ActiveX Controls Technical Articles

ADODB: ActiveX Data Objects 2.1

Kenneth Lassesen
Microsoft Corporation

March 1, 1999

Summary: Object model and notes for ActiveX Data Objects 2.1. (7 printed pages)

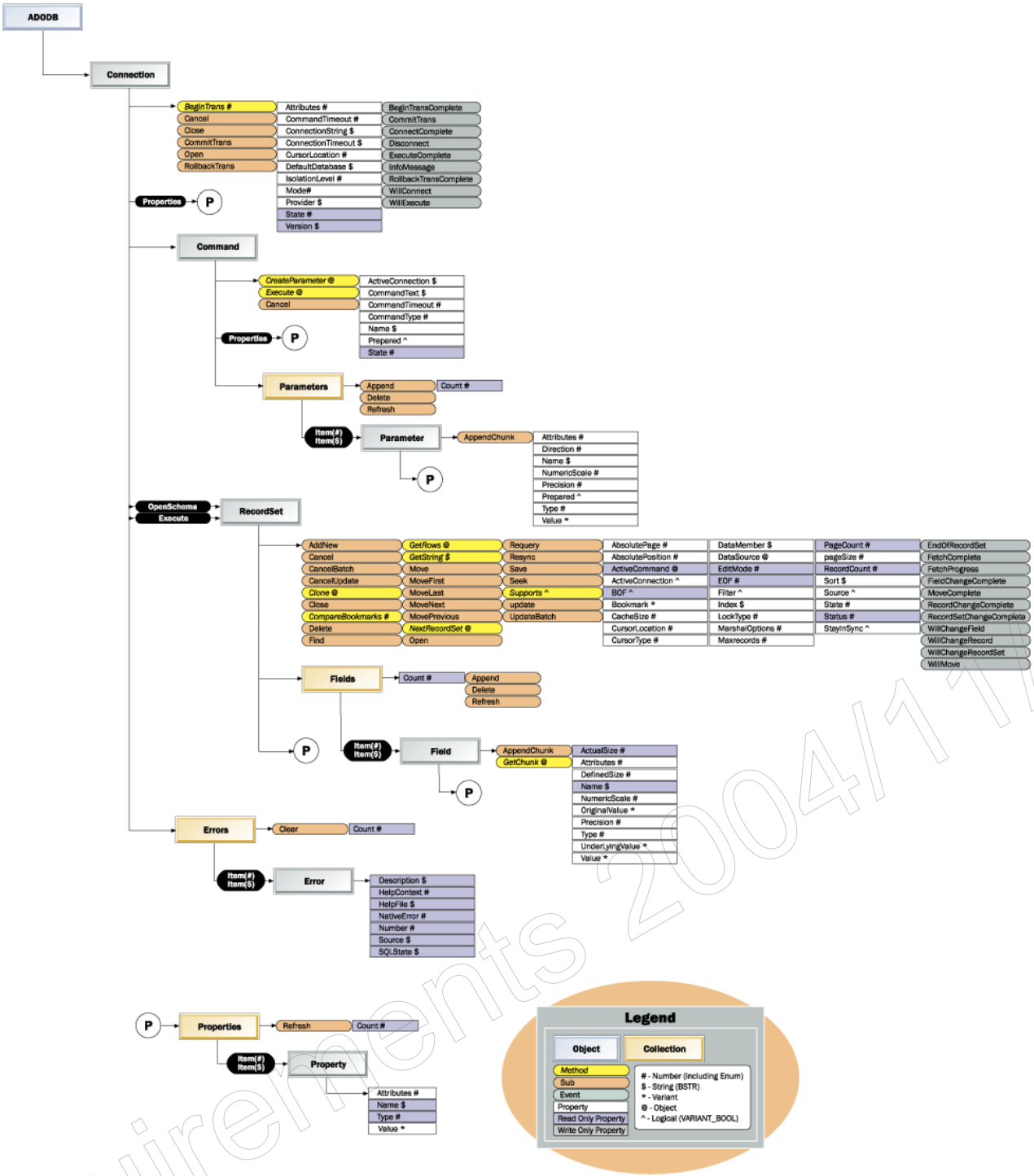

Figure 1. ActiveX Data Objects 2.1 object model

Microsoft® ActiveX® Data Objects (ADO) provide access to a rich variety of data sources through an OLE DB Provider. Typically, the OLE DB Provider is the OLE DB Provider for ODBC Drivers, effectively granting access to any data source having an ODBC Driver.

