Getting Started: SAP® Sybase® Advantage Database Server with Visual Objects and Vulcan.NET

INTRODUCTI ON

This guide was written for DevShare 2010, a VO and Vulcan user group meeting held in Shrewton and organized by Phil Hepburn (http://www.clickstartvulcan.net) and Alwyn Jennings-Bramly (honored author of the VO and Vulcan bible).

Prerequisites

For Chapter 1 you need to install Visual Objects (minimum supported Version is VO 2.7, I'm using VO2.8SP2) and the Advantage RDD for Visual Object (minimum Version 8.1), available from the Advantage Developer Zone at http://devzone.advantagedatabase.com.

Chapter 2 uses Vulcan (a Snap-In for Visual Studio bringing the XBase-language into the .NET world) and Visual Studio. For supported versions, please visit the Vulcan homepage at http://www.govulcan.net. The minimum build should be 159 since this version added the ability to alias an RDD (which makes it easier to transport the application from Visual Objects to Vulcan). At the time of writing this guide the actual build is 161, but I've used build 160 in conjunction with Visual Studio 2008. The Advantage RDD for Vulcan.NET Version 10 (beta) can be downloaded from the Advantage Developer Zone (link see above). Older Versions of this client kit are available from the Advantage newsgroups.

All of the Advantage Client Kits contain the free of charge Advantage Local Server. To work with Client/Server you need to install Advantage Database Server on your file server. The files can be downloaded from the Advantage Developer Zone, licenses and test codes are available from the Advantage Sales Team (email: ADS-Team@sap.com). The minimum version you need is the highest version of the client being used since Advantage Servers are backwards compatible to older client versions but cannot be connected from newer clients.

A useful utility to create and manage Advantage Databases is the Advantage Data Architect, also available as free download from the Advantage Developer Zone.

The most important tool I've used to get started with Vulcan.NET was Red Gate's .NET Reflector (available at http://reflector.red-gate.com) and Fabrice Foray's fabulous Vulcan Reflector add-In (available via Fabrice's homepage at http://www.fabtoys.net).

CHAPTER 1: ADVANTAGE IN THE VO WORLD

Why use Advantage with Visual Objects?

SAP Sybase Advantage Database Server has its roots in the old Clipper world. The first version of Advantage was just a tool running on a Netware Server which allowed re-indexing and packing DBF tables with NTX indexes directly on the server. This speeded up the nightly rebuilds of the database a lot. When the RDD concept became available for Clipper, Extended Systems developed an RDD to communicate from the clipper application with the Advantage NLM (Netware Loadable Module) on the server bringing Client/Server to Clipper. Later on this concept was enhanced to support FoxPro's CDX indexes as well. Over the years Advantage became a full blown relational Client/Server Database Management System with a SQL interface and its own storage format (ADT), but still supporting DBF files from all of the interfaces. In Advantage 9 a new storage engine was added, called VFP. This allows working on Visual FoxPro's latest DBF format (VFP 9), bringing a new vision to VFP developers after Microsoft announced to discontinue VFP.

Natively supporting DBF files makes Advantage a natural choice for Visual Objects developers who want to bring their application into a new decade without the need of changing massive parts of their existing application or database. Even parallel access to the DBF files from legacy applications and new, Advantage enabled applications is possible.

Sharing DBF Tables between Advantage enabled applications and legacy VO applications

Advantage supports two locking modes. With proprietary locking, the default, Advantage loads data files exclusively so that no other applications are allowed to work on these files at the same time. This locking mode is the fastest, since Advantage handles all record locks in memory, yet is also the most stable, since no other process can write (and corrupt) the files. If you need to share DBF data files with non-Advantage applications, you can set the locking type to compatible. In compatible mode, Advantage works just like a legacy VO application opening the files in shared mode. All record locking is done on file level, thus multiple processes are allowed to work on the physical files at the same time. Locking type can be chosen in Advantage enabled applications on table level, passing an option to the AdsOpenTable API. When using the Advantage RDDs in Visual Objects (or Vulcan), a call to the extended function AX_AXSLocking(.T.) will set the work area to use proprietary locking for all subsequent table opens, passing .F. to that function will select compatible locking. The following image demonstrates parallel access to a DBF table from VO using DBFCDX RDD and from Advantage Data Architect (compatible mode).

