# CometLib for Internet Basic Users

This document presumes that the reader is already familiar with Comet Internet Basic (IB) and the Comet file system. If you are not familiar with either of these concepts, please refer to the document "CometLib for Non-Internet Basic Users"

For those of you who are already proficient at programming the Comet file system, the use of CometLib will require that you also acquire and learn how to use Microsoft Visual Basic (VB). Once you gain an understanding of how to use VB you should have little difficulty understanding how to use the features of CometLib.

This document will point out changes you must adapt to in order to access Comet files using CometLib.

## Numerics

Internet Basic supports fixed-point numeric values with a maximum length of 16 and a maximum precision of 15. VB Is limited to at most 15 digits of significant and precision digits. Place close attention to the "Numeric Data Limits" section of the CometLib Library Reference.

## Formatting (or lack thereof)

Internet Basic provides the ability to layout data records using the FORMAT statement. This is NOT the case with CometLib. Instead, you place data into a record at a specified position on a field-by-field basis. Field data is retrieved from a record in a similar manner. Methods have been provided to help ensure data is properly fielded.

## Exception Handling

Comet file system functions use an exception (EXCP=) mechanism to specify the label of an error handling routine. Not all COM compatible languages have such a mechanism so CometLib handles errors in a more generalized manner.

## True/False Values

Internet Basic pre-defines the constants TRUE (1) and FALSE (0) for use by a IB program. In VB the constant True is actually equal to –1 and False is equal to 0. It is very important that you understand that any reference to a Boolean value of True or False must not be mistaken for 1 or 0. To avoid possible programming errors it is recommended that you always use the appropriate symbolic constant in your code.

**File System Functions**

For the most part, the Comet file system functions you already use on a day-to-day basis are provided by CometLib. In some cases minor name changes have been made to account for the varying nature of certain file system functions.

For Example:

> In IB when you want to create a file you simply specify a K, S, or T to indicate the file type. In CometLib, you must use the method that corresponds to the file type you want to create.
>
> Internet Basic
> 
>     CREATE <FileName>, <RecSize>, K, <KeySize>, Dir=<Directory>
>
> CometLib
>
>     CreateKeyedFile(<FileName>, <Directory>, <RecSize>, <KeySize>)

The following is a list of Comet functions that will be substantially different as CometLib methods.

CREATE
- CreateKeyedFile
- CreateSequentialFile
- CreateTextFile

READ/WRITE/EXTRACT/INQUIRE – for moving sequentially through a file
- ReadNext
- WriteNext
- ExtractNext
- InquireNext

KEY
- NextKey

LOCK/UNLOCK
- LockFile
- UnLockFile

GETHANDLES
- GetDosHandles

FSTAT
- FileStatus

DSTAT
- DabStat

FILE – FORWARD/REVERSE/EOF/BOF

- Forward
- Reverse
- EndFile
- BeginFile

Please refer to the “CometLib Library Reference” for additional information on CometLib and Comet file system features.

# CometLib for Non-Internet Basic Users

The Comet file system consists of a collection of distributed folders that each contains varying combinations of Comet file types. A Comet folder can be potentially located on any computer connected to a Local Area Network that is serviced by the Comet network file server, CFileServe. Client access to CFileServe is facilitated through the use of CometLib, a COM server DLL that is installed on the client computer. Also, in order for CometLib to be accessible to a client application, Comet must first be installed and running on the same computer. Once Comet is running, CometLib can be used by an application to gain access to any of the configured Comet folders/files.

## Comet Folders

As was earlier stated, Comet files are stored in folders located on various file servers. Any folder that will contain a Comet file must be "registered" in advance with Comet. This is accomplished through the use of a configuration .INI file that must be processed by the Comet utility, SysGen. The person who performs the Comet installation is responsible for ensuring that any needed folders are configured for access by Comet. When a folder is configured for use by Comet it is assigned a short (up to 3 characters) name that will be used by CometLib to gain access to any files contained in that folder. To simplify this document, we will refer to the short folder name simply as a "Directory."

