

# Starting the Interface with Access Data Projects

Martin Reid



This month, Martin Reid shows you how to build the user interface to an Access Data Project. You'll see how easy it is to create an Access form that's bound to a stored procedure without giving up the benefits of parameters.

**I**n this article, I'll start building the user interface to an Access Data Project application (the core database can be found in the Download file that's available at [www.smartaccessnewsletter.com](http://www.smartaccessnewsletter.com)). I'll introduce you to the features of binding forms that will be new to you when working with forms and Access Data Projects.

When you enter the world of ADPs, you'll find many new options when creating the user interface to your database. Some of these features have the same name as features in Access/Jet development but bear no resemblance to their counterpart in Access/Jet development—queries are the best example. In addition to the familiar names masking new features, there are new options, including user-defined functions, in-line table value functions, input parameters, and stored procedures. However, as you'll see, your existing skills are easily transferable to the world of SQL Server.

This month, I'll be showing you how to set up forms to work with stored procedures and how to pass parameters to them using a form's Input parameters property. Stored procedures are an important part of any server database (like SQL Server) and a critical item in getting the best performance out of the database.

## Building your first form

You can build a form in an Access Data Project just as you would in Access/Jet development, by selecting Forms | New in the database window. As in Jet, in the resulting dialog you can select the Form Wizard from the list box at the top of the dialog and (if you're using my sample project in the source code) select the Staff table as the record source from the drop-down list at the bottom of the dialog. The rest of the process for creating the form is just as familiar to Access/Jet developers. I'll use the form that the wizard would generate to point out those areas that are new to ADPs (Figure 1 shows the generated form).

Most of what you can see in Figure 1 should be

familiar to you from Access/Jet development. Notice that the primary key is even identified as an AutoNumber datatype. The navigation bar at the bottom of the form shows the first major departure from Access/Jet. The "X" button that appears there will be red when the form is downloading data. This is important because the user can click that button to stop the data transfer to the form.

Remember that with ADPs, you could be working with tables containing millions of rows. The last thing that you want is for all these records to be pulled down to the client. One of the most important aspects of developing with SQL Server and ADPs is restricting the number of records downloaded to the client. As I progress I'll be demonstrating various techniques that can be used to limit the records returned to forms, including several new form properties. However, the red X button in the navigation bar will let you (or your users) terminate data retrieval if you do end up retrieving those millions of records.

## Form properties

As in table design in ADPs, many of the form properties in an ADP form are identical to those used in Access/Jet. However, some are new, and others have been extended to include new functionality. Table 1 lists some of the form properties that you may not be familiar with.

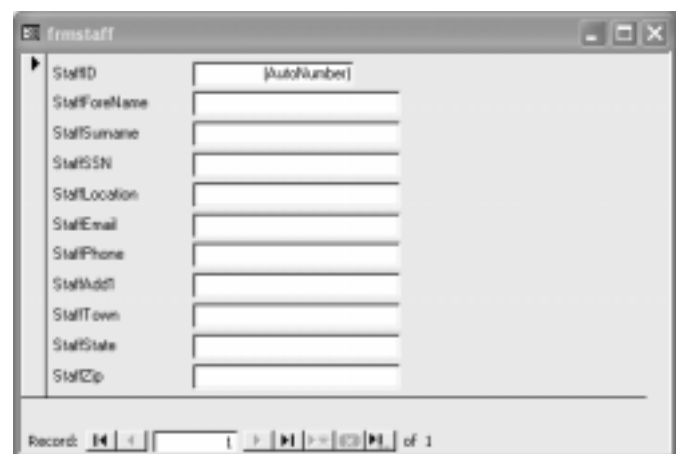


Figure 1. An auto form generated in an ADP application.

Just like with Access, there are several ways you can supply data to forms. Since stored procedures and views are new to ADPs, I'll use them in this article to create some simple forms. In later articles, I'll be looking at more sophisticated options.

### Additional forms

Before proceeding, I'll also need the following forms (all of which can be created using the Access wizards):

- *frmMaintainProjects*—This is a simple Master/Detail form created using the Form Wizard. It uses the Project and Task tables from my sample table as its record source.
- *frmAssignstafftask*—This, again, is a simple wizard-generated form that permits the user to allocate staff to specific tasks. This table uses the Staff, Tasks, and Staff\_To\_Task tables. The main form is based on the Task table. The form selects data from both the Staff and the Staff\_To\_Task tables to create the subform.

