Thank you for your interest in StorageTools. This edition of StorageTools are fully functional 32 bit OLE controls. The only limitation is that they will not run inside of an executable. You may use this demo control only from within vb32.exe. You must first register the controls by selecting the VB "Tools - Custom Controls" menu command and adding the "Desaware Storage Demo". and "Desaware Registry Demo".

Please refer to the Stg_Demo and Reg_Demo folders for examples on how to use StorageTools. This is just a small example of the full functionality of the StorageTools controls. The remainder of this file describes the properties and methods of each control.

The Storage Control

Storage Control Methods

DWSTG - AllocateMemoryHandle ()
This creates a global sharable block of memory suitable for Structured Storages. The block of memory uses 0 bytes initially, but expands to accommodate any data written to the Structured Storage. You may have more than one memory handle open at a time.

DWSTG - CompressStorageFile (Filename as String)
After heavy usage, Compound Files become fragmented and can contain many gaps that take up disk space. This method unfragments and compresses the Compound File. This method is time intensive, and should not be used often (certainly not every time the file is modified - when used on significantly compressed files it might actually make the file larger).

DWSTG - CreateStorageFile (Filename as String, AccessMode as Long)
Creates a new blank Compound File on the disk. The Filename parameter should contain the name (with full path, as necessary) of the file to create. The file is opened in the style dictated by the flags in the AccessMode parameter. The possible flags are detailed in the Constants section of this manual. If CreateStorageFile is successful, a Storage Object is returned that represents the root directory of the Structured Storage. If not, an error is triggered (see the Possible Errors section for a description of errors).
This method can be used to open files that are not in the Structured Storage format. Include STG_CONVERT as one of the access mode flags. The file will appear to have a single stream named “Contents”, which will contain the entire file contents. Simply reading from the file will cause no changes, but modifying anything will change the file into Structured Storage format permanently.

DWSTG - CreateStorageMemory (Name as String, AccessMode as Long, hGlobal as Long)
Creates a new blank Structured Storage in memory at the location specified by the global memory handle hGlobal. (See AllocateMemoryHandle for information as to how to create one of these handles.) The storage is given the name found in the Name parameter, and is opened in the style dictated by the flags in the AccessMode parameter. The possible flags are detailed in the Constants section of this manual. In Windows 3.1, the hGlobal handle can be passed to other applications which can then use it to open the Structured Storage. This creates a powerful method for sharing large amounts of organized information with very little overhead. Memory storages are useful in all operating systems as a mode of organizing memory and allowing easy saving of data to disk. The Structured Storage initially utilizes no memory but expands to accommodate any data written to it.
If CreateStorageMemory is successful, a Storage Object is returned that represents the root directory of the Structured Storage. If not, an error is invoked. This storage can be copied to an existing storage on disk using the CopyTo or MoveElementTo methods.

DWSTG - DeallocateMemoryHandle (hGlobal as Long)
This deallocates the block of memory allocated previously. Use this method after the Structured Storage is cleared (after the object is set to Nothing).

DWSTG - IsStorageFile (Filename as String)
Returns True if the file specified by the Filename parameter is a Compound File, False if it is not. If something goes wrong, an error is generated. Remember that non-Compound Files can still be opened as Compound Files. (This is done by using CreateStorageFile, specifying the file to be converted, and setting the STG_CONVERT flag. See the STG_CONVERT flag in the Constants section of this manual for more information.)

DWSTG - IsStorageMemory (hGlobal as Long)
Returns True if the memory specified by the global memory handle hGlobal contains a Structured Storage, False if not.

DWSTG - OpenStorageFile (Filename as String, AccessMode as Long)
Opens a Compound File on disk. The Filename parameter contains the name (with full path, if needed) of the file to open. The file is opened in the style dictated by the flags in the AccessMode parameter. The possible flags are detailed in the Constants section of this manual. If OpenStorageFile is successful, a Storage Object is returned that represents the root directory of the Structured Storage. If not, an error is generated.

DWSTG - OpenStorageMemory (Name as String, AccessMode as Long, hGlobal as Long)
Opens an already existing Structured Storage at the location in memory specified by the global memory handle hGlobal. It is opened in the style dictated by the flags in the AccessMode parameter. The possible flags are detailed in the Constants section of this manual. The Structured Storage expands to accommodate any data written to it. If OpenStorageMemory is successful, a Storage Object is returned that represents the root directory of the Structured Storage. This storage can be copied to disk using the CopyTo or MoveElementTo methods.

