ROCK**-solid **E**ngine **T**echnology™ **(ROCK-E-T)

SuccessWare International Technical Paper

Product Description:
SuccessWare International's **ROCK**-solid **E**ngine **T**echnology™ ***(ROCK-E-T)*** provides Windows® programmers with high-speed, multi-user local access to the data and index files used by CA-Clipper® (.DBF/.DBT/.NTX), FoxPro® (.DBF/.FPT/.IDX/.CDX), as well as our super-fast, compact/compound HiPer-SIx® .NSX/.SMT index and memo file formats. Other features include index Scoping, Conditional Indexing, Sub-indexing, and record-level Data Encryption (non-DES).

Programming Languages Supported:
ROCK-E-T can be used with any Windows development language that supports standard Windows .DLLs and .VBX custome controls. These include Visual Basic™, Delphi™, Visual C++™, Visual FoxPro™, Borland C++™, dBASE™ for Windows v5.0, and PowerBuilder™.

Direct File Access:
By providing direct access to the data and index files, ***ROCK-E-T*** is able to outperform systems using "middle-ware", such as ODBC drivers. Instead, ***ROCK-E-T*** uses a high-speed Replaceable Database Engine (RDE) system, which allows your code to remain virtually unchanged, while selecting from any of the three ***ROCK-E-T*** data and index file formats.

Bound Data Controls (.VBXs):
ROCK-E-T includes bound controls (.VBXs) for data access and flexible browses, supporting input masks (like an Xbase PICTURE clause) and pre and post-validation of data. Display colors, formatting, and source of the data for individual columns in the browse are easily handled through the browse control's GetLine event.

Runtime License / Royalty Requirements:
ROCK-E-T has **no runtime license or royalty requirements**. You are free to distribute ***ROCK-E-T's*** .DLL and .VBX controls with your applications. The shareware version of **Visual Navigator**™ *(included)* can also be freely distributed with your ***ROCK-E-T*** applications.

.DLL/.VBX File Sizes:

ROCKET.DLL	(required for all ROCK-E-T applications)	59k
SXDBFNTX.DLL	(required only for CA-Clipper data/index support)	169k
SXDBFCDX.DLL	(required only for FoxPro data/index support)	189k
SXDBFNSX.DLL	(required only for HiPer-SIx data/index support)	189k
6BROW.VBX	(browse grid control)	78k
6DATA.VBX	(data edit control, combo box, list box, check box)	150k

Native Record/File Locking:
ROCK-E-T uses the file and record locking schemes native to the file format it's accessing (CA-Clipper, FoxPro, and HiPer-SIx). For example, if you open a FoxPro database with ***ROCK-E-T*** and update it concurrently with a FoxPro application, each application will respect the other's locks. Most ODBC engines, including JET™, use an incompatible *page* locking system.

Query Optimization:

To provide super-fast queries, SuccessWare's **Mach SIx**™ query optimization (similar to FoxPro's Rushmore™ technology) is built in. Mach SIx uses existing index files to determine the records meeting a query condition, in many cases, without the need to access the database file. The query result can be obtained up to 1000 times faster than conventional database filters.

Xbase-Like Syntax:

ROCK-E-T's syntax was designed to leverage the existing knowledge of Xbase programmers migrating to a Windows environment. With this in mind, each ***ROCK-E-T*** function name is based on the equivalent Xbase command or function name. For example, to open a database file, you call sx_Use(). To create a new index file, call sx_Index(). Append a new record with sx_Append(), search for a record in the active index with sx_Seek(), ZAP the database with sx_Zap(), and so on. Built-in Xbase functions allow you to continue using indexes created with Xbase-native functions like UPPER() and DTOS().

Memo File Enhancements:

ROCK-E-T's memo fields support storage of anything from text to graphic images and other Binary Large Objects (BLOBs). Windows bitmap (.BMP) images can even be displayed directly from within a ***ROCK-E-T*** memo field, without the use of any other add-on product. Memo file "packing" is also supported, eliminating the memo file bloat problem typical with all Xbase memo file formats.