### Stored procedures

A stored procedure is a precompiled collection of SQL statements. Because it's precompiled, it has speed advantages over standard Access queries. Unlike SQL issued from Access code, with stored procedures the server creates an execution plan for the procedure that contributes to the stored procedure's speed. SQL Server's version of SQL is known as Transact SQL (T-SQL) and is almost a full programming language with branching, conditional statements, Case structures, and error control. If the thought of learning T-SQL is intimidating, remember that T-SQL is nothing more than SQL: different in many ways, but still the same SQL that you've written for years.

Stored procedures have many advantages over Access queries, including:

- The ability to be shared by different applications, including Web-based apps.
- Reduction in network traffic, as a single call to a stored procedure can execute many SQL statements.
- Encapsulation, as complex business rules can be

## Extended Properties

It's possible to use Input Masks with SQL Server tables. Just to show how much ADPs are like Access/Jet, go to the properties of the text boxes and add the following Input Mask: 00/00/0000;0. You can even add Input Masks to fields in your tables (for example, the SSN field in the staff table could use an Input Mask of !00-000-0000). The ability to use Input Masks with tables is new to SQL Server 2000 as part of Extended Properties and enhances the ease with which Access developers can work with the database.

A quick way to discover the extended properties used in a SQL Server table is to run the following SELECT statement using a row-returning system function. Running the following SQL statement either in Access as a stored procedure or from the SQL Server Query Analyzer gives the results shown in Figure A.

```
SELECT * FROM ::fn_listextendedproperty
(NULL, 'user', 'dbo', 'table',
'Project', 'column', default)
```

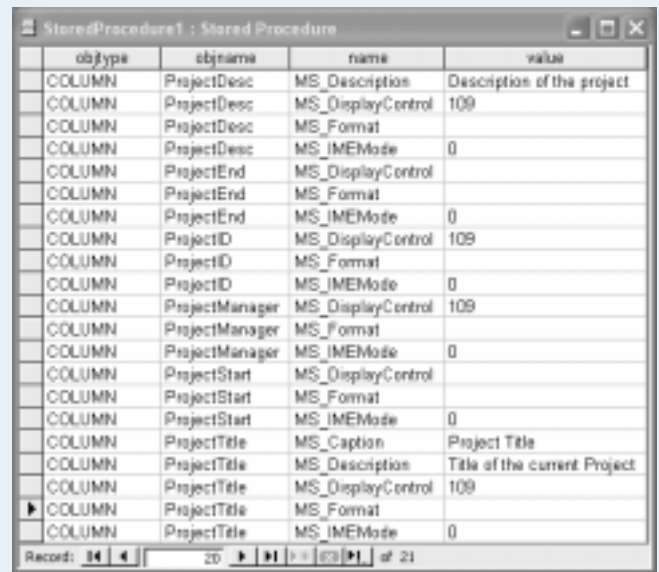


Figure A. Discovering Extended Properties.

Table 1. Form properties.

Property	Description
Recordsource Qualifier	The owner of the object being used to supply data to the form. See the sidebar on table owners for more information.
Max Records	Sets the maximum number of records to be returned by the form. Can also be set programmatically using ADO.
Server Filter	A simple WHERE clause applied to the data at the server. If the form's recordsource is a stored procedure, then this is ignored.
Server Filter By Form	Opens the form in filter mode. Similar to the form filter used in Access, but the filter is applied at the server before the records are sent to the client.
Unique Table	When working with multiple tables, views, and stored procedures, you can select one as unique. This table will be updateable. The Access Help documentation states that this property must be set or the recordset will be read-only. This isn't in fact the case, as we shall see later.
Input parameters	One of the more interesting properties. Can be used to pass values from forms to stored procedures.

“hidden” within a stored procedure.

- Better speed through pre-compilation and optimization.

### Creating your first stored procedure

Access 2002 ADPs let you create stored procedures using a graphical builder—just as if you were creating a standard Access query (Access 2000 isn’t quite as friendly). I’ll create a stored procedure that returns a record from the Staff table. To create the procedure:

1. Select Queries in the database window.
2. Click on New to display the New Query dialog.
3. From the New Query dialog, select Design Stored Procedure.

A graphical query builder now opens, which isn’t unlike the Access/Jet query builder (see Figure 2). Just like the usual QBE in Access, it’s simply a matter of selecting the columns that you want to use within the stored procedure. To select a column, simply click on the check box to the left of the column name. Just like Access, the column “drops” into the lower half of the QBE.