Storage Object Methods

Storage - Commit (CommitFlags as Long)
If the storage is opened in Transacted mode (see Access Flags in the Constants section of this manual), this procedure makes permanent all changes made to the storage since it was opened or since the last Commit. It makes reverting any changes up to this point impossible. If the storage is opened in Direct mode, changes might still be buffered. Commit flushes the buffer (although this only speeds up what would have been done in a few seconds in either case). CommitFlags describes how the storage should be committed (see the Constants section of this manual).

Storage - CopyTo (DestinationStorage as Object)
Copies the contents of this storage to the already opened Storage Object passed to the DestinationStorage parameter.

Storage - CreateStorage (StorageName as String, AccessMode as Long)
Creates a sub-Storage in this Storage. The Filename parameter contains the name of the storage to be constructed. The file is opened in the style dictated by the flags in the AccessMode parameter. The possible flags are detailed in the Constants section of this manual. If CreateStorage is successful, a Storage Object is returned that represents the newly created storage. If not, an error is generated.

Storage - CreateStream (StreamName as String, AccessMode as Long)
Creates a sub-stream in this storage. The Filename parameter contains the name of the stream to be constructed. The file is opened in the style dictated by the flags in the AccessMode

parameter. The possible flags are detailed in the Constants section of this manual. If CreateStream is successful, a Stream Object is returned that represents the newly created stream. If not, an error is generated.

Storage - DestroyElement (Name as String)
Deletes the named storage or stream from this storage. If there is any problem, an error is sent.

Storage - Directory (Index as Long, Type as Integer)
This method can be used to obtain a catalogue of all storages and streams contained in this storage. Each call to Directory returns the name of the storage or stream specified by the Index. The Type integer is changed to reveal the type of the element: STG_TYPE_STREAM if the returned name represents a stream, STG_TYPE_STORAGE if the returned name represents a storage, and STG_TYPE_NONE if there are no more elements. This sample code details how to employ this method:

```
i = 0
Do
   ElementName = StorageObject.Directory(i, FileType)
   If FileType = STG_TYPE_NONE Then Exit Do
   If FileType = STG_TYPE_STORAGE Then
      ' ElementName is a storage
   ElseIf FileType = STG_TYPE_STREAM Then
      ' ElementName is a stream
   End If
   i = i + 1
Loop
```

Storage - GetCreationDate ()
Obtains the date this storage was created.

Storage - GetLastModifyDate ()
Obtains the date this storage was last changed in any way.

Storage - GetLastAccessDate ()
Obtains the date this storage was last read. Not all filesystems carry this information -- in that case GetLastAccessDate will return a zero as the last date of access.

Storage - MoveElementTo (ElementName as String, NewName as String, DestStorage as Object, MoveFlags as Long)
Copies or moves an element from within this storage to the already open storage object passed to the DestStorage parameter. The element can be renamed by putting a different name in NewName. If NewName is Null, then the old name is kept. If MoveFlags is STG_MOVEMOVE then the original is erased (a move); if MoveFlags is STG_MOVECOPY then the original is not erased (a copy).

Storage - OpenStorage (StorageName as String, AccessMode as Long)
Opens a storage that is in the same file as this storage. The Filename parameter contains the name (and path, as necessary) of the storage to open. Like changing directories in DOS, the path can be relative to the root (with a leading backslash ("\")) or relative to this storage (without the leading backslash). The storage is opened in the style dictated by the flags in the AccessMode parameter. The possible flags are detailed in the Constants section of this manual. If OpenStorage is successful, a Storage OLE Object is returned that represents the newly opened storage. If not, an error is generated.

Storage - OpenStream (StreamName as String, AccessMode as Long)
Opens a sub-stream of this storage. The Filename parameter contains the name of the stream to open. The stream is opened in the style dictated by the flags in the AccessMode parameter. The

possible flags are detailed in the Constants section of this manual. If OpenStream is successful, a Stream OLE Object is returned that represents the newly opened stream. If not, an error is generated.

Storage - RenameElement (OldName as String, NewName as String)
Renames one of the storages or streams in this storage. Compound File names are limited to 32 characters in length.

Storage - Revert ()
When this method is called, if the storage is opened in Transacted mode, then all changes made to the storage since it was created or since the last Commit are discarded. Revert has no effect in Direct mode.

Storage - SetElementTimes(Element as String, CreationDate as Date, LastModifyDate as Date, LastAccessDate as Date)
Sets any or all of the dates for a sub-storage of the storage named in Element (streams do not carry any date information in the current implementation of OLE2). Use Null for any date you do not want changed. LastAccessDate is not saved on FAT or NTFS filesystems, so changing it might not be useful.