Visual Navigator Utility:

ROCK-E-T also includes the **Visual Navigator** utility, which is a stand-alone Windows executable program that lets you open, browse, and edit any of the database and index file formats supported by ***ROCK-E-T***. **Visual Navigator** includes standard CUA menus, a toolbar of the most common DBMS operations, and a command-line interpreter, similar to an Xbase dot-prompt.

Documentation:

ROCK-E-T includes a 300 page manual with complete documentation on every ***ROCK-E-T*** feature, with samples for each function in both Visual Basic and Visual C++. Three comprehensive Windows help (.HLP) files are included instant reference during development.

Sample Applications:

ROCK-E-T includes complete source code for sample database applications written in Visual Basic, Delphi, and Visual C++. These can be used as a starting point for your own application, or as overall examples of how ***ROCK-E-T*** can be used in your existing application.

Automatic OEM / ANSI Conversion:

ROCK-E-T automatically translates European OEM character set databases and indexes into Windows ANSI and vice-versa. This allows you to run DOS based applications concurrently with Windows applications using the same data.

System Requirements:

If your PC can run Windows, it can use ***ROCK-E-T***. Nothing additional is required to use ***ROCK-E-T***, above and beyond the standard requirements of the Windows development language you are using.

Coding Examples:

Visual Basic Example:

```
Sub ButtonPack_Click ()
  If Not sx_Use("c:\vb\cust.dbf", "cust", EXCLUSIVE, SIXNTX) Then
     MsgBox "File in use. Try again later."
  Else
     iRet = sx_IndexOpen("c:\vb\sxcust1.ntx")
     iRet = sx_IndexOpen("c:\vb\sxcust2.ntx")
     iRet = sx_IndexOpen("c:\vb\sxcust3.ntx")
     sx_SetGaugeHook GaugeBox.hWnd
     GaugeFiles = 4
     sx_Pack
     sx_SetGaugeHook 0
     sx_Close
  End If
End Sub
```

MFC/C++ Example:

```
if (!sx_Use("c:\\vb\\cust.dbf", "cust", EXCLUSIVE, SIXNTX))
  AfxMessageBox((LPCSTR) "File in use. Try again later.");
else
  {
  sx_IndexOpen("c:\\vb\\sxcust1.ntx");
  sx_IndexOpen("c:\\vb\\sxcust2.ntx");
  sx_IndexOpen("c:\\vb\\sxcust3.ntx");
  sx_Pack();
  sx_Close();
  }
```

Delphi Example:

```
procedure TForm1.Button1Click( Sender : TObject );
begin
  If sx_Use('c:\data\cust.dbf', 'cust', EXCLUSIVE, SIXNTX) < 0 Then
     MessageDlg( 'File in use. Try again later.', mtInformation, [mbOk], 0 )
  else
     begin
        sx_IndexOpen('c:\data\sxcust1.ntx');
        sx_IndexOpen('c:\data\sxcust2.ntx');
        sx_IndexOpen('c:\data\sxcust3.ntx');
        sx_Pack;
        sx_Close;
     end;
end;
```

Fast Access to FoxPro and Clipper Tables in Visual Basic

by Cecilia Smith and Loren Scott

So you want to move that Xbase application to Windows? If you're like many developers, you figured Visual Basic might be the quickest route. Your hopes may have been dashed when you began implementing Visual Basic's data methods on your FoxPro tables. Even if you made it past the data object syntax, you were in for a surprise when you realized that Visual Basic was largely intolerant of even the simplest Xbase functions in an index expression. UPPER(), for instance. And if you were counting on Visual Basic to read your Clipper index files, you were completely out of luck. Clipper .NTX files are not supported by Visual Basic at all.

This article will demonstrate a method for quickly getting your database application into Windows, without sacrificing what you already know in Clipper and FoxPro. We will step through a couple of the typical things developers like to do to databases. We will show you what Visual Basic native syntax would look like. And we will present an alternative method using SuccessWare's **ROCK**-solid **E**ngine **T**echnology (ROCK-E-T), which provides developers with a set of Xbase-like functions that talk directly to FoxPro and Clipper tables.