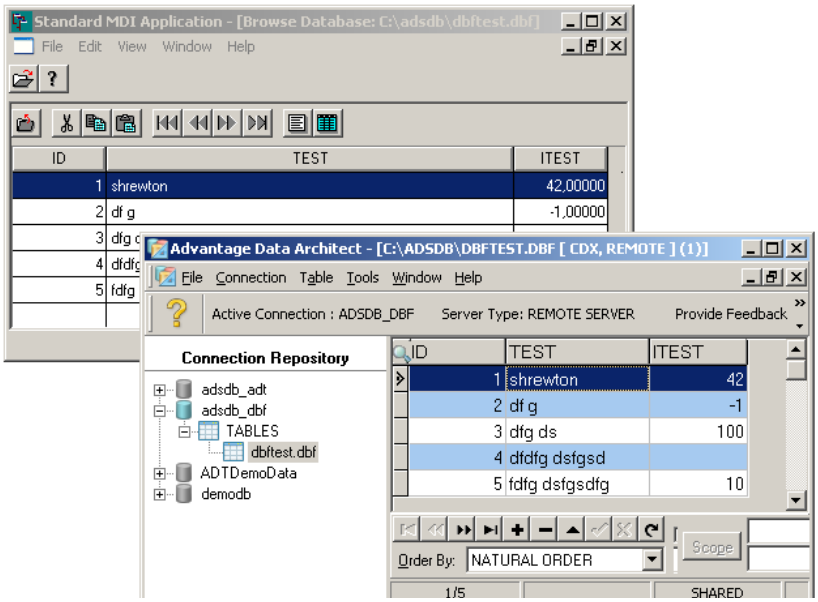

Image 1: Parallel access from Advantage and VO's DBFCDX

Modify VO applications to use Advantage

As a first example, we'll use the MDI example above (based on the DBFCDX RDD) and switch it over to use Advantage. For that purpose, you just need to set the default RDD to Advantage's AXDBFCDX. Open the project in Visual Objects, search for the Start() method and change it to:

```
METHOD Start() CLASS App
    LOCAL oMainWindow AS StandardShellWindow

    //RddSetDefault( "DBFCDX" )
    // Set the default RDD to Advantage
    // For CDX files, use AXDBFCDX
    RddSetDefault( "AXDBFCDX" )

    SetAnsi(.F.)
    SetExclusive( FALSE )
    SetDeleted( TRUE )

    oMainWindow := StandardShellWindow{ SELF }
    oMainWindow:Show( SHOWCENTERED )
    SELF:Exec()
```

Remark: Advantage will handle OEM/ANSI conversion internally, so you always need to disable VO's ANSI conversion by calling SetAnsi(.F.).

Introduction to the Advantage VO RDDs

The Advantage RDDs for Visual Objects are wrappers around the Advantage Client API to ease using Advantage for Visual Objects developers. There are four basic RDDs available:

- AXDBFCDX.RDD to access FoxPro compatible files (DBF with CDX indexes, up to Dbase III+)
- AXDBFNTX.RDD to access Clipper compatible files (DBF with NTX indexes)
- ADSADT.RDD to access tables in Advantage's proprietary ADT format
- AXDBFVFP.RDD for enhanced DBF tables up to Visual FoxPro 9's format

These RDDs provide direct, navigational table access and can be used in the same ways you can use native VO RDDs like DBFCDX: As a parameter to DBServer's Init method, in DBServer Editor, as Default RDD for the application (RDDSetDefault), in non-object oriented code (USE ... VIA).

By default, these RDDs use the connection settings provided by the ACE (Advantage Client Engine). If you need to use different settings, you need to call Advantage's advanced functions.

Advantage advanced functions (AX_)

Advantage advanced functions are a set of functions provided via the library DBFAXS.AEF that wrap ACE API calls into VO functions and influence the current work area. To use these functions, you need to add that library to your project. Examples for these functions are AX_AXSLocking() to change the locking behavior of Advantage RDDs (.T. **—>** proprietary locking, .F. **—>** compatible locking) and AX_SetServerType() to select the server types your application should use to connect to Advantage. Following code snipped can be used to access Advantage compatible Foxpro tables via the royalty free Advantage Local Server:

```
// Set the ADS server type
AX_SetServerType( .F. /*Remote*/, .F. /*Internet*/, .T. /*Local*/ )
// For CDX files, use AXDBFCDX
RddSetDefault( "AXDBFCDX" )
```

Using the Advantage Client API

Most likely you will not need to access the Advantage Client Engine (ACE) directly. On occasion, however, when developing Advantage-enabled Visual Objects applications, you may find that the functionality provided by the Advantage RDD is not exactly what you need. In this case, you need to add the library ACE.AEF to your project and make direct API calls.