Before executing the stored procedure, you must save it. I saved this particular example as `usp_Staff` (the prefix indicates that this is a User Stored Procedure).

Within the query builder, you’re shielded from the full syntax used to create the procedure. To take full advantage of stored procedures, you must learn how to code in T-SQL, but for simple queries like this that’s not necessary. For instance, when using the graphical tools,

what you’re actually doing is defining the SQL statement used in a Create Stored Procedure statement. When the procedure is saved, the Create statement is executed, creating the stored procedure with your SQL statement in it. The full syntax can be viewed from the main menu by selecting View | SQL:

```
CREATE PROCEDURE dbo.usp_staff
AS SELECT StaffID, StaffForeName, StaffSurname,
StaffSSN, StaffLocation, StaffEmail, StaffPhone
FROM dbo.Staff
```

There are one or two things to note about this statement. When you’re creating a stored procedure, you use the Create Procedure statement. However, once it’s been saved and viewed, the Create keyword is removed and replaced by the Alter keyword (this can lead to some confusion when you’re starting to work with these objects). By switching to Alter, the procedure can be amended without having to delete it. The formal syntax of Create stored procedure is:

```
CREATE PROC[EDURE] procedure_name [ ; number ]
[ { @parameter data_type }
[ VARYING ] [ = default ] [ OUTPUT ]
] [ ,...n ]

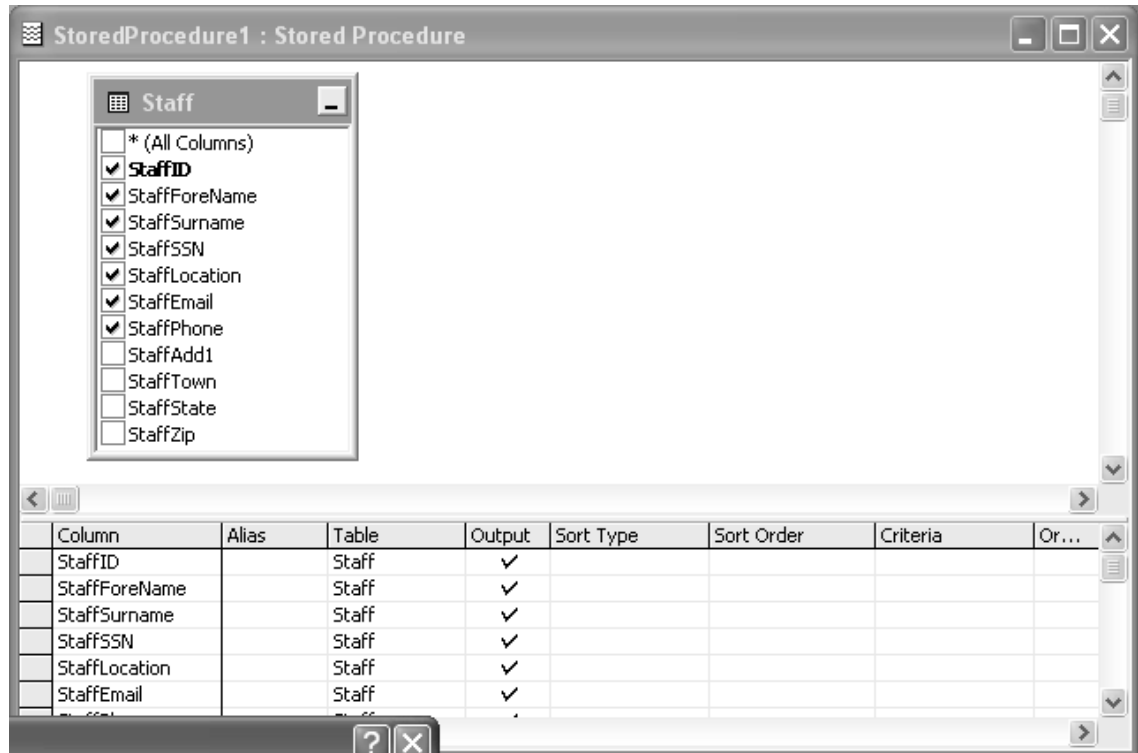
[ WITH
{RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION}]

[ FOR REPLICATION ]

AS sql_statement [ ...n ]
```

Now that you have a stored procedure, it’s a simple matter to use it as the recordset of a form. If you run the

Figure 2. The stored procedure builder.