For Example:

```
Const Lun = 1
Const FileName = "MyFile"
Const Directory = "Dat"
DB.Open (Lun, FileName, Directory)
```

## Comet Files

Comet filenames for non-keyed files follow the 8.3 DOS file format of <filename>.<3 byte extension>. For keyed files, the filename is limited to 8 characters with no file extension.

CometLib provides several input/output methods designed for working with Comet files. To identify a file that has been opened, CometLib uses the concept of a logical-unit-number (LUN). Unlike a file "handle" which is assigned by a file system function, a LUN is assigned in advance by the user of the file. A LUN must have a value from 0 to 49, allowing a program to have up to 50 files opened simultaneously.

The Comet file system supports the following three file types:

- Text files
- Sequential files
- Keyed files

**Text files** are standard ASCII data files containing text records (up to 8,000 bytes per record). These records are stored in the order in which they are written to the file, and are accessible in the same order (i.e., sequential order). Text files are typically used to import and export standard ASCII data. They are also used to store IB source programs.

**Sequential files** are fixed record size files (up to 8,000 bytes) with fixed field sizes. Records are stored in the order they are written, and may be retrieved in sequential order (first record to last) as well as direct order (using a numeric index to represent the record number in the file). Hence, sequential files are sometimes referred to as “indexed sequential” files. Sequential files are commonly used as temporary transaction files.

**Keyed files** provide direct access to data records. Keyed files have fixed record sizes (up to 8,000 bytes) with fixed field sizes. The key for a keyed file is a string field up to 64 characters long. Records are stored in the order they are written, while the keys are stored in ASCII order. Each key includes a pointer to the associated data record. Records may be retrieved in random order (using the key field) or in key-sequential (ASCII) order.

## File/Record Position - Sequential Access

CFileServe keeps track of your program’s current position (record pointer) within a data file.  When a program opens a data file, the record pointer points to the beginning of the file. After the program reads the first record, the record pointer points to the second record. Subsequent reading by this program moves this pointer to subsequent records. If the program reaches the end of the file and tries to read another record an error will occur.  If the program closes the file and opens it again, the record pointer is positioned back to the first record in the file.

Here’s a code excerpt that demonstrates sequential reading from a file. This program opens the file “Contacts” from the “Bas” directory. It reads a record and prints the first 80 bytes to the main program form, and repeats this process until all of the records have been read/printed.

```
DB.Open (1, "Contacts", "Bas")
While (DB.ReadNext(Lun))
   RecData = DB.GetRecField(1, 0, 80)
   Print RecData
Wend
DB.Close(1)
```

## File/Record Position - Keyed Access (Keyed Files Only)

Comet keyed files can also be accessed using a record “key”.  A key is a string of characters (up to 64 bytes long) that uniquely identifies a record in a key file.  A record key can be used to randomly access a record without the need to traverse through the entire file.

Here’s a code excerpt that demonstrates reading a record from a file using keyed access. This program opens the file “Contacts” from the “Bas” directory. It reads a record and prints the first 80 bytes to the main program form.

```
DB.Open (Lun, "Contacts", "Bas")
DB.Read (Lun, “john”))
RecData = DB.GetRecField(Lun, 0, 80)
Print RecData
DB.Close(Lun)
```

## Compatibility with the Internet Basic Programming Language

Comet files may contain data of any kind. A CometLib record is represented as a string up to 8000 bytes long. Further limitations are placed on this data if it is meant to be shared with Internet Basic programs. The following paragraphs describe those limitations.

### Record Layout (Fields)

Internet Basic record data is organized in fixed-length fields. That is, every field must have a fixed position and length that is known to the application accessing a file. Field data is stored as ASCII strings with no added delimiters. String fields are limited to 254 characters each and numeric fields are limited to 18 (16 + 2 extra characters for a sign and decimal) characters each.

### Numeric Data

Internet Basic numeric data is expressed as an unpacked decimal fixed length field with fixed precision. It is up to the programmer to ensure that data placed into a field conforms exactly to the format required by a Comet program. Techniques will be described in the “CometLib Library Reference” document to aid with the formatting of numeric data.

Please refer to the “CometLib Library Reference” for additional information on CometLib and Comet file system features.

