and

```
Forms![frmClients]![Surname].ControlSource
```

You might have noticed the use of a dot operator for properties, unlike the bang used for object names. Although Access sometimes accepts the wrong operator, use the bang for any object that you name, and use the dot for any built-in system names such as properties.

As you can no doubt imagine, object references can become quite lengthy. It was with this in mind that you were warned in an earlier chapter against monstrous names that require typing things such as the following:

```
Forms![My main form for entering new clients]
➥![Street Number and Street Name].EnterKeyBehavior
```

Fortunately, it is not necessary to use the full syntax to refer to another control within the same form. For example, if you want to create a textbox to display course 201 from department PSY as PSY201, you could add a text box to the same form and simply set its `ControlSource` to the following:

```
=[DepartmentID] & [CourseID]
```

Note the initial =. Whenever you want to set one control equal to another, the equal sign must be included at the beginning of the `ControlSource` property.

In version one, only a very few properties such as `Visible` and `Enabled` could be modified while a form was in use. Now, only a few properties cannot be modified in use. This means you are able to change almost anything about a form or its controls in response to the data they contain or the way they are being used: fonts, sizes, colors, formats, content, behavior, visibility, editability, sorting, filtering, and even what a user is permitted to do next.

How to Refer to Controls on Subforms

How Open any form in Design view, and if it contains a subform, you will see a white area where the child form's data is to be displayed. This area is a subform control, with properties of its own just like a text box or other control. For example, hiding the subform control could be accomplished with the following:

```
Forms![frmClients]![frmMySub].Visible = No
```

The subform control contains a child form when the form is in use. However, the subform control is not the same thing as the child form. The subform control has a `Form` property that you must use to refer to the child form, as follows:

```
Forms![frmClients]![frmMySub].Form
```

You can then refer to a control within the subform, as follows:

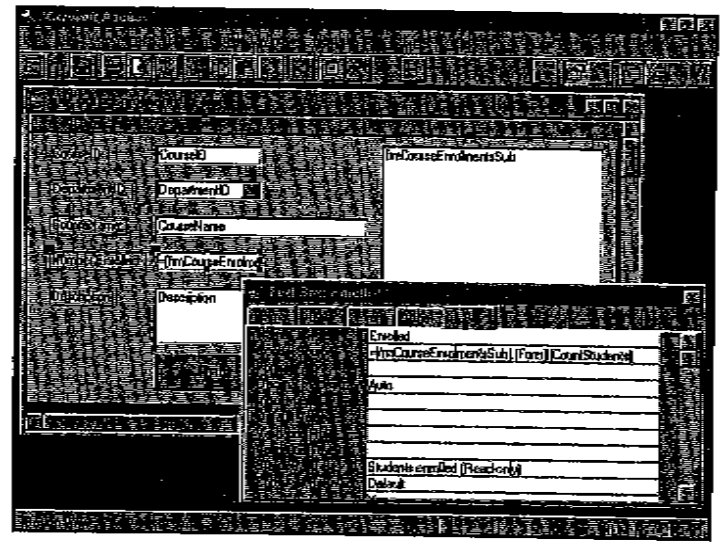
```
Forms![frmClients]![frmMySub].Form![MyControl]
```


Most Access programmers forget Form more than once before it becomes second nature. Remember that you do not have a reference to the subform itself until you go through the `.Form` property of the subform control.

You now know how to bring a total from a subform back onto the main form. Figure 13.1 illustrates the case where a subform contains a control called CountStudents, which is the count of records in the subform. A control on the main form displays the count as Number Enrolled, with a text box bound to the expression:

```
= [frmCourseEnrolmentsSub].Form![CountStudents]
```

Figure 13.1.
A control on the main form can be bound to a control in a subform.



When working within a subform, it can also be handy to refer to the parent form without having to type the full syntax or even without knowing the parent form's name. To do this, use the `Parent` property. All controls have a parent, and in most cases, the parent is the form that contains the control. For example, if you are working in a subform with a control named Quantity, you can refer to the subform as

```
[Quantity].Parent
```

and the main form as

```
[Quantity].Parent.Parent
```

When to use these references will become clear in the following chapters. For now, just note that the `Parent` property is a useful way of going back up the hierarchical object tree.