The ACE uses handles to identify every connection, table, and index. The Advantage Visual Objects RDDs wrap the handle values. Therefore, you do not need to access the handles unless you plan to call the Advantage Client Engine directly. To retrieve the handle for a specific object, use the Advantage functions AX_GetAceTableHandle() or AX_GetAceIndexHandle(). These methods return table or index handles associated with the currently selected work area. Once these handles are obtained, they can be used to call Advantage Client Engine APIs directly. Following code snippet demonstrates the usage of the API:

```
// Adjust record caching to minimize network traffic
dwRetCode := AdsCacheRecords( AX_GetAceTableHandle(), 50 )
IF ( dwRetCode != AE_SUCCESS )
ErrorBox{ , "AdsCacheRecords failed with error: " + ;
Str( dwRetCode )}:Show()
ENDIF
```

The Advantage RDDs provide no method of connecting to an Advantage Data Dictionary. Thus you need to call the connect API explicitly in your code and set the connection handle being active for subsequent table opens:

```
pszConnectPath := "C:\adsdb\demodb\demodb.add"
ulRetVal := AdsConnect60( pszConnectPath, ADS_REMOTE_SERVER, ;
            String2Psz( "ADSSYS" ), NULL_PSZ, 0, @hConn )
IF ( dwRetCode != AE_SUCCESS )
     ErrorBox{ , "AdsConnect60 failed with error: " + ;
     Str( dwRetCode )}:Show()
ENDIF
AX_SetConnectionHandle(hConn)
```

Digression: Introduction to Advantage Data Dictionaries

The Advantage Data Dictionary (ADD) introduces additional features and functionality that compliment the Advantage Database Server and Advantage Local Server. With the Advantage Data Dictionary, a database can be clearly defined with its associated tables and indexes. Access to the database tables can be more securely guarded by the Advantage servers because users and user groups can be defined in the database and specific rights can be assigned to the users and user groups. The Advantage Data Dictionary allows the Advantage server to ensure the logical validity of the data in the database through the use of field level constraints, record level constraint and referential integrity with ADT tables. The Advantage Data Dictionary also supports

the use of stored procedures. Descriptions of the database, tables, fields, indexes, and default field values can be stored in the Advantage Data Dictionary to allow developers to develop and deploy applications more efficiently. Much of the Advantage Data Dictionary functionality is available to both ADT and DBF tables.

Advantage Data Dictionaries consist of three files: The file with the extension ADD is the dictionary itself, the AM file contains the binary data and the AI file contains the indexes for faster access to the system tables. Contrary to most relational databases, Advantage ADDs still keep the table, index and memo data of the associated tables in separate files.

More information about Advantage Data Dictionaries can be found in the Advantage documentation.

Using SQL queries in VO

Advantage by nature is an ISAM (indexed sequential accessing method) server. But it also provides a SQL interface into the supported data files. SQL is available through the ACE and supported by the entire ACE based clients. In Visual Objects there are different ways to use SQL against Advantage. The first is to use standard drivers like ADO/OLE DB (e.g. via Vo2Ado) or ODBC, the second to use the Advantage API. In more recent versions (beginning with Advantage 9) the Advantage RDDs for Visual Objects client kit contains RDDs to wrap SQL queries into ISAM like data access. There's a SQL RDD for each of the supported table types available.

Instead of passing a table name into these RDDs you pass in a SQL query. Since there is no connection path available in SQL, you first need to connect to the database and set the connection handle of the active work area to that connection.