The function suite contained in ROCK-E-T's DLLs was designed to leverage the Xbase knowledge of programmers migrating to a Windows environment. Each ROCK-E-T function name is based on the equivalent Xbase command or function name. For example:

To open a database file	**sx_Use()**
To create a new index file	**sx_Index()**
Append a new record	**sx_AppendBlank()**
Search for a record in the active index	**sx_Seek()**
Pack the database	**sx_Pack()**

Let's take a closer look at the real code required for accomplishing these tasks.

For purposes of this example we have one Foxpro .DBF with an associated structural multi-tagged CDX index file. We will perform some of the simple tasks (simple in Xbase, that is!) data maintenance tasks, that most of us could do in our sleep under Clipper and FoxPro. In our example, we will open TEST.DBF, index it, delete a record and finally, we will take a look at what appears to be a daunting task in VB, packing a data table.

Open a Table

Sounds easy enough! But look at the VB code required. In fairness, there are good reasons for this, and VB is not just opening one "file". It is, in the case of Xbase tables, defining a set of files that make up the entire database.

```
Sub vbUse()
  Dim db As Database
  Dim test As Table
  Dim lname As New Index
  dim bExcl%, cRde$
  bExcl = true
  Set db = OpenDatabase(App.Path, bExcl, False, "FoxPro 2.5;")
  Set test = db.OpenTable("TEST")
end sub
```

Now in VB with ROCK-E-T:

```
sub rkUse()
  dim wa%
  wa = sx_Use(App.Path & "\TEST.DBF", "TEST", EXCLUSIVE, "SIXFOX")
end sub
```

In the ROCK-E-T example, **wa** is a unique number that can be forever associated with the table we just opened. It works much like a Clipper or Foxpro workarea. One welcomed difference, however, is that when you open the next workarea, you don't automatically close the one you just opened. Instead, a new, unique workarea number is returned, assuming there was no error opening the file.

Creating a Simple Index

Now that we have our table, let's build an index. Note that since we are using the FoxPro engine, an existing CDX files with the same root name as the DBF table would have been opened automatically under ROCK-E-T, just as it would under FoxPro itself. In Visual Basic, indexes named in a <table>.INF file would be opened automatically.

```
sub vbIndex()
  lname.Name     = "LNAME"
  lname.Fields   = "LAST"
  lname.Unique   = False
  lname.Primary  = False
  test.Close
  db.TableDefs("TEST").Indexes.Append lname
  Set db = OpenDatabase(App.Path, bExcl, False, cRDE)
  Set test = db.OpenTable("TEST")
  test.Index = "LNAME"
end sub
```

The equivalent ROCK-E-T code would be:

```
sub rkIndex()
  dim iOrd as integer
  iOrd = sx_IndexTag( 0&, "LNAME", "LAST", 0, 0, 0& )
end sub
```

Certainly this is shorter. Arguably more intuitive. That is, it makes sense if you are used to the sx_IndexTag parameters. If you would like your code to be more explicit, you can create an user defined ORDER type that you could pass to the rkIndex function, making it more generic. Here's what it might look like:

```
Type order
  name As String
  expr As String
  unique As Integer
  desc as integer
  cond As String
End Type

Function ixCreate (ix As order) As Integer
       ' returns the number of the newly created order
       if Len(Trim$(ix.cond)) = 0 Then
       ixCreate = sx_IndexTag(0&, ix.name, ix.expr, ix.unique, ix.desc, 0&)
  Else
  ixCreate = sx_IndexTag(0&, ix.name, ix.expr, ix.unique, ix.desc, ix.cond)
  End If
End Function
```

sx_IndexTag() takes 6 parameters:

1) Index file name. When working with FoxPro files, you would most likely pass the Visual Basic NULL, that is **0&**. Then filename defaults to <table>.CDX.
2) A string that represents the identifying index TAG. (The same used in foxpro.
3) A string representing the Xbase expression on which the index is created. We will come back to this point later.
4) TRUE or FALSE to create an equivalent Xbase UNIQUE index.
5) TRUE or FALSE to create an equivalent Xbase DESCENDING index.
6) And finally, you may create Conditional indexes by passing a string representing an Xbase condition. For example, to get an index that shows only records with an AGE field greater than 30, you would pass "AGE > 30". In most cases, you won't have a condition, in which case you pass the VB NULL of **0&**.