How to Refer to Controls on Reports

How

The syntax for reports is identical to that for forms, except that the collection of open reports is called Reports. If a report called rptSurvey is currently open, it is referenced as follows:

```
Reports![rptSurvey]
```

The control called HighestPeak is

```
Reports![rptSurvey]![HighestPeak]
```

A subreport control has a `Report` property that can be used to refer to the subreport it contains, as follows:

```
Reports![rptSurvey].Report![MySub]
```

Properties are also referenced in the same way:

```
Reports![rptSurvey]![ReportDate].Visible
```

A naming error in a report can sometimes be hard to trace. The report either pops up a dialog box asking for an unexpected parameter, or a control displays `#Name?`. Although these problems apply equally to forms, they tend to occur more often with reports. Here are some suggestions that might help.

Double check the syntax. Is there a missing = in a `ControlSource` property? For example, a control bound to `Sum([Amount])` will not yield the total until altered to `=Sum([Amount])`.

Are you trying to use an aggregate function on a calculated field? For example, if your report contains controls for Quantity and UnitPrice, you might add an Amount control with a `ControlSource` of `=[Quantity] * [UnitPrice]`. This works fine, but if you then try to include a total for Amount, Access cannot total the calculated field. The solution is to include the calculated field in the underlying query; if it exists in the query, the report will have no problem totaling the field.

Check for any ambiguity in the control's Name. If a field is present in the query feeding a report but not used on the report, you can still refer to it using the same syntax as you do for controls. For example, even if the SurveyID field is not used on the report, you can still refer to the data in the field as follows:

```
Reports![rptSurvey]![SurveyID]
```

If a control and a field both have the same name, and both are referenced by the same syntax, there is a potential for ambiguity. Generally, this is not a problem because the two values will be identical. Why would they be different?

Consider this situation. You allow a report wizard to create a tabular listing that includes Surname and FirstName fields. The wizard names these controls with the same name as the fields. As you examine the results, you decide the report would look better if the names were combined, so you delete the second control and alter the `ControlSource` for the first one as follows:

```
=[Surname] & ", " & [FirstName]
```


You now have a control with the same name as a field but bound to something different. Sooner or later this problem will catch you, unless you always remember to rename a field when you change its `ControlSource`. Alternatively, follow the practice of always using a different name for the control so that the ambiguity can never occur. The problem illustrates why many professional Access programmers consider it worth the effort to use a naming convention such as Leszynski/Reddick to differentiate between objects.

How to Refer to Data in a Table

How

Quite frequently, you will need to refer to data stored in a table that is not included in a form or report's `RecordSource`. This is accomplished by creating a text box to display the result and using the `DLookup()` function in its `ControlSource`. Each time you move to the next record, `DLookup()` provides the appropriate data in the text box.

The `DLookup()` function expects three arguments inside the brackets. Think of them as follows:

Look up the...field, *from the*...*table, where*....

Each of these arguments must be placed in quotation marks and separated by commas. The technique itself is straightforward, but it is complicated by the need to embed quotes with quotes.

Imagine that you have a CompanyID such as 874, and you need to print the company name on a report. Filling in the blanks in the previous expression, you have the following:

Look up the CompanyName *field from the* tblCompany *table, where* CompanyID = 874

To achieve this, the text box on your report will have a `ControlSource` of

```
=DLookup("CompanyName", "tblCompany", "CompanyID = 874")
```

You probably won't want the name of Company 874 printed for every record. It is more likely that the 874 will be in a control such as CompanyID. Using an ampersand to concatenate the current value of the CompanyID field to the Where clause, this translates to

```
=DLookup("CompanyName", 'tblCompany", 'CompanyID = " & [CompanyID])
```

This technique can even be used to look up another record in the same table. When breeding animals for example, it is usual to have a field for SireID and another for DamID, containing the primary key values for the sire and dam from the same table. To display the names of the parents on a form, two text boxes are added with their `ControlSource` bound to `DLookup()` expressions as follows:

```
=DLookUp("Name","tblHorses","ID = " & [SireID])
```

The result will look like Figure 13.2. (Note the consistent use of a different background color for fields the user cannot edit.)