Property Set Functions
The following functions deal with reading and writing a PropertyPropert Set. They should be used only on the root Storage of a Compound File.
Here is an example of how you would use these methods to edit a string in the Summary Information Property set:

```
‘ if the SummaryInformation stream exists, load its information
If (RootStorage.siOpenSummaryInfo = True) Then
  ‘ get the title
  titleString = RootStorage.siGetTitle()
  titleString = titleString & “ version 2”
  ‘ set the new title
  RootStorage.siSetTitle titleString
  ‘ save the changes to the SummaryInformation stream
  RootStorage.siSaveSummaryInfo
        ‘ just like any stream, you must commit the        root storage to
        ‘ actually write to disk
        RootStorage.Commit STG_DEFAULT
End If
```

Storage - siSetTitle (Title as String)
Sets the title of the document.
Storage - siGetTitle ()
Returns the title of the document.

Storage - siSetSubject (Subject as String)
Sets the subject of the document.

Storage - siGetSubject ()
Returns the subject of the document.

Storage - siSetAuthor (Author as String)
Sets the author of the document.

Storage - siGetAuthor ()
Returns the author of the document.

Storage - siSetKeywords (Keyword as String)
Sets key words related to the subject of the document. These keywords are typically used in search routines.

Storage - siGetKeywords ()
Returns keywords relating to the subject of the document.

Storage - siSetComments (Comment as String)
Sets the comment field. This area might be used for making notes to oneself or others.

Storage - siGetComments ()
Returns the comment field.

Storage - siSetLastAuthor (LastAuthor as String)
Sets the latest person to have modified the document.

Storage - siGetLastAuthor ()
Returns the last person who has modified the document.

Storage - siIncrementRevNum ()
Increments the number of times a document has been revised.

Storage - siGetRevNum ()
Returns the number of times the document has been revised.

Storage - siStartEditTimer ()
This starts an internal timer. It is useful for determining how long a document has been opened.

Storage - siAddEditTimerToTotal ()
Adds the amount of time since siStartEditTimer was called to the field recording the total amount of time the document has been opened.

Storage - siGetTotalEditTime ()
Returns the total amount of time a document has been open since it has been created, in minutes.

Storage - siRecordPrintDate ()
Sets the field containing the last time this document was printed to the current date and time.

Storage - siGetLastPrintDate ()
Returns the last time the document was printed.

Storage - siRecordCreateDate ()
Sets the field containing the time the document was created to the current date and time.

Storage - siGetCreateDate ()
Returns the time the document was created.

Storage - siRecordSaveDate ()
Sets the field containing the last time this document was saved to the current date and time.

Storage - siGetLastSaveDate ()
Returns the last time the document was saved.

Storage - siSetNumberOfPages (NumPages as Long)
Sets the number of pages.

Storage - siGetNumberOfPages ()
Returns the number of pages.

Storage - siSetNumberOfWords (NumWords as Long)
Sets the number of words.

Storage - siGetNumberOfWords ()
Returns the number of words.

Storage - siSetNumberOfCharacters (NumChars as Long)
Sets the number of characters.

Storage - siGetNumberOfCharacters ()
Returns the number of characters.

Storage - siSetApplication (AppName as String)
Sets the name of the application that created the document.

Storage - siGetApplication ()
Returns the title of the application that created the document.

Storage - siSetTemplate ()
Sets the filename of the template used in the document.

Storage - siGetTemplate ()
Returns the filename of the template used in the document.

Storage - siSetSecurity (SecurityLevel as Long)
Sets the security level of the document. The recommended values are as follows:
Value Security Procedure
0 None.
1 Password protected. Do not allow viewing or editing of this document. Other programs cannot view or edit this document.
2 Read-Only recommend. The user will be warned if there is any attempt to edit the document.
3 Read-Only enforced. Does not allow any changes to the document.
4 Locked for Annotations. Does not allow any changes to the document.

Storage - siGetSecurity ()
Returns the security level of the document. Your program should behave as follows:
Value Security Procedure
0 None.
1 Password protected. Do not allow viewing or editing of this document.
2 Read-Only recommend. Warn the user if there is any attempt to edit the document.
3 Read-Only enforced. Do not allow any changes to the document.
4 Locked for Annotations. Do not allow any changes to the document.

Storage - siOpenSummaryInfo ()
Retrieves information from the SummaryInformation Property Set stream in the root storage of this Compound File. If there is no SummaryInformation Property Sset, siOpenSummaryInfo() generates an error. Before the siOpenSummaryInfo() is called, using any function beginning with “Get“ will return the default value, which will be either zero or a blank string depending on the type.