ADO is my preference for a data access mechanism and is Microsoft's recommended choice for future application development. Older technologies such as Remote Data

Objects (RDO) and Data Access Objects (DAO) are DOA (dead on arrival) for any new development I do.

ADO recordsets are very similar to RDO/DAO recordsets with improvements and simplifications. A few of them are:

- No need to call an .edit method to start editing
- The ability to move a recordset from a server to a client as a file using the lightweight, client-side ADOR object (which lacks Connection and Command objects of ADODB)
- Faster execution than older technologies
- Less collections
- Support for events on the Connection object, simplifying coding

Most developers are very familiar with ADO's ancestors, DAO and RDO. A few items of note:

Cancel applies to asynchronous Execute and Open; an error will occur if called otherwise.

Recordset.Save writes the recordset to a physical file. No more writing out the data as text!

MarshalOptions controls whether a client-side recordset sends back all records or only the modified records.

PageSize, PageCount, AbsolutePage are ideally for Web sites. They allow you to request the 123rd page of records where there are 63 records on a page—no more counting through records.

Recordset.StayInSync is part of the data shaping available in ADODB. Version 2.1 adds grandchild aggregates, reshaping and parameterized commands using COMPUTE to the earlier data shaping, and hierarchical recordsets of version 2.0.

Using ADODB on the Web

ADO may be used in many environments, one of the most popular of which is Internet Information Services (IIS) Active Server Pages (ASP). To display a recordset in an HTML page from an SQL statement and a DSN requires just a few lines of generic code, as shown here:

Copy Code

```
 <!– #Include file="adovbs.inc" –>
<SCRIPT RUNAT=SERVER LANGUAGE=VBSCRIPT>
Sub SQL2Table(strSQL, strDSN) `A Generic Function
Dim i
Set rs=Server.CreateObject("ADODB.Recordset")
rs.Open strSQL,strDSN
Response.Write "<TABLE BORDER=1>"
`Write out Field Names
Response.Write "<TR>"
For i=0 to rs.fields.count-1
Response.Write "<TH>"+rs(i).Name+"</TH>"
Next
Response.Write "</TR>"
`Write out Data
Do while not rs.eof
     Response.Write "<TR>
For i=0 to rs.fields.count-1
          Response.Write "<TD>
          Response.Write rs(i)
          Response.Write "</TD>
Next
     Response.Write "</TR>
     rs.movenext
Loop
Response.Write "</TABLE>"
End Sub
</SCRIPT>
```

The "adovbs.inc" file supplies the ADO constants because they are not intrinsically available in an .asp file. All of the "Response.Write" lines are outputting HTML into a Web page (with automatic conversion of data types).

If we put the preceding into an include file, a Web site may be coded with simple pages as shown in the following six lines:

Copy Code

```
<HTML><BODY>
<!– #Include file="SQL2TABLE.inc" –>
<%
SQL2TABLE "Select Item,Name, Price from MyCatalog Order by Name","DSN=DogFood"
%>
</BODY></HTML>
```

The execution of stored procedures in ADO is also simple. We start by creating a set of simple subroutines to make the actual code simpler to read and debug, and save the amount of typing that must occur.