Form Wizard, you'll find that the stored procedure is now available from the Source drop-down list in the dialog. Simply select the stored procedure and proceed as usual to complete the form, using the stored procedure as the form's record source. The recordset is fully updateable with the actual updates handled by system procedures within SQL Server.

### Using a parameter with a stored procedure

It's often useful to view only a subset of data. Stored procedures permit you to use both input and output parameters. Passing an input parameter to a stored procedure allows you to control how much data is retrieved. However, using parameters with stored procedures, particularly via forms, appears to be something that confuses many Access developers. Much of the Microsoft documentation (which points out all the things that you can't do with stored procedures) gives the appearance that it's quite difficult. In reality, it's neither impossible nor difficult.

### Input parameters

Parameters that are to be passed to a stored procedure are specified in the query using the form @Parameter. For example, this sample uses a parameter in its Where clause:

```
Select *
  From tablename
 Where column = @parameter
```

To indicate a parameter in the query designer, just enter the parameter name with the @ in front of the name in the criteria column.

You use the Input parameters property of a form to pass form values to the parameters of stored procedures. I'll re-create the usp\_Staff stored procedure to accept a single parameter, StaffID:

```
CREATE PROCEDURE dbo.usp_StaffPara
```

```
(@StaffID int)
AS SELECT StaffID, StaffForeName,StaffSurname,
StaffSSN,StaffLocation,StaffEmail,StaffPhone
FROM dbo.Staff
WHERE (StaffID = @StaffID)
```

In this example, I've made use of the table owner prefix (dbo). Many developers recommend prefixing the owner name to the table name to remove confusion. On some occasions, the two-part name must be used as discussed in the sidebar "Table Owners."

When the stored procedure is executed from the database query window, you're prompted in the usual way for a StaffID. Of course, being an ADP there's a slight difference in the prompt: The dialog also provides a drop-down list. Using the drop-down list, you can select a Default or Null value to be passed to the procedure.

### Using defaults

In order to cope with the possibility of the user not entering a value, you can also assign a default value to the stored procedure's parameter. I'll amend the usp\_staff procedure to include a default for the StaffID parameter like this:

```
CREATE PROCEDURE dbo.usp_StaffPara
(@StaffID int=1)
AS SELECT StaffID, StaffForeName,StaffSurname,
StaffSSN,StaffLocation,StaffEmail,StaffPhone
FROM dbo.Staff
WHERE (StaffID = @StaffID)
```

## Table Owners

In Access, it's assumed that all tables are owned by Admin and duplicate table names aren't permitted. In SQL Server, it's possible to have tables with the same name but created by different users (for instance, martin.Customers and dbo.Customers). Without the dbo prefix, the server will first look for a table prefixed by the current user's name. If SQL Server fails to find a matching table, it'll then look for a table owned by dbo. I recommend that you only permit tables to be owned by the dbo or the system administration user. The Recordsource Qualifier property on a form lets you specify the owner of the tables used in the query.

However, if you now execute the procedure from the database window, Access 2002 doesn't prompt you for a value for the parameter. Once you add a default value, the procedure simply executes and restricts records to those meeting the default value. In order to be prompted when using default values in Access ADPs, you must use User-Defined Functions (UDFs), which are new to SQL Server 2000.

### Passing parameters via forms

In order to pass a parameter using a form, you use the form's Input parameter property. For this example, to demonstrate the flexibility in ADPs, I'm going to create a couple of forms. Form 1 will be used to collect the data from the user and will be unbound. Form 2 will be based on a stored procedure and will pass two parameters, @DateFrom and @DateTo, to my stored procedure. The parameters will be passed from Form 1 to Form 2 and then to the parameters to filter the records returned. There's nothing stopping you from passing parameters from controls on the bound form, but this example is a little more interesting.