# CometLib Library Reference – Visual Basic

Before you begin using CometLib it is not imperative that you have a good understanding of Microsoft Visual Basic. CometLib is useable by any programming language that can link to a COM server object. All code examples associated with this document will be presented in Microsoft Visual Basic 6.0. This is for illustrative purposes only. If you have an understanding of object programming, by reviewing the VB examples, you should be able to extrapolate into the language of your choice.

## Getting Started

Before you can use CometLib in a VB program you must first add it to the VB project. This is accomplished by opening the “References” dialog located in the “Project” menu. Locate “CometLib” in the dialog and select/check it to make it available to your program. A reminder - this procedure applies only to VB 6.0. In VB.NET (and other versions of VB), these steps may be somewhat different.

You can familiarize yourself with the objects, methods, properties and Enums of CometLib by opening the “Object Browser” dialog located in the “View” menu. In the combo-box at the top of the dialog choose “CometLib”. You should now see three objects listed: CometErrors, CometFiles, and CometFileStatus. By selecting an object you will be presented a list of each method, property, or Enum for the object.

## Making a Connection to Comet

Before using any of the CometLib file-system functions you must first declare, construct and initialize a CometFiles object in your program.

```
‘ Variable to hold a CometFile object
Dim DB As CometLib.CometFiles

‘ Create a CometLib object and connect to Comet
Set DB = New CometFiles
DB.Initialize “ContactMgr”
```

Generally, a program would define (Dim) the CometFiles object in the global section of a program and perform the initialization in the program startup routine (Form Load).

## Ending a Connection to Comet

When you are done using the CometFiles object (or for that matter, any object) it is a good practice to explicitly end the connection to Comet and destroy the object.

```
‘ Terminate connection with Comet and delete object
DB.Terminate
```

```
      Set DB = Nothing
```

Generally, a program would place the termination code in the program shutdown routine (Form Unload).

## Error Handling

Error handling for methods involved in the creation/destruction of a CometFiles object is accomplished through the use of the standard VB error handling mechanism. This technique is also used for the FileStatus, DabStat, PutRecField, and GetRecField methods. All other file system methods can be tested for an error by examining the method return value. In these cases the return value of False for Boolean variables, -1 for Numeric variables, or a Null String (“”) for string variables would indicate an error.

```
Sub Main()

      ‘ Go here if an error occurs during startup
      On Error Goto StartupError

      ‘ Variable to hold a CometFiles object
      Dim DB As CometLib.CometFiles

      ‘ Create a CometLib object and connect to Comet
      Set DB = New CometFiles
      DB.Initialize “ContactMgr”

      ‘ Open returns True if successful else False if error
      If (DB.Open(1, “Contacts”, “Bas”)) Then
            .
            . Do some file work
      .
            DB.Close (1)
      Else
            ‘ Report the error and exit program
      MsgBox “Comet Error #” & DB.LastCometError & _
             “: ” & DB.GetCometExcpString(DB.LastCometError)
      End If

      Goto FuncExit

StartupError:
      ‘ Report the error and abort program
      MsgBox Err.Description, , "Startup Error"
      End

FuncExit:
      ‘ Terminate connection with Comet and delete object
      DB.Terminate
      Set DB = Nothing
End Sub
```

## Numeric Data Limits

VB supports two type of floating point variables, single precision and double precision. **For the purposes of storing decimal numeric data, the maximum number of digits retained for single precision variables will be 6 digits and for double precision variables it will be 15 digits**. These limits are specified such that a floating-point number with the maximum specified number of decimal digits can be rounded into a floating-point representation and back without loss of precision. That means you will be able to represent your decimal values in the range –999999 to 999999 for single precision and –999999999999999 to 999999999999999 for double precision without losing decimal precision. The smallest decimal that values you can represent in VB are affected in a similar manner. For single precision variables the smallest value would be 0.000001 for double precision it would be 0.000000000000001.

## Formatting Numeric Data

Internet Basic numeric data is expressed as fixed length fields with fixed precision. The data is stored in a right-justified manner with the sign (' ' or '-') placed to the right of the field. It is up to the programmer to ensure that data placed into a field conforms exactly to the format required by an Internet Basic program. To facilitate correct numeric formatting, the method StrNumToIB can be used to format the field based upon a supplied length and precision.