Copy Code

```
<!– #Include file="adovbs.inc" –>
<SCRIPT RUNAT=SERVER LANGUAGE=VBSCRIPT>
</SCRIPT>
function spX(spName,strDSN)
dim cnn1, cmdSP
     ` Open connection.
     Set cnn1 = Server.CreateObject("ADODB.Connection")
     cnn1.CursorLocation = adUseClient
     cnn1.CommandTimeout=600
     cnn1.Open strDSN
     ` Open command object
     Set cmdSP= Server.CreateObject("ADODB.Command")
     cmdSP.CommandText = spName
     cmdSP.CommandType = adCmdStoredProc
     cmdSP.ActiveConnection = cnn1
     SET spX= cmdSP
end function
sub spRet(sp)
sp.Parameters.Append x.CreateParameter("RETURN_VALUE", adInteger, adParamReturnValue)
end sub
sub spVarChar(sp,parm,pSize,pValue)
sp.Parameters.Append sp.CreateParameter(parm, adChar, adParamInput, pSize)
sp.Parameters(parm) = pValue
end sub
sub spInt(sp,parm,pvalue)
sp.Parameters.Append sp.CreateParameter(parm, adInteger, adParamInput)
```

```
sp.Parameters(parm) = pvalue
end sub
</SCRIPT>
```

These procedures may then be included in any ASP page, resulting in simple clean code that is easy to read and understand:

Copy Code

```
      <HTML><BODY>
<!– #Include file="MyADOLibrary.inc" –>
<%
Set MySP=spX("ui_MySP", "DSN=DogFood"
      spRet MySP      `A returned value will occur
      spInt  MySP, "@MyAge", UserAge
      spVarChar MySP,"@MyName",40,UserName  '40 character Name
      set rsSP=SP.execute `execute the stored procedure
```

Because stored procedures may return multiple recordsets, we need to handle them. Assuming we wish to display all of the recordsets as HTML, the following code suffices:

Copy Code

```
      Do until rsSP Is Nothing
            RS2TABLE rsSP
            rsSP=rsSP.NextRecordSet
      Loop
      %></BODY></HTML>
```

RS2Table is almost identical to the SQL2TABLE shown, except it takes a results set in as an argument instead of just using an SQL string and a DSN string.

Some Personal Experiences

My first exposure to ADO was helping my wife with her pet Web site on an IIS where she was developing a pedigree database for Welsh Corgis (http://corgi.folkarts.com/pedigree/). First, I discovered that the administrators did not have to create a DSN for her Microsoft Access database; instead, I opened it with the following code:

Copy Code

```
Set DogConn=CreateObject("ADODB.Connection")
DogConn.Open "Driver=Microsoft Access Driver (*msb);DBQ=C:\Corgi\DogTree.Mdb"
```

If she wanted to use another file-based database system like Microsoft Visual FoxPro®, it is just as simple. If I wished to fine-tune the connection, it is trivial:

Copy Code

```
DogConn.Open = "DRIVER={Microsoft Access Driver (*.mdb)};User Id=admin; DBQ=C:\Corgi\DogTree.Mdb; DefaultDir=C:\Corgi;
FIL=MS Access; ImplicitCommitSync=Yes; MaxBufferSize=512; MaxScanRows=8; PageTimeout=5; SafeTransactions=0;
Threads=3; UserCommitSync=Yes;"
```

What impressed me was the speed of ADO—she could display the known pedigree of our dogs back 12 generations (a 2^{11} branch tree!) in seconds using an Access database as the source. (Try her Web page yourself!)

When I started developing industrial-grade Web applications going against Windows NT and SQL Server, I found that the use of system DSNs often leads to days tracking down setup errors on test boxes. The most common error was getting "GetOverLappedResult()" when accessing Windows NT and SQL Server from an IIS server. The cause was the DSN being set up (or later changed) to use Named Pipes to connect to Windows NT and SQL Server and not TCP/IP. The solution was to use the following string to set up the connection with the appropriate network connection:

Set MyConn=CreateObject("ADODB.Connection")

Copy Code

```
MyConn.Open " driver={SQL Server}; server=corwyn; database=family;  UID=Woof; PWD=Bone; network=dbmsscon"
The `network=dbmsscon" parameters determines the network library to use, in this case TCP/IP.
```

Conclusion

ADO is a sweet, simple tool that allows access to many data sources. If you are (or are becoming) a Web developer you will find that it is an indispensable tool for creating Web sites that perform well and scale to many users.