The first step is to create the stored procedure that accepts the two parameters:

```
CREATE PROCEDURE dbo.usp_Stafftasks
(@DateFrom datetime,@DateTo datetime)
AS SELECT  dbo.Staff.StaffForeName,
          dbo.Staff.StaffSurname,
          dbo.Staff_To_Task.DateAssigned,
          dbo.Tasks.TaskTitle, dbo.Staff_To_Task.Notes
FROM      dbo.Staff INNER JOIN dbo.Staff_To_Task
          ON dbo.Staff.StaffID = dbo.Staff_To_Task.StaffID
          INNER JOIN dbo.Tasks
          ON dbo.Staff_To_Task.TaskID = dbo.Tasks.TaskID
WHERE     (dbo.Staff_To_Task.DateAssigned
          BETWEEN @DateFrom AND @DateTo)
```

The next step is to create a simple unbound form containing two text boxes (txtDateFrom and txtDateTo) and a command button. The user will enter the required dates into the boxes and click the command button to open the Staff\_Tasks form. The data displayed in the

## SQL Desktop (Formerly MSDE)

Developers using MSDE will be at a disadvantage compared to developers using the full version of SQL Server. One of the most useful tools available in developing with SQL Server is the Enterprise Manager, which isn't included with MSDE. There's a way to get Enterprise Manager. If you download SQL Server 2000 Enterprise Edition (which expires after 120 days), you get Enterprise Manager as part of the package. However, after the SQL Service license expires, Enterprise Manager appears to still function. The licensing issues surrounding this are unclear (but when Bill reads this maybe they will become clearer!).

Staff\_Tasks form will be based on the date values entered into the text boxes. I've defaulted txtDateFrom to Now() and txtDateTo to Now()+7. Both have been formatted as Short Date.

The second form (frmStaffTasks) is used to display the records returned by usp\_Stafftasks. Again, I used the Form Wizard to create the form, this time selecting usp\_Stafftasks as the record source. In the form's properties sheet, I set its Input parameter properties to this statement:

```
@DateFrom Datetime =
[Forms]![frmPassDates]![txtDateFrom],
@DateTo Datetime =
[Forms]![frmPassDates]![txtDateTo]
```

This statement sets the DateFrom parameter of the stored procedure to the value in the txtDateFrom control on the form frmPassDates; it also sets the DateTo parameter to the txtDateTo control. As you can see, you also need to specify the datatype for the parameter (in this case, Datetime). Figure 3 shows the results when the command button is clicked.

For these examples, and to demonstrate to you how to get started quickly with ADPs, I simply based my recordsets on SQL Server tables. This isn't the recommended approach, and next month I'll be looking at more sophisticated techniques. I'll also investigate some of the other methods for reducing the amount of data passed to the client. I shall also begin to touch on some SQL Server security issues. See you next month. ▲

**DOWNLOAD** [ADP0205.ZIP at www.smartaccessnewsletter.com](http://www.smartaccessnewsletter.com)

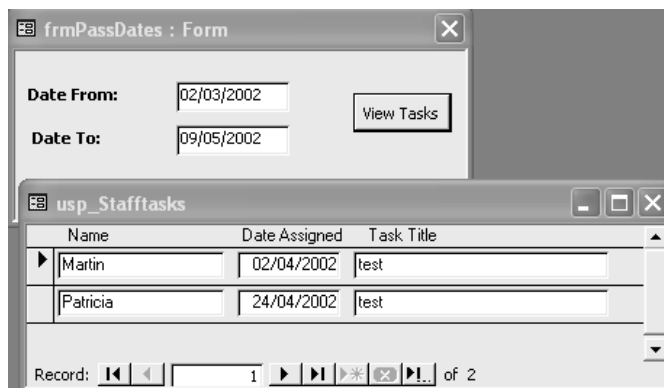


Figure 3. A form based on a parameterized stored procedure.

- [XMLREPORT.ZIP](#)—Danny Lesandrini continues to find uses for Access 2002 support for XML. In this sample he shows how to use a combination of Access 2002 and ASP to create reports that can be viewed over the Web. Reports can be generated either on schedule or on demand. (Access 2002)
- [FRMCLSS.ZIP](#)—Garry Robinson demonstrates several cool techniques for using Forms as Class objects to enhance your application. Garry shows how to create duplicate copies of forms, modify one form from code in another, and add new methods to your forms. Not all features work in the Access 97 version. (Access 2000, Access 97)
- [COMPARE.ZIP](#)—This download from Rickard Olsson demonstrates how to match rows in separate tables in order to create a complete record. The sample code also allows you to test to see which method for loading the tables gives you the best performance. (Access 97)
- [ADP0205.ZIP](#)—To accompany his series on creating Access ADPs, Martin Reid has provided a sample database that demonstrates the techniques covered so far in his articles. (Access 2002)