# CometLib - Quick Reference

## CometFiles Object - Methods

## Initialization/Termination

```
Sub Initialize(ClientID As String, [TaskID As Long = -1], [DirCount As Long = 50],
               [FileCount As Long = 51])
Sub Terminate()
```

## File Create/Erase/Rename/Status

```
Function CreateKeyedFile(FileName As String, Directory As String,
         RecordSize As Long, KeySize As Long) As Boolean
Function CreateSequentialFile(FileName As String, Directory As String,
         RecordSize As Long) As Boolean
Function CreateTextFile(FileName As String, Directory As String,
         RecordSize As Long) As Boolean
Function Erase(FileName As String, Directory As String) As Boolean
Function Rename(FromFileName As String, FromDirectory As String,
         ToFileName As String, ToDirectory As String) As Boolean
Function FileStatus(FileName As String, Directory As String)
         As CometFileStat
```

## File Open/Close

```
Function Open(Lun As Long, FileName As String, Directory As String) As Boolean
Function Close(Lun As Long) As Boolean
```

## Record Read/Write/Delete

```
Function Delete(Lun As Long, Key As String) As Boolean
Function Extract(Lun As Long, RecordKey) As Boolean
Function ExtractNext(Lun As Long) As Boolean
Function Inquire(Lun As Long, RecordKey) As Boolean
Function InquireNext(Lun As Long) As Boolean
Function Read(Lun As Long, RecordKey) As Boolean
Function ReadNext(Lun As Long) As Boolean
Function Insert(Lun As Long, RecordKey As String) As Boolean
Function ReWrite(Lun As Long, RecordKey As String) As Boolean
Function Write(Lun As Long, RecordKey) As Boolean
Function WriteNext(Lun As Long) As Boolean
```

## File Get/Set Position

```
Function FirstKey(Lun As Long) As String
Function LastKey(Lun As Long) As String
Function NextKey(Lun As Long) As String
Function PrevKey(Lun As Long) As String
Function BeginFile(Lun As Long) As Boolean
Function EndFile(Lun As Long) As Boolean
Function Position(Lun As Long, RecordKey) As Boolean
```

## File Control

```
Function LockFile(Lun As Long) As Boolean
Function UnlockFile(Lun As Long) As Boolean
Function Forward(Lun As Long) As Boolean
Function Reverse(Lun As Long) As Boolean
```

## File Information

```
Function GetDosHandles(Lun As Long, KeyHandle As Long) As Long
Function GetFileType(Lun As Long) As Long
Function GetKeySize(Lun As Long) As Long
Function GetRecordSize(Lun As Long) As Long
```

## Data Formatting

```
Function IBToStrNum(IBStrNum As String) As String)
Function StrNumToIB(StrNum As String, Length As Long, Precision As Long) As String
Function WriteInit(Lun As Long) As Boolean
Function GetRecField(Lun As Long, FieldPosition As Long, FieldLength As Long)
                     As String
Sub PutRecField(Lun As Long, FieldData As String, FieldPosition As Long,
                FieldLength As Long)
```

## Directory Status

```
Function DabStat(Dab As Long, Path As String) As String
```

## Error Handling

```
Function GetCometExcpString(Excp As Long) As String
Function GetLastErrorString() As String
```

## Properties

```
IsConnected As Boolean (read-only)
LastCometError As Long (read-only)
LastError As Long (read-only)
```

## CometFileStat Object

## Properties

```
AccessDate As String (read-only)
Attributes As Long (read-only)
CreateDate As String (read-only)
DataFileSize As Long (read-only)
Directory As String (read-only)
ExtractCount As Long (read-only)
FileType As Long (read-only)
KeyFileSize As Long (read-only)
KeySize As Long (read-only)
Locked As Boolean (read-only)
ModifyDate As String (read-only)
NextFile As String (read-only)
RecordSize As Long (read-only)
UserCount As Long (read-only)
```