The AdsSQLServer class is a class library that inherits from the DBServer class, with some additional functionality specific to the AXSQL RDDs. The majority of its methods and properties are exactly the same as the DBServer class. Although the AdsSQLServer class isn't required to use the AXSQL RDDs, only the most basic functionality will be available without it. Therefore it is highly recommended that it is always used with the AXSQL RDDs. Following code snippet demonstrates how to connect to a database and query the demo.dbf table using an AdsSQLServer and the Advantage AXSQLCDX RDD:

```
FUNCTION SelectTest()
LOCAL oDB AS AdsSQLServer
LOCAL hConn AS DWORD
LOCAL dwRet AS DWORD
LOCAL i AS INT

dwRet := AdsConnect60( String2Psz( "C:\ads_db" ), ADS_REMOTE_SERVER, ;
                  NULL_PSZ, NULL_PSZ, 0, @hConn )
IF dwRet <> 0
    AdsShowError( "" )
    RETURN
ENDIF

AX_SetConnectionHandle( hConn )

oDB := AdsSQLServer{ "SELECT * FROM demo", , , "AXSQLCDX" }
IF IsNil( oDB ) .OR. !oDB:Used
    AdsShowError( "" )
    RETURN
ENDIF

DO WHILE !oDB:EoF
    ? ""
    FOR i:=1 TO oDB:FCount
    ?? AsString( oDB:FIELDGET(i))
    ?? ", "
   NEXT
   oDB:Skip( 1 )
ENDDO
```

```
        oDB:Close()
        oDB := NULL_OBJECT

        AdsDisconnect( hConn )
        AX_SetConnectionHandle( 0 )

    RETURN
```

Distributing Advantage enabled VO applications

There's no need to install any Advantage Client on end customer side. It is enough to ship the used RDD and the Advantage Client Engine files along with the application. The core client engine is the ace32.dll. To access Advantage Database Server, the communications redirector axcws32.dll needs to be shipped. For Advantage Local Server based applications, you need to ship adsloc32.dll, adslocal.cfg and the collation files ansi.chr and extend.chr. When using dynamically loaded collations with Advantage Local Server, you also need to ship adscollate.adt and adscollate.adm.

CHAPTE R 2: ADVANTAGE AND VULCAN

Advantage being the natural choice for Visual Objects developers going forward also obliges SAP to provide an interface to Advantage for Vulcan.NET development.

Using the Advantage Vulcan RDDs

Vulcan.NET in its native mode provides a similar database concept as Visual Objects to provide compatibility with legacy code. Advantage provides all of the RDDs mentioned before for Vulcan.NET. To use these RDDs, following steps are necessary:

1. Add the AdvantageRDD assembly to the reference list of your project 1. (to avoid any conflicts you may set its CopyLocal property to true)
2. Load the assembly on application start
3. Modify the RDD calls to the fully qualified name (e.g. Advantage.ADSADT)

Following code snippet demonstrates steps 2 and 3:

```
System.Reflection.Assembly.Load("AdvantageRDD")
RddSetDefault("Advantage.ADSADT")
```

Beginning with Vulcan build 159 you can also register RDDs. With that approach you can bypass steps 2 and 3 above and use an alias (in our case ADSADT) instead of the fully qualified name.

```
VoDbRegisterRdd("ADSADT",TypeOf(Advantage.ADSADT))
RddSetDefault("ADSADT")
```

Calling Advantage Client Engine API and Advanced Functions in Vulcan.NET

The Advantage Client Engine (library ace) and the Advanced Functions (library DBFAXS) are already part of Advantage's RDD implementation. To use functions of these libraries, you need to add following line to your code:

```
#using AdvantageClientEngine
```

After that call, you can simply call ACE and AX_ functions by specifying their fully qualified name:

```
local hConn as intptr
local ulRetVal as DWORD
local connpath as string
connpath:="c:\ads db\demodb\demodb.ADD"
ulretval:=ACE.AdsConnect60( connpath,ACE.ADS_REMOTE_SERVER, ;
     "adssys", NULL, (DWORD)ACE.ADS_DEFAULT, hConn)
if ulretval=ACE.AE_SUCCESS
```

```
    self:textbox1:text += Environment.NewLine+"connection successful"
    Advantage.DBFAXS.AX_SetConnectionHandle(hConn)
  else
    self:textbox1:text += Environment.NewLine+"connection failed with errorcode
  "+ulretval:ToString()
    endif
    ulretval:=ACE.AdsDisconnect(hConn)
    Advantage.DBFAXS.AX_SetConnectionHandle(IntPtr.Zero)
```

Transporting Advantage enabled applications to Vulcan

Once Advantage enabled applications compile without errors and warnings in Visual Objects 2.8, you can use the transporter utility shipped with Vulcan.NET to migrate them to Vulcan.NET. How to use transporter is described in detail in Paul Piko's *Vulcan.NET At Warp Speed Guide*. For Advantage you just need to add the steps from the previous topic — that's it!