Now, back to point #3, the index expression itself. The Visual Basic engine is set up to work properly only with very simple field-only indexes. Typical Xbase style indexes created with functions like DTOS(), upper(), STR() can not be counted on under Visual Basic itself.

Built-in Xbase functions allow you to continue using indexes created with Xbase-native functions like UPPER() and DTOS().

Creating Typical Xbase-Style Indexes

Here is an example that creates four tags. All of these are acceptable with the ROCK-E-T engine. Additionally, it is not required that they be created under ROCK-E-T. Foxpro and/or Clipper-created indexes can be read as well.

```
sub rkIndex()
  dim iLast%, iState%, iCond%, iDate%
  iLast = sx_indexTag  (0&, "last", 0, 0, 0&)
  iState% = sx_indexTag(0&, "upper(state)", 0, 0, 0&)
  iDate% = sx_indexTag(0&, "dtos(hiredate)", 0, 0, 0&)
  iCond% = sx_indexTag(0&, "last", 0, 0, "age > 20")
end sub
```

Packing the Table

Our final example is that of packing a database. The `compactDatabase` function in Visual Basic has no effect on Xbase tables. Performing the equivalent of a PACK requires several steps. A recent Microsoft Knowledge-Base article outlines how.

The first thing the article points to is that you must have the following in a VB.INI or <appname>.INI file:

```
[dBase ISAM]
Deleted=On
```

This is VB’s way of saying `SET DELETED ON`. With ROCK-E-T, the expression is:

```
sx_SetDeleted(TRUE)
```

So far, so good. Here’s the rest of the code excerpted from the Knowledge Base article:

```
Sub main ()
  Dim db As Database
  Set db = OpenDatabase("c:\jet\x", False, False, "foxpro 2.5")
  Call Pack_DBF(db, "test")
  db.Close
End Sub

Sub Pack_DBF (db As Database, tblname As String)
 Const MB_YESNO = 4                   ' Yes and No buttons
 Const MB_ICONEXCLAMATION = 48        ' Warning message
 Const IDYES = 6                      ' Yes button pressed

 Dim dbdir As String, tmp As String   'Temp variables
 Dim i As Integer, ret As Integer     'Counter and return value of MsgBox
 Dim flags As Integer                 'Flags for MsgBox
 ReDim idxs(0) As New index           'Holds indexes

 On Error GoTo PackErr
 flags = MB_YESNO Or MB_ICONEXCLAMATION
 ret = MsgBox("Remove All Deleted Records in " & tblname & "?", flags)
 If ret = IDYES Then
   dbdir = db.Name + "\"              'Hold database directory

   'Delete the temp file if it exists.
   If Dir$(dbdir & "p_a_c_k.*") <> "" Then
     Kill dbdir & "p_a_c_k.*"
   End If
   'Store the indexes.
   For i = 0 To db.TableDefs(tblname).Indexes.Count - 1
     ReDim Preserve idxs(i + 1)
     idxs(i).Name = db.TableDefs(tblname).Indexes(i).Name
     idxs(i).Fields = db.TableDefs(tblname).Indexes(i).Fields
     idxs(i).Primary = db.TableDefs(tblname).Indexes(i).Primary
     idxs(i).Unique = db.TableDefs(tblname).Indexes(i).Unique
   Next

   'Create the new table without the deleted records.
   db.Execute "Select * into [p_a_c_k] from " & tblname
   'Delete the current table.
   db.TableDefs.Delete tblname
   'Rename the DBF file and any memo files.
   tmp = Dir$(dbdir & "p_a_c_k.*")

   Do While tmp <> ""
     'Rename with the correct file extension; this should be on one line.
     Name dbdir & tmp As dbdir & tblname & Right$(tmp, Len(tmp) -
                                                InStr(tmp, ".") + 1)
     tmp = Dir$
   Loop

   'Refresh the tabledefs and add the indexes to the new table.
   db.TableDefs.Refresh

   For i = 0 To UBound(idxs) - 1
     db.TableDefs(tblname).Indexes.Append idxs(i)
   Next
   MsgBox "'" & tblname & "' successfully Packed!", MB_ICONEXCLAMATION
 End If
Exit Sub
PackErr:
 MsgBox Error$
Exit Sub
PackEnd:
End Sub
```