Storage - siSaveSummaryInfo ()

Saves any changes to the SummaryInformation Property Set stream in the root storage. Just as with any stream, these changes are not actually written to disk until the root storage object's Commit method is called.

Stream OLE Object Methods

Stream - Flush (CommitFlags as Long)
In the current implementation of OLE2, streams can only be opened in Direct mode. Flush flushes any internal buffers (although this only speeds up what would have been done by a few seconds in any case). CommitFlags describes how the stream should be committed (see the Constants section of this manual).

Stream - Get (SeekPosition as Long, Buffer as Variant)
Reads information in Visual Basic Binary form from this stream. The SeekPosition parameter specifies the position in number of bytes from the start of the file, beginning with zero. Use the constant SEEK_DONTMOVE to read at the location specified by the last Seek or at where the last Get or Put concluded. The amount of data read depends upon the size and type of the variant Buffer. For example, if you want to read in a 10 character string, you might set up Buffer in this manner:
Dim Buffer as Variant
Buffer = String (10, 0)
Stream.Get (SeekPosition, Buffer)

Stream - GetRecord (RecordNumber as Long, RecordSize as Long, RecordAddress as Long)
(Currently available in 16 bit Windows only. See DWADDR32.dll for 32 bit Windows.)
Reads a block of information from the record RecordNumber of size RecordSize. The RecordNumber parameter specifies the number of records from the start of the file, beginning with zero. All records in the stream must be of the same size and must match the size of the record exactly in order to prevent corruption of data. Use the constant SEEK_DONTMOVE to read at the location specified by the last Seek or where the last GetRecord or PutRecord terminated. If you do use Seek, be sure to seek to an whole multiple of the size of the record. You can get the address of a record by using the GetAddressForRecord function from the DWADDR16.DLL.

Stream - GetSize ()
Returns the size, in bytes, of this stream.

Stream - GetString (StringData as String)
Obtains a string from the Visual Basic Sequential formatted stream. The string returned is changed to the correct size. Currently you may have to convert the string to the correct type using Visual Basic conversion functions. The seek pointer should be pointing to the start of the item to retrieve. After GetString is done the seek pointer will point to the next item.
For example, if the next element is the integer "123", the statement
Stream.GetString (InputString)
will return the string "123", which you can convert to an integer with:
InputInteger = CInt(InputString)

Stream - Put (SeekPosition as Long, Buffer as Variant)
Writes information to this stream in the same way Visual Basic writes Binary data. The SeekPosition parameter specifies the position in number of bytes from the start of the file, beginning with zero. Use the constant SEEK_DONTMOVE to write at the location specified by the last Seek or at where the last Get or Put concluded. The amount of information written depends on the length of the data in the variant Buffer. If the data is written past the end of the stream, the stream automatically expands.

Stream - PutRecord (RecordNumber as Long, RecordSize as Long, RecordAddress as Long)
(Currently available in 16 bit Windows only.)
Writes a block of information to the record RecordNumber of size RecordSize. The RecordNumber parameter specifies the number of records from the start of the file, beginning with zero. All records in the stream must be the same size and must match the size of the record exactly in order to prevent the corruption of data. Use the constant SEEK_DONTMOVE to write at the location specified by the last Seek or at where the last GetRecord or PutRecord concluded. If you do use Seek, be sure to seek to an whole multiple of the size of the record. You can obtain the address of a record by using the GetAddressForRecord function from the DWADDR16.DLL.

Stream - PutString (StringData as String, VarType as Integer, Comma as Integer)
This writes a string to the stream in Visual Basic Sequential format. VarType should contain the type of the variable (obtained from Visual Basic with the VarType() function). If Comma is True, then commas are used to separate items; if not, then a Line Feed/Carriage Return is used. The stream automatically expands if the data is written past the end of the file.

Stream - Seek (Position as Long, SeekFlag as Long)
Sets the position at which the next Put or Get will be done. The Position parameter specifies the number of bytes from the location specified by the SeekFlag parameter. (See SeekFlag in the Constants section of this manual.)
Seek retruns a long integer which contains the current position of the seek pointer. You can locate the current position of the seek pointer with the line:
CurrentPos = Stream.Seek (0,STG_STREAM_SEEK_CUR)

Stream - SetSize (NewSize as Long)
This sets the probable maximum size of this storage. The Compound File will then allocate the appropriate amount of (usually contiguous) memory within the file. This improves the efficiency of the entire file without limiting the size of the stream.