## CometErrors Object

## Enums

```
excp02EndOfFile
excp03DiskFull
excp04KeyRequired
excp05DirRequired
excp06NoDiskSpace
excp07LockViolation
excp10FileNotAvailable
excp11FileNotFound
excp12FileExists
excp13DOSFileExists
excp24RenameSameDisk
excp26DOSEndOfFile
excp27DOSCallError
excp28WriteProtected
excp29DriverError
excp30DeviceInoperable
excp31DeviceUnavailable
excp32KeyNotFound
excp33Extracted
excp34LunInUse
excp35NoFileOpen
excp36FileIsOpen
excp37NoPermission
excp38InvalidFunction
excp43NoExtensionAllowed
excp44BufferOverflow
excp45InvalidParameter
excp46NonNumericData
excp47ParamTooLarge
excp48InvalidCreateParameter
excp49InvalidLun
excp53InvalidFileAccess
excp54InvalidFileFunction
excp56KeyFoundOnInsert
```

## CometFileTypes Object

### Enums

```
ftSequential
ftContiguous
ftKeyed
ftText
ftProgram
```

## CometLibErrors Object

### Enums

```
clibNoError = 0               ` No error
clibNoFileSystem = 1          ` CFAM not running
clibLoginRefused = 2          ` Login refused by CFAM
clibSharedMemory = 3          ` Couldn't create shared memory
clibControlWinError = 4       ` Error creating control window
clibBufferBusy = 5            ` Buffer was busy during send
clibLinkTimeout = 6           ` CFAM Link timeout error - gone?
clibNoCOnnection = 7          ` Connection to CFAM not established
clibShortBuffer = 8           ` Insufficient bytes returned from CFAM
clibCometError = 9            ` A Comet FS error occurred
clibDOSError = 10             ` Error in a DOS function
clibUserAborted = 11          ` User (app) sent an abort
clibLoginFailed = 12          ` Login refused because Comet is not connected
clibNoLicense = 13            ` CometLib server is not licensed
```

# CometLib - Detailed Reference

## CometFiles Object

### Methods

### Initialization/Termination

```
Sub Initialize(ClientID As String, [TaskID As Long = -1],
    [DirCount As Long = 50], [FileCount As Long = 51])
```

This method established a connection to the Comet file system and establishes the initial operating parameters required by the application.

Where:

> **ClientID** is a name that will be used to identify this application instance. This identifier will be visible to other administrative-type applications.
>
> **TaskID** is the number of the Comet task whose Directories you wish to inherit. If this value is -1 (default) the you will have access up to the first 50 configured Directories. Unlike a Comet partition #, the TaskID is always 0-based. To determine the TaskID for a Comet partition you subtract the first Comet partition # from the desired partition #. For instance, if your first partition is P05 and you want the Directories for P10, you would subtract the first partition # (5) from the target partition # (10) to reach a target TaskID of 5.
>
> **DirCount** specifies the maximum # (default is 50) of Directories your program can support. THIS FEATURE IS FOR FUTURE USE.
>
> **FileCount** specifies the maximum # (default 51) of files (LUN's) that your program will have open at the same time. THIS FEATURE IS FOR FUTURE USE.

```
Sub Terminate()
```

This method breaks a connection to the Comet file system causing any open files to be closed and any other resources to be returned to the operating system.

## File Create/Erase/Rename/Status

### Create

```
Function CreateKeyedFile(FileName As String, Directory As String,
         RecordSize As Long, KeySize As Long) As Boolean
Function CreateSequentialFile(FileName As String, Directory As String,
         RecordSize As Long) As Boolean
Function CreateTextFile(FileName As String, Directory As String,
         RecordSize As Long) As Boolean
```

Each of these functions creates a file of the specified type.

Where:

**FileName** for sequential and text files use the DOS 8.3 format while keyed files are restricted to 8 characters only (the extension is reserved for internal use).

**Directory** specifies the directory in which the file is to be created.

**RecordSize** is the number of characters to be stored in each record. For all file types the maximum record size is 8000. The minimum record size for sequential and text files is 1 and for keyed files 0 (key-only files).

**KeySize** (key-files only) specifies the number of characters that will be used to represent each record. The minimum key size is 1 and the maximum is 64.

