDestination = crptToPrinter
End If

With rptAnnSalExp
.ReportFileName = GetAppPath() & strReportName
.DataFiles(0) = GetAppPath() & "EMPLOYEE.MDB"
.Destination = intReportDestination
.Action = 1 ' 1 = "Run the Report"
End With
```

```
Exit Sub

cmdOK_Click_Error:

MsgBox "The following error has occurred:" & vbNewLine _
& Err.Number & " - " & Err.Description, _
vbCritical, _
"cmdOK_Click"

End Sub

'-----------------------------------------------------------------------
Private Sub cmdExit_Click()
'-----------------------------------------------------------------------
Unload Me
End Sub
```

Code for modCommon:

```
Option Explicit

'-----------------------------------------------------------------------
Public Sub CenterForm(pobjForm As Form)
```

```
'-----------------------------------------------------------------------

With pobjForm

.Top = (Screen.Height - .Height) / 2

.Left = (Screen.Width - .Width) / 2

End With


End Sub


'-----------------------------------------------------------------------

Public Function GetAppPath() As String

'-----------------------------------------------------------------------


GetAppPath = IIf(Right$(App.Path, 1) = "\", App.Path, App.Path & "\")


End Function
```

Note that in the cmdOK_Click event procedure, the variable strReportName is set to "SALDEPT.RPT" or "SALJOB.RPT", depending on which optReport button was clicked. The variable intDestination is set to either crptToWindow or crptToPrinter, both built-in Crystal Reports constants, depending on which optDestination button was clicked (if you choose to send the report to a window, a "print preview" type screen appears allowing the user to view the report on screen; if you choose to send the report to the printer, it will be sent directly to the default printer with no preview).

The sample app references the following properties of the Crystal Report control:

Â·   ReportFileName refers to the name of the report definition that you saved in Crystal Reports. Alternatively, this property can also be set at design-time.

Â·   DataFiles is a Crystal Reports property array specifying the database file(s) to be used as the basis for the report. The file you specify here will override the database file used when the report was created in Crystal Reports. Although different MDB files can be used (at Crystal Reports design-time vs. VB run-time), they still must contain the same table and/or query names, with the same structure, that were used to build the report.

Â·   Destination refers to where you want to direct the output of the report. In code, you can use a predefined constant (like crptToPrinter) or its numeric equivalent as used in the sample code. This property can also be set at design-time. The sample app uses either crptToWindow (numeric value of 0) or crptToPrinter (numeric value of 1).

Â·   Action is the property that triggers the running of the report. It must be set to 1 to run.

Download the project files for the sample application here.