Stream - VariantGet (SeekPosition as Long, Buffer as Variant)
Reads information in Visual Basic Variant Binary form from this stream (that is, the way Variants are read from Binary files). The SeekPosition parameter specifies the position in number of bytes from the start of the file, beginning with zero. Use the constant SEEK_DONTMOVE to read at the location specified by the last Seek or at where the last Variant Get or Variant Put concluded. The amount of data read depends upon the size and type of the data written before, because the type information is saved in the file with the data. The Variant Buffer is automatically converted to the correct type.

Stream - VariantPut (SeekPosition as Long, Buffer as Variant)
Writes information to this stream in the way Visual Basic writes Variants to Binary data. The SeekPosition parameter specifies the position in number of bytes from the start of the file, beginning with zero. Use the constant SEEK_DONTMOVE to write at the location specified by the last Seek or at where the last VariantGet or VariantPut concluded. The amount of information written depends upon the length of the data in the variant Buffer. If the data is written past the end of the stream, the stream automatically expands.

Stream OLE Object Properties
Storage - EOF
This serves the same purpose as the Visual Basic EOF function - that is, it returns True if the seek pointer is at the end of the stream. You can verify this to ensure that you do not read past the end of the stream (remember that writing past the end of the stream, even if you set a maximum size with SetSize, simply expands the stream).
Stream - Name
The name of the Object. Begins as the name of the stream itself. Use the RenameElement

method to rename the actual streams and storages.

Functions in the DWADDR16.DLL

(16 bit windows only)
Declare Function dwGetAddressForRecord& Lib "dwaddres.dll"
(Record as Any)
Returns the address of the specified record as a long integer. These addresses often change, so it should only be used right after this call, and not kept for any length of time. Use with GetRecord and PutRecord.
Functions in the DWADDR32.DLL
(32 bit windows only)
Declare Sub dwGetBytesFromRecord Lib "dwaddr32.dll"
(ByVal RecordSize As Integer, Record As Any, Bytes As Variant)
Copies the information in the user-defined type Record into the array of bytes held in the variant Bytes. Bytes need not be defined. RecordSize is the length, in bytes, of the entire user-defined type. You can then use Put or VariantPut to save the array to a stream.
Declare Sub dwGetRecordFromBytes Lib "dwaddr32.dll"
(ByVal RecordSize As Integer, Record As Any, Bytes As Variant)
Copies the information held in the array of bytes in the variant Bytes into the user defined type record. RecordSize is the length in bytes of the entire user defined type. Verify that the Bytes variant array contains as many bytes as are needed to fill the record structure. In order to use this function to read a record from a stream, first read the required number of bytes into an array of bytes using Get or VariantGet. Then you can use this function to populate the record properly.

Constants

Access Flags
Use these with CreateStorageFile, OpenStorageFile, CreateStorage-Memory, OpenStorageMemory, CreateStorage, OpenStorage, Create-Stream, or OpenStream.
any one of:
STG_READ = 0
Read from, but not write to the element. Put and other routines that write information will still work, but any attempt to Flush or Commit will result in an error. With STG_DIRECT set, this is faster and takes less memory and disk space than almost any other way of opening a file.
STG_WRITE = 1
Write to, but not read from the element.
STG_READWRITE = 2
Allows both reading and writing to the element.
any one of:
STG_SHARE_DENY_READ = &H30
Does not allow any other to read from this element.
STG_SHARE_DENY_WRITE = &H20
Does not allow any other to write to this element.
STG_SHARE_EXCLUSIVE = &H10
Does not allow any other to access this element in any way. All child streams and storages must have this flag set.
STG_SHARE_DENY_NONE = &H40
Allows any other to read from or write to this element as long as it is open.
either one of:
STG_DIRECT = &H0
All changes are made directly to the file. This is faster and requires less memory, but is less flexible. All streams must have this flag set.
STG_TRANSACTED = &H00010000
All changes are made to an internally kept copy of the storage. It is possible to undo any changes made since opening the storage or the last Commit() by using the Revert method. The parent of a storage that is transacted need not be transacted itself.