### Erase

```
Function Erase(FileName As String, Directory As String) As Boolean
```

This function erases the specified file from the specified directory. If no Directory is specified, CometLib will search through all configured directories until it finds a file with the specified name. Erased file are NOT put in to the system trash-can. Once a file has been erased it cannot be recovered.

Where:

**FileName** is the name of the file to be erased.

**Directory** specifies the directory in which the file resides.

### Rename

```
Function Rename(FromFileName As String, FromDirectory As String,
         ToFileName As String, ToDirectory As String) As Boolean
```

This function renames the specified file to a new filename and directory. If no FromDirectory is specified, CometLib will search through all configured directories until it finds a file with the specified name. If no ToDirectory is specified CometLib will use the FromDirectory. If ToDirectory is different than FromDirectory the respective drive letters must be the same.

Where:

**FromFileName** is the name of the file to be renamed.

**FromDirectory** specifies the directory in which the FromFileName resides.

**ToFileName** is the new name of the file being renamed.

**ToDirectory** specifies the directory in which ToFileName resides.

## Status

```
Function FileStatus(FileName As String, Directory As String)
         As CometFileStat
```

This function returns a CometFileStat object which contains information about the specified file. If no FromDirectory is specified, CometLib will search through all configured directories until it finds a file with the specified name.  See the section on the CometFileStat object for more information regarding file status.

Where:

**FileName** is the name of the file whose information is desired.

**Directory** specifies the directory in which the file resides.

## File Open/Close

### Open

```
Function Open(Lun As Long, FileName As String, Directory As String)
         As Boolean
```

This function opens the specified file in the specified directory. If no Directory is specified, CometLib will search through all configured directories until it finds a file with the specified name.

Where:

> **Lun** is the LUN under which the file is to be opened.
>
> **FileName** is the name of the file to be opened.
>
> **Directory** specifies the directory in which the file resides.

### Close

```
Function Close(Lun As Long) As Boolean
```

This function closes a previously opened file.

Where:

> **Lun** is the LUN under which the file was opened.

## Record Reading/Writing/Deleting

Before continuing on with this section a word about data formatting is in order. As was earlier stated, Comet files are designed to maintain data in records of fixed-length fields. That is, the program that reads and/or writes data to a Comet file must be aware of the position and length of each and every field that it will read or modify. The Comet runtime has features built into it that allow record fields to be laid out in a ordered list (Format statement) that determines the field positions by the order and length of each variable contained in the list. In addition, the Comet runtime Format statement automatically handles the formatting of fields that contain numeric data represented in a program as numeric variables. CometLib does not have such a mechanism so it will remain up to the programmer to ensure that fields are laid out in a manner that will be compatible with the Comet runtime. To this end CometLib provides a handful of methods that can be used to aid in the layout and formatting of field data so that it will remain compatible with a Internet Basic program. If you are going to write a program that does not need to maintain compatibility with the Comet runtime you can format your data in whatever manner you choose. The following is a list of the methods that you can use to exchange record field data.

**IBToStrNum** converts a Internet Basic numeric string into a VB numeric string.
**StrNumToIB** converts a VB numeric string into a Internet Basic numeric string.
**WriteInit** clears the record buffer prior to storing write data in the buffer.
**PutRecField** places a VB string into the record buffer.
**GetRecField** retrieves a string from the record buffer.

### Read/ReadNext

```
Function Read(Lun As Long, RecordKey) As Boolean
Function ReadNext(Lun As Long) As Boolean
```

This method reads the record identified by the specified RecordKey (Read) or the next record in the file (ReadNext). Use the method GetRecField to retrieve data from the record buffer.

Where:

**Lun** is the LUN under which the file was opened.

**RecordKey** is the String or Numeric index that identifies the record.

For Example:

```
‘ If read is successful - get first 3 bytes from record
If (Read(Lun, “T00”))
   FieldData = GetFieldRec(Lun, 0, 3)
End If
```

### Inquire/InquireNext

```
Function Inquire(Lun As Long, RecordKey) As Boolean
```

Function **InquireNext**(Lun As Long) As Boolean

An Inquire is essentially the same as a read with the following exception. Unlike a read which will get an error (excp33Extracted) if the specified record is extracted (locked) by another program, the Inquire method will complete successfully even if the record is extracted.