Wow! Here is the equivalent using ROCK-E-T:

```
Const MB_YESNO = 4                              ' Yes and No buttons
Const MB_ICONEXCLAMATION = 48                   ' Warning message
Const IDYES = 6                                 ' Yes button pressed

' ROCK-E-T code
Sub main ()
 Dim db%
 db = sx_Use("c:\jet\x\test", "TEST", EXCLUSIVE, SIXFOX)
 Call Pack_DBF(db, "TEST")
 sx_Close
End Sub

Sub Pack_DBF (db%, tblname$)
 Dim dbdir As String, tmp As String  'Temp variables
 Dim i As Integer, ret As Integer    'Counter and return value of MsgBox
 Dim flags As Integer                'Flags for MsgBox
 ReDim idxs(0) As New index          'Holds indexes
 sx_SetErrorHook (True)
 On Error GoTo PackErr
 flags = MB_YESNO Or MB_ICONEXCLAMATION
 ret = MsgBox("Remove All Deleted Records in " & tblname & "?", flags)
 If ret = IDYES Then
   sx_Pack
 End If
Exit Sub
PackErr:
 MsgBox Error$
Exit Sub
```

```
PackEnd:
 sx_SetErrorHook (False)
End Sub
```

Okay. Let’s give Visual Basic a tiny break here. Let’s say you that one of the things you learned in Xbase school is “thou shalt not PACK”. The reason for this school of thought is that, PACK means REINDEX, and REINDEX is “bad” because it pre-supposes non-corrupt index headers that contain the indexed expression. It’s still an easy an intuitive user-defined sub-routine for those with an Xbase background.. Here is the routine:

```
Sub Pack_DBF ()
       Dim ret As Integer
       Dim flags As Integer
       Dim dbName as String, ixName as String

       sx_SetErrorHook TRUE     ' hook into ROCK-E-T error messages
       On Error GoTo PackErr

       flags = MB_YESNO Or MB_ICONEXCLAMATION
       ret = MsgBox("Remove All Deleted Records in " & sx_alias(0) "?", flags)
       If ret = IDYES Then
    dbName = sx_baseName()     ' DOS file name for table file
    ixName = sx_indexName()    ' DOS file name for index file
    If Dir$("p_a_c_k.*") <> "" Then
      Kill "p_a_c_k.*"
    End If

    'Copy non-deleted records to a temp table.
    sx_SetDeleted TRUE   'sx_copyfile respects filters and DELETED setting
    If sx_copyFile( "p_a_c_k" )
      sx_close
      Kill dbName   ' Delete the current table.
      Kill ixName   ' Because we're going to re-create the index
      Name “p_a_c_k.Dbf” As dbName
      rkUse()       ' open again
      rkIndex()     '  create indexes
    End If
    MsgBox "'" & dbName  & "' successfully Packed!", MB_ICONEXCLAMATION
       End If
       Exit Sub
PackErr:
       MsgBox Error$
       Exit Sub
PackEnd:
end sub
```

Note that the Knowledge Base routine saves the index information before the PACK. We do not do that in this example because the only reason we would be doing a PACK the long way might be because we’re fanatical about not depending on current index header info for our index expression. Instead, we call our `rkIndex` function.

Summary

This article has shown you a way to quickly move your DOS Xbase applications into the Windows environment using Microsoft’s Visual Basic and SuccessWare’s ROCK-E-T. Coverage here has been admittdly biased. There *can* be advantages to using the VB syntax. For one thing, all tables, regardless of source, can be accessed in the same manner. If you are familiar with SQL, VB’s implementation may also be more comfortable to you.

As for ROCK-E-T, we’ve really just touched on one of its features: ease of use. ROCK-E-T comes with a rich set of functions from sx_Append to sx_Zap, and includes data-aware VBX controls, built-in query optimization, and is much faster at accessing Xbase data than is native Visual Basic,

About the authors: Cecilia Smith is the Technical Support Manager for SuccessWare 90, Inc and Loren Scott is SuccessWare’s Product Manager.

References: Microsoft Knowledge base article: PSS ID Number: Q119116 Article last modified on 08-12-1994