Using ADO.NET and the Advantage .NET data provider

Example: ADO.NET Vulcan Browser

In my opinion, using Vulcan.NET and the Vulcan.NET RDDs is a perfect first step into .NET development for Visual Objects developers, but only a first step. For further development as well as for new projects you might consider using .NET's recommended data access layer ADO.NET.

ADO.NET works completely different than RDDs. This framework consists of classes for a disconnected in-memory database with client side cursors only. The main data provider classes are a connection class (e.g. sqlConnection) to handle a database connection, a command class (e.g. sqlCommand) to execute SQL commands and a data reader class (e.g. sqlDataReader) to read the result of a command. As a kind of super class there's the data adapter class (e.g. sqlDataAdapter) which provides a connector from the SQL database to the in-memory database (DataSet). The data adapter class also handles tracing data modifications and thus provides an easy way to send updates back to the server. All of these classes are database dependant, i.e. there's a sqlConnection class to handle connections to Microsoft's SQL Server, oraConnection for Oracle, and so on. Advantage has its own set of compatible classes contained in the Advantage .NET data provider, supporting all of the data provider specific functionality, including Entity Framework and model first support.

As a first example in using ADO.NET, create a new Vulcan.NET Windows Forms Application project in Visual Studio and add two Buttons, a DataGridView a Dataset (untyped DataSet) and an AdsDataAdapter to it. Adding the AdsDataAdapter runs a wizard to set up the connection path and select the required tables (see Image 2).

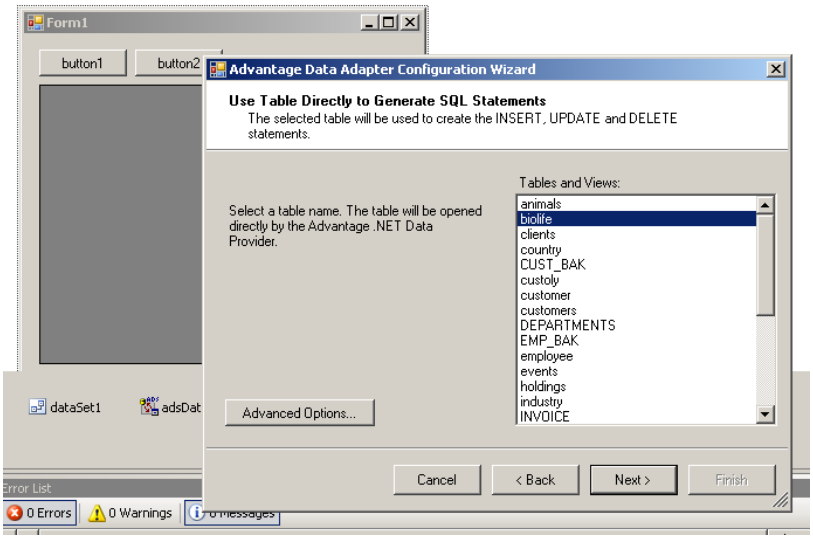

Image 2: ADSDataAdapter Wizard

In the first example, I just use one table called biolife (it's shipped as part of Advantage's demo databases). To load the data of that table into the (in-memory!) dataset, double click the first button and insert following code into the click event:

```
PRIVATE METHOD button1_Click( sender AS System.Object, ;
   e AS System.EventArgs ) AS System.Void
  SELF:dataSet1:Clear()
   SELF:adsDataAdapter1:Fill(dataSet1,"biolife")
   SELF:dataGridView1:DataSource:=dataSet1
   SELF:dataGridView1:DataMember:="biolife"
RETURN
```

If you now compile and run the project, you'll get the data loaded from the database on button click. You can also edit the data, but remember that it's completely done in memory! To get the changes back to the server, you need to do it explicitly in code, e.g. on a click on the second button:

```
PRIVATE METHOD button2_Click( sender AS System.Object, ;
   e AS System.EventArgs ) AS System.Void
   SELF:adsDataAdapter1:Update(dataSet1,"biolife")
RETURN
```

Remark: The DataAdapter classes handle insert/update and delete tracking automatically, but if there were any modifications to the same records from a different instance of the application, you'll get an exception and need to handle it in code. This is the concurrency issue common to all flavors of the disconnected model.