if creating a new storage or stream (including a new Compound File) either one of:
STG_CREATE = &H00001000
Create the new storage or stream, writing over any file that has the same name (if such a file exists).
STG_FAILIFTHERE = &H0
Create the new storage or stream unless one with the same name currently exists.
if creating a new Compound File, one can use:
STG_CONVERT = &H00020000
This will convert any normal file with the name and path specified to be converted into a Compound File with a single stream labeled “CONTENTS”. All of the data that the file held previously will be in the “CONTENTS” stream. No changes are made to the original file until the root storage is committed, so it is possible to read any file as if it was a Compound File without changing the original in any way.
STG_DELETEONRELEASE = &H04000000
Automatically deletes the file as soon as the object that represents it is released (that is, set equal to “Nothing”). Useful for temporary files.
if opening a Compound file, one can use:
STG_PRIORITY = &H00020000
Opens files with a minimum of overhead. Useful in cases where you are opening a storage just to copy it to another location. Must be used in conjunction with STG_READ and STG_DIRECT. Do not keep a STG_PRIORITY storage open any longer than necessary, as it prevents the Compound File from having any changes committed.
SeekPosition Flag
For use with the Stream object functions Get, Put, VariantGet, VariantPut, GetString, PutString, GetRecord, SetRecord.
STG_SEEK_DONTMOVE = -1
For reading or writing at the current location of the seek pointer.
Seek Flags
For use with Stream object procedure Seek.
STG_STREAM_SEEK_SET = 0
Position parameter is a count in bytes from the front of the stream.
STG_STREAM_SEEK_CUR = 1
Position parameter is a count in bytes forwards from the current seek position.
STG_STREAM_SEEK_END = 2
This flag means that the Position parameter is a count in bytes backwards from the end of the stream.
Move Flags
For use with Storage object procedure MoveElement
STG_MOVEMOVE = 0
MoveElement will erase the original element.
STG_MOVECOPY = 1
MoveElement will not erase the original.
Commit Flags
For use with Commit and Flush. You can specify STG_DEFAULT or any combination of the others.
STG_DEFAULT = 0
Writes to those sections of this storage that have changed, but does not touch those that have not. This flag should be generally employed, as it faster than using STG_OVERWRITE.
STG_OVERWRITE = 1
Writes all information held in this element, whether or not it has been modified. This will occasionally result in smaller files than using STG_DEFAULT. See CompressStorageFile.
STG_ONLYIFCURRENT = 2
Prevents multiple users of a storage from overwriting the other’s changes.
STG_ONLYIFCURRENT commits changes only if no one else has made changes since the last time this user opened or committed this storage. If other changes have been made, the error
STG_E_NOTCURRENT is sent. You can overwrite the changes by attempting another

Commit with STG_DEFAULT set.
STG_DANGEROUSLYCOMMITMERELYTODISKCACHE = 4
Only commits to the disk cache.
Directory File Type
For use with the Storage object method Directory. These flags indicate the type of object returned.
STG_TYPE_STREAM = 2
The name belongs to a stream.
STG_TYPE_STORAGE = 1
The name belongs to a storage.
STG_TYPE_NONE = 0
The end of the directory has been reached. There are no more names.

The Registry Control

Properties

Registry - CurrentKey
This property contains the current key upon which other properties and the majority of the methods are based. It starts at the root of the CurrentRoot. To change this to a different key, set CurrentKey with a full path (list the parent keys all the way to the root as necessary). If you want to change the key relative to the current key, use the ChangeKey method. If CurrentKey is set to a non-existent or non-accessible key, it is reset to the root of the CurrentRoot and an error is generated. Err.Description is “Current Key Error” and Err.Number depends upon which error caused the problem.

Registry - CurrentRoot
This specifies in which root of the Registry the CurrentKey currently resides. The possible values depend upon which operating system is being used. Windows 3.1 only has HKey_Classes_Root. Windows NT has three more. Windows 95 has the same ones as Windows NT in addition to two more. However, the controls will display all root keys contained in Windows 95. If you select a key that does not exist in the operating system that you are using, an error will result. If the root is changed, CurrentKey is changed to become the root of the new root. All operations take place in the current root only. Any problem will generate a “Invalid Root Value” error.

Registry - DefaultValue
This holds the “default” value, the value directly associated with the CurrentKey. It is the only type of value available in Windows 3.1. This is the value known as “<no name>“ in Windows NT and “(default)” in Windows 95 registry editors. If the CurrentKey does not have a value, or if the default value is not a string, DefaultValue is empty. When DefaultValue is changed, the Registry is updated to reflect the change. If the default value exists, it is also represented as a value in the ValueNameArray (with an empty string as its name). DefaultValue is not very useful in roots other than Hkey_Classes_Root, as this property only holds Reg_SZ or Reg_Expand_SZ values.

Registry - FindResultKey
This property holds the latest key found with FindFirstKey, FindNextKey, FindFirstValue, FindNextValue, FindFirstValueName or FindNextValueName. This is a read-only property. It is empty by default.

Registry - FindResultValueName
This is a name of a value. It holds the latest result in a search for a certain value name, or it specifies the latest value found with FindFirstValue or FindNextValue. If a search was made by

FindFirstKey or FindNextKey, FindResultValueName will be blank. This is a read only property. To find out more about the value named here, use the various GetValue methods.

