May, 2005

Advisor Answers

Creating Access databases

VFP 9/8/7

Q: I need to create an Access database, including the tables, from inside my VFP application. How can I do it?

A: You have two choices for creating an Access database with VFP code. One of them works whether or not Access is installed on the target machine; the other requires Access to be present. Either approach lets you create the Access MDB file, add tables to it, and if you want, add data to those tables.

I'll look first at the solution that requires Access. It uses Automation and DAO, one of Microsoft's older data access technologies. As usual, you can use VFP's Object Browser to explore the object models involved, but I figured out what commands were needed by examining a public domain tool originally created by John Koziol (now a member of the VFP team), which I downloaded from the Universal Thread. That tool, which provides conversion both ways from VFP to Access, but has some instabilities, is included for your reference on this month's Professional Resource CD as Converter.ZIP.

Creating an Access Automation object is easy:

```
oAccess=CREATEOBJECT("Access.Application")
```

The Application object's NewCurrentDatabase method lets you create a database and make it current. Once a database is current, it can be accessed through the CurrentDB method, which returns a reference to a DAO Database object. You can use the Database object to define tables. This code creates a new database called AccessTest and grabs a reference to it:

```
oAccess.NewCurrentDatabase("AccessTest.MDB")
oDB = oAccess.CurrentDb()
```

To create a table in a DAO database, you create a table object, add fields to it, and then add the table object to the database. Adding fields to a table works the same way--you create a field object, set its properties, and then add it to the table.

The method for creating a table object is CreateTableDef; it has one required parameter, the name of the table. CreateTableDef returns an object reference to a DAO TableDef object. The database object has a TableDefs collection; to add a table to the collection, use its Append method, passing the TableDef object as a parameter.

The TableDef object has a CreateField method you can use to add fields to the table. It has one required parameter, the name of the field. You can also pass the field type and size. For field type, you need to use an integer value. Table 1 shows some VFP data types and the corresponding DAO constants and integer values. You can also create the field first and then specify the type and size by setting the Type and Size properties of the Field object. CreateField returns a Field object.

Table 1. Field types for DAO--To create fields for an Access table, you need to specify the data type using the appropriate integer value.

VFP type	**DAO constant**	**DAO value**
Character	dbText	10
Memo	dbMemo	12
Logical	dbBoolean	1
Date, DateTime	dbDate	8

This code creates a table called Customers and adds two fields:

```
oTable = oDB.CreateTableDef("Customer")

* Add fields
oField = oTable.CreateField("iID",dbInteger)
oTable.Fields.Append(oField)

oField = oTable.CreateField("Company", dbText, 30)
oTable.Fields.Append(oField)

oDB.TableDefs.Append(oTable)
```

CreateAccessAutomation.PRG, included on this month's PRD, demonstrates the entire technique by creating an Access database and adding two simple tables to it. It also documents the entire list of data types by defining constants for them.

The second approach to creating an Access database doesn't require Access to be on the computer. It uses OLE DB, ADO and ADOX (ADO Extensions), so the appropriate libraries must be present and the OLE DB provider for the Jet engine must be available. Another benefit of this approach is that, by specifying a different OLE DB provider, the same code can create another type of database.

To create a database, instantiate an ADOX Catalog object and then call its Create method. The hardest part is setting up the appropriate connection string for the Create method. This code demonstrates the necessary string for an Access database:

```
oCatalog = CREATEOBJECT("ADOX.Catalog")
cConnString = "Provider=Microsoft.Jet.OLEDB.4.0;" + ;
              "Jet OLEDB:Engine Type=5;" + ;
              "Data Source=AccessTest.MDB"
oCatalog.Create(cConnString)
```

Once the database exists, you can manipulate it with an ADO Command object. To work on the database you just created, point the Command object's ActiveConnection property to the connection the Catalog object is using:

```
oCommand = CREATEOBJECT("ADODB.Command")
oCommand.ActiveConnection = oCatalog.ActiveConnection
```

From this point, it gets pretty easy. You set the Command object's CommandText property to the appropriate SQL command and then call the Execute method. Here's the code to create the same Customer table as in the earlier example:

```
oCommand.CommandText = ;
  "CREATE TABLE Customer (iID INTEGER, Company TEXT(30))"
oCommand.Execute()
```

The only tricky part is knowing what field types to use. Table 2 shows the mapping between some VFP field types and the Access types.

Table 2. Field types for ADO—To create an Access table via ADO, you need to use the appropriate SQL field types.

VFP type	**ADO/SQL type**
Character	Text
Memo	Text
Logical	Bit

VFP type	**ADO/SQL type**
Date, DateTime	DateTime

This month's PRD includes CreateAccessADO.PRG, which creates a new Access database and adds very simple Customer and Orders tables.

Whichever approach you take, in an application, you're likely to want to create tables that match existing VFP tables. Use the AFIELDS() function to put the structure of the VFP table into an array. Then you can loop through the array to create the necessary field objects or field definitions.

Both approaches also let you copy data from VFP tables to the new Access tables. In the Automation approach, you can use a RecordSet object to hold data for one record at a time. In the ADO approach, you can build an INSERT command and then execute it.

–Tamar