The main differences between navigational data access and disconnected datasets

As you've seen, disconnected datasets completely changes how to work with data. There are some advantages over ISAM access using such an approach, like update of static cursor result sets, universal data access, customized control of constraints and relationships and a simpler migration towards a different backend. But there are also a lot of disadvantages, like complete data being loaded into client memory, schema changes requiring application changes, lack of record locking, lack of visibility in multi user environments and so on. But if there's a need for ISAM data access, there's no real solution available in pure ADO.NET.

Using ADSDataReader to navigate through a result set

Example: DataReader

One approach to get at least some kind of navigational access is to use a data reader. Data readers are read only and forward only cursors to the result set of a SQL command. They can be used to easily fill lists (object lists, list boxes) and are also used internally in the Data Adapter classes to pump the data into datasets. Following example shows how to use an AdsDataReader to access data. In the constructor, a command is being used to create a data reader:

```
CONSTRUCTOR()
   local cmd as AdsCommand
   SUPER()
   SELF:InitializeComponent()
   SELF:cn := AdsConnection{"Data Source = C:\ADSDB\ads_data; "+ ;
                            "TrimTrailingSpaces = TRUE"}
   SELF:cn:Open()
   cmd := SELF:cn:CreateCommand()
   cmd:CommandText := "select * from biolife order by [species no]"
   cmd:CommandType := System.Data.CommandType.Text
   SELF:rdr := cmd:ExecuteReader()
RETURN
```

To get the first visible record (on opening it points to BOF), you need to call the method Read(). If this method returns false, there are no more visible records available (EOF). To access the single fields of a record, data readers usually implement an indexer.

```
self:rdr:Read()
try
  self:species_NoTextBox:Text := self:rdr:Item[“Species No”]:ToString()
  self:SpeciesNo := System.Convert.ToInt32( ;
                        self:rdr:Item[“Species No”]:ToString())
  self:categoryTextBox:Text := self:rdr:Item[“Category”]:ToString()
  self:common_NameTextBox:Text := self:rdr:Item[“Common_Name”]:ToString()
  self:species_NameTextBox:Text := self:rdr:Item[“Species Name”]:ToString()
  self:length__cm_TextBox:Text := self:rdr:Item[“length (cm)”]:ToString()
  self:length_InTextBox:Text := self:rdr:Item[“length_in”]:ToString()
  self:notesTextBox:Text := self:rdr:Item[“notes”]:ToString()
catch obj1 as Exception
  self:species_NoTextBox:Text := “0”
  self:SpeciesNo := 0
  self:categoryTextBox:Text := “<NULL>”
  self:common_NameTextBox:Text := “<NULL>”
  self:species_NameTextBox:Text := “<NULL>”
  self:length__cm_TextBox:Text := “0”
  self:length_InTextBox:Text := “0”
  self:notesTextBox:Text := “<NULL>”
end try
```

After editing a record, you need to send the update back to the database via a SQL command:

```
local cmd as AdsCommand
local param as AdsParameter
cmd := self:cn:CreateCommand()
cmd:CommandText := “MERGE [biolife] ON ([Species No] = :p1) “+ ;
                   “ WHEN MATCHED THEN UPDATE SET [Category] = :p2, “+ ;
                   // all other fields
                   “ WHEN NOT MATCHED THEN INSERT ([Species No], “+ ;
                   // all other fields
                   “ ) VALUES (:p11,:p12,:p13,:p14,:p15,:p16,:p17)”
param := cmd:CreateParameter()
param:ParameterName := “p1”
param:Value := System.Convert.ToInt32(self:species_NoTextBox:Text)
self:SpeciesNo := System.Convert.ToInt32(self:species_NoTextBox:Text)
cmd:Parameters:Add(param)
// do this for all other parameters aswell
cmd:Parameters:Add(param)
cmd:ExecuteNonQuery()
```

Deleting a record is a bit simpler:

```
local cmd as AdsCommand
local param as AdsParameter
cmd := self:cn:CreateCommand()
cmd:CommandText := “delete from biolife where [species no] = :p1”
cmd:CommandType := System.Data.CommandType.Text
param := cmd:CreateParameter()
param:ParameterName := “p1”
param:Value := System.Convert.ToInt32(self:species_NoTextBox:Text)
cmd:Parameters:Add(param)
cmd:ExecuteNonQuery()
```

As you can see in this example, it could be very complicated to write a simple navigational example with pure ADO. NET classes. Even more since this example doesn't really navigate: a data reader can only be used to get read only, forward only access, in other words do a SKIP(1). Insert, update and delete require separate SQL calls, jumping to the previous record, BOF or EOF is not supported.