Registry - KeyLock
When KeyLock is set to True, the CurrentKey is kept open. This speeds up access to the Registry, but it is somewhat less safe. The increases in speed take effect during those functions that occur at the CurrentKey (such as CreateKey or DeleteKey) but not during those functions that do not occur at the CurrentKey (such as the Find functions). You should probably set KeyLock to True just before a series of calls to read or change the Registry, then set KeyLock back to False. Never leave this True longer than needed. KeyLock is False by default.

Registry - NumOfSubkeys
This holds the number of subkeys currently in the CurrentKey. Use SubkeyArray() to access them.

Registry - NumOfValues
This holds the number of values currently in the CurrentKey. Use the ValueNameArray() property to access the names of each of the values, then the various GetValue methods to obtain the actual data in the value. In Windows 3.1, this is always either 0 or 1.

Registry - SubkeyArray (Index as Integer)
An array of all subkeys of the CurrentKey is kept in the SubkeyArray(). The array starts at 0. The number of subkeys is in the NumOfSubkeys property. If the index is beyond either of these bounds, SubkeyArray() returns a null string and a standard Out Of Bounds error.

Registry - ValueNameArray(Index as Integer)
An array of the names of all the values held in the CurrentKey are kept in the ValueNameArray(). The array starts at 0. The number of values is in the NumOfValues property. If the index is beyond either of these bounds, ValueNameArray() returns a null and an Out Of Bounds error. This is not available on the Properties palette. A blank ValueNameArray element actually holds the “name” of the default value (blank because a default value does not have a name).

Methods

Registry - ChangeKey (ChangeStr as String)
ChangeKey allows you to change the CurrentKey using syntax similar to changing a file directory. If ChangeStr begins with a backslash (“\”), the change is relative to the root key -- otherwise, it is relative to the CurrentKey. A dot-dot (“..”) is used to move from a subkey to its parent. If ChangeKey cannot get to the specified key, it causes a “Trying to change to invalid key” error. You cannot go to the parent of the root key, even if such a key exists. (For example, you cannot use ChangeKey(“..”) to move from the root key of HKey_Current_User to HKey_Users even though HKey_Current_User is technically a subkey of HKey_Users.)

Registry - CreateKey (NewKey as String)
This function is used to make a new key. The NewKey string can specify an absolute path, or one relative to CurrentKey. Any subkeys of the new key that do not exist are also created. If the key specified already exists, nothing happens. If a new key cannot be made, CreateKey causes a “Failure trying to create a subkey” error.

Registry - DeleteKey (Key as String, EraseSubkeys as Boolean)
This function is used to delete an existing key. The key may be specified relative to the CurrentKey (without a leading “\”) or as an absolute path (with leading “\”). If EraseSubkeys is True, any subkeys are also deleted, otherwise if the key has any subkeys it is left alone. If the key cannot be deleted for some reason, DeleteKey generates the “Failure trying to delete key” error.

Registry - DeleteValue (Name as String)
This function is used to delete an existing value. The value to be deleted must reside in the CurrentKey. If Name is blank, then the default value is erased. If the value cannot be deleted successfully, DeleteValue sends a “Failure trying to delete value” error.

Registry - FindFirstKey (Name as String, Case as Boolean, FullString as Boolean, StartAtRoot as Boolean)
Call this function first in order to search for any keys that have a name matching Name. If Case is True, than the comparison between Name and the keys are case sensitive. If FullString is True, then the two strings are compared over their whole length. If FullString is False, then Name can be a substring of a key. If StartAtRoot is True, then the search begins at the root key and progresses through all subkeys -- otherwise, the search starts at the CurrentKey and progresses through its subkeys. If at least one match was found, FindFirstKey will return True and FindResultKey will hold the first key found. If no matches were found, FindFirstKey will return False. This function might take several seconds to complete, depending upon the size of the root being searched.

Registry - FindFirstValue (Value as Variant, TypeOfValue as Long, SizeOfValue as Long, FullData as Boolean, StartAtRoot as Boolean)
Call this function first in order to search for any keys that have a value matching Value. The Registry type of the variant Value should be specified in TypeOfValue. The size of the Value (in bytes) should be in SizeOfValue.
If FullData is True, then the two values are compared over their whole length. If FullData is False, then Value can be located anywhere within the value being viewed. (FullData is ignored if TypeOfValue is Reg_Dword). If StartAtRoot is True, then the search begins at the root key and progresses through all subkeys -- otherwise, the search starts at the CurrentKey and progresses through its subkeys.
If Value is a string (REG_SZ, REG_EXPAND_SZ or REG_MULTI_SZ), any comparisons are made without regard to case.
If at least one match was found, FindFirstValue will return True and FindResultKey and FindResultValueName will hold the first key found. If no matches were found, FindFirstValue will return False. This function may take several seconds to complete, depending upon the size of the root being searched.