True navigation: AdvantageExtendedReader

Example: AdsExtendedReader

The Advantage Client Engine provides both, ISAM and SQL access to the underlying data — and the Advantage.NET data provider is based on that client engine. So why not simply enable these classes to make use of the powerful ISAM concept in .NET? Well, the basic data provider classes do use the ISAM concept to retrieve the data, but from a design standpoint it is not possible to modify these basic classes. So Advantage introduces a new class for true navigational data access: the AdvantageExtendedReader. This class is a descendant of AdsDataReader and provides additionalfunctionality:

- It's always connected , so you'll get actual data
- You can directly edit the underlying data, including pessimistic record locking
- It implements methods to skip to the previous record (ReadPrevious)
- It implements methods to jump to the first (AdsGotoTop) and last (AdsGotoBottom) record without reading through all of the records in between
- It implements properties to set the active index order instead of re-querying the same table with a different ORDER BY clause

This getting started guide is surely not the right place to discuss all of AdsExtendedReader's features, so I want to refer to the Advantage documentation. But to show how simple it is to use AdsExtendedReader instead of AdsDataReader to implement the same navigational example, here's the code. In the constructor, a command is being used to create a data reader, but this time it creates an AdsExtendedReader instead:

```
CONSTRUCTOR()
  local cmd as AdsCommand
  SUPER()
  SELF:InitializeComponent()
  SELF:cn := AdsConnection{"Data Source = C:\ADSDB\ads_data; "+ ;
                          "TrimTrailingSpaces = TRUE"}
  SELF:cn:Open()
  cmd := SELF:cn:CreateCommand()
  cmd:CommandText := "biolife"
  cmd:CommandType := System.Data.CommandType.TableDirect
  SELF:rdr := cmd:ExecuteExtendedReader()
  SELF:rdr:RecordCache := 0
  SELF:rdr:ActiveIndex := "Primary"
RETURN
```

On the navigation side you now can fully navigate through the result set:

```
do case
  case ( mode == - 1 ) self:rdr:ReadPrevious()
  case ( mode == 1 ) self:rdr:Read())
  case ( mode == 10 ) self:rdr:GotoBottom()
  case ( mode == -10 ) self:rdr:GotoTop()
endcase
```

Reading the data is no difference to the AdsDataReader, just use the indexer, but deleting doesn't require any additional SQL command, it shrank down to one method call:

```
self:rdr:DeleteRecord()
```

The real power of a writeable data reader can be seen when a modified record is to be saved:

```
try
  self:rdr:Item[“Species No”] := ;
    System.Convert.ToInt32(self:species_NoTextBox:Text)
  self:rdr:Item[“Category”] := self:categoryTextBox:Text
  self:rdr:Item[“Common_Name”] := self:common_NameTextBox:Text
  self:rdr:Item[“Species Name”] := self:species_NameTextBox:Text
  self:rdr:Item[“length (cm)”] := ;
    System.Convert.ToDouble(self:length__cm_TextBox:Text)
  self:rdr:Item[“length_in”] := ;
    System.Convert.ToDouble(self:length_InTextBox:Text)
  self:rdr:Item[“notes”] := self:notesTextBox:Text
catch ex as System.Exception
  System.Windows.Forms.MessageBox.Show(ex:Message)
finally
 try
   self:rdr:UnlockRecord()
   self:ReadRecord(0)
catch obj1 as System.Exception
    //exception handling here
  end try
end try
```

After updating the record, a call to UnlockRecord() releases the record lock (which is done on a textbox changed event via AdsExtendedReader’s method LockRecord).

SUMMARY

Using Advantage in Visual Objects or Vulcan.NET is really simple. Supporting the RDD concept ensures only minor changes to the source code are necessary and almost none required to the underlying data. The first step is easy; however there are also some incompatibilities that are not handled in this guide. The Advantage help file contains a chapter regarding dealing with such incompatibilities in the RDDs.

Going further with .NET and using ADO.NET as the data access mechanism, means changing the view to the databases a lot. You need to handle large amount of data being loaded into client side memory, deal with lack of record locks and concurrency and work with set based access instead of using ISAM. Data readers can help to streamline that process, but it’s still a lot of code to be written for the simplest tasks.

Advantage can help to fill that gap by providing basic ADO.NET data provider classes as well as an AdsExtendedReader class to get true navigational ISAM access in .NET.

WWW.SAP.COM