Registry - FindFirstValueName (Name as String, Case as Boolean, FullString as Boolean, StartAtRoot as Boolean)
Call this function first in order to search for any keys that have a value named Name.
If Case is True, than the comparisons between Name and the value name are case sensitive. If FullString is True, then the two strings are compared over their whole length. If FullString is False, then Name can be a substring of a value name. If StartAtRoot is True, then the search begins at the root key and progresses through all subkeys -- otherwise, the search starts at the CurrentKey and progresses through its subkeys.
If at least one match was found, FindFirstValueName will return True and FindResultKey and FindResultValueName will hold the first key and value name found. If no matches were found, FindFirstValueName will return False. This function does not exist in Windows 3.1. This function may take up to 10 seconds to finish, depending on the size of the root being searched.

Registry - FindNextKey ()
After the first key is found using FindFirstKey, use this function to obtain the remainder of the keys matching those search parameters. If there are more keys that match, each call to FindNextKey will put another key into the FindResultKey property. If there are no more matching keys, FindNextKey returns False and the FindResultKey property is cleared.

Registry - FindNextValue ()
After the first key is found using FindFirstValue, use this function to obtain the remainder of the

keys with values matching those search parameters. If there are more keys that match, each call to FindNextValue will put another key into the FindResultKey and FindResultValueName properties. If there are no more matching values, FindNextValue returns False and FindResult properties are cleared.

Registry - FindNextValueName ()
After the first key is found using FindFirstValueName, use this function to obtain the remainder of the keys with value names matching those search parameters. If there are more keys that match, each call to FindNextValue will put another key into the FindResultKey and FindResultValueName properties. If there are no more matching keys, FindNextValueName returns False and FindResult properties are cleared. This function does not exist in Windows 3.1.

Registry - FlushRegistry ()
In 32 bit versions of Windows, much of the Registry is kept in memory to improve performance. Changes to the Registry are made to the copy in memory. The actual Registry on disk is usually updated a few seconds later. Call this function when you want to be 110% sure the Registry is completely saved to disk. This call is time intensive, so use it sparingly, if at all. KeyLock should be False before calling this function. FlushRegistry has no effect in Windows 3.1, because its Registry is always immediately updated to disk. If the Registry cannot be flushed, FlushRegistry returns False.

Registry - GetValueData (Name as String, AsString as Boolean)
This function returns the data in the value named Name residing in CurrentKey. Use the GetValueSize and GetValueType functions to correctly set aside the right amount of memory for the variant. See the Data Type section of this manual for more information on how the data is actually stored. If the AsString parameter is set to True, then any binary data will be returned in the form of a string. If it is set to False, then any binary data will be put into an array of bytes. If the specified value does not exist, GetValueData returns a Null.

Registry - GetValueSize (Name as String)
This function returns the size, in bytes, of the data in the value named Name residing in CurrentKey. If the value specified does not exist, GetValueSize returns 0.

Registry - GetValueType (Name as String)
This function returns a number representing the type of the data held in the value named Name in CurrentKey. If the value specified does not exist, GetValueType returns 0.

Registry - SetValue (Name as String, Data as Variant, TypeOfValue as Long, SizeOfValue as Long)
This function is used to modify the contents of a value in the CurrentKey. If the value named Name does not exist, it is created. If an error occurred while attempting to set the value, SetValue generates the error code “Cannot Create\Change Value”.

Conversion of Value Data Types for Visual Basic
Values of type Reg_SZ, Reg_Expand_SZ, or Reg_Multi_SZ are treated as Visual Basic strings. Values of type Reg_Dword or Reg_Dword_Big_Endian are treated as Visual Basic long integers. Other values are treated as binary information. These can be held in either Visual Basic strings or byte arrays.
Reg_Resource_List, Reg_Full_Resource_Descriptor, and Reg_Resource_Requirements_List are utilized by device drivers and other very low level Windows elements. Do not modify these unless you really know what you are doing.

WARNING!
Remember that the Registry holds a great deal of information that both Windows itself and many Windows applications depend upon. Be careful when making changes to any part of the Registry

that you did not create. A corrupted Registry might render a program or Windows itself unusable.