## Extract/ExtractNext

Function **Extract**(Lun As Long, RecordKey) As Boolean
Function **ExtractNext**(Lun As Long) As Boolean

A Extract is essentially the same as a read with the following exception. When a Extract is performed on a record the successfully read record will remain locked to other programs until the extract is released by another file operation.

## Write/WriteNext

Function Write(Lun As Long, RecordKey) As Boolean
Function WriteNext(Lun As Long) As Boolean

The Write method writes the contents of the record buffer using the specified RecordKey (Write) or the next record position (WriteNext). Prior to performing a Write, the data must be placed into the record buffer using the PutRecField method.

Where:

**Lun** is the LUN under which the file was opened.

**RecordKey** is the String or Numeric (write only) index that identifies the record.

For Example:

```
      WriteInit(Lun)                ' Clear record buffer
      PutRecField(Lun, "T00", 0, 3) ' Put copy of Key in record
      PutRecField(Lun, "LP1", 3, 3) ' This terminal uses LP1
      Write(Lun, "T00")
```

## ReWrite

Function **ReWrite**(Lun As Long, RecordKey As String) As Boolean

The ReWrite method is essentially the same as a write except that if the record does not already exist an error will occur.

## Insert

Function **Insert**(Lun As Long, RecordKey As String) As Boolean

The Insert method is essentially the same as a write except that if the record already exist an error will occur.

## Delete

Function **Delete**(Lun As Long, Key As String) As Boolean

The Delete method removes the record specified by RecordKey from a file.

Where:

> **Lun** is the LUN under which the file was opened.
>
> **RecordKey** is the String that identifies the record.

## FirstKey/LastKey/NextKey/PrevKey

Function **FirstKey**(Lun As Long) As String
Function **LastKey**(Lun As Long) As String
Function **NextKey**(Lun As Long) As String
Function **PrevKey**(Lun As Long) As String

These methods each return their respective RecordKey for the specified file.

Where:

**Lun** is the LUN under which the file was opened.

For Example:

```
‘ If next record is a “Terminal” record then read it
While (Left(NextKey(Lun), 1) = “T”)
   ReadNext(Lun)
   .
   . Process record data
   .
Wend
```

## BeginFile/EndFile

Function **BeginFile**(Lun As Long) As Boolean
Function **EndFile**(Lun As Long) As Boolean

These methods each positions the file record pointer to their respective positions.

Where:

**Lun** is the LUN under which the file was opened.

For Example:

```
‘ If End Of File then start second pass
If (ReadNext(Lun))
   .
   . ‘ Process record
   .
Else  ‘ Exception occurred
   If (LastCometError = excp02EndOfFile)
      BeginFile(Lun)
      Pass = 2
   Else
      MsgBox GetLastErrorString
      End
   End If
End If
```

## Position

Function **Position**(Lun As Long, RecordKey) As Boolean

This method set the current record position to the specified RecordKey.  If the specified RecordKey does not exist (or you reach the end of file), no error will be reported and the record pointer will be set to the next higher key in the file.  You can use the NextKey method to determine your current position.

Where:

**Lun** is the LUN under which the file was opened.

**RecordKey** is the String or Numeric index that identifies the record.

## File Control

### LockFile/UnLockFile/Forward/Reverse

```
Function LockFile(Lun As Long) As Boolean
Function UnlockFile(Lun As Long) As Boolean
Function Forward(Lun As Long) As Boolean
Function Reverse(Lun As Long) As Boolean
```

These methods each perform their respective function for the specified file.

Where:

**Lun** is the LUN under which the file was opened.

**LockFile** places a system-wide file lock on the specified file making it inaccessible to other programs. If the file cannot be locked because it has been opened by another user an error excp07LockViolation will be generated.

**UnlockFile** removes a lock from a previously locked file.

**Reverse** causes the file record pointer to move in descending direction during read/write/position operations.

**Forward** causes the file record pointer to move in the default ascending direction during read/write/position operations.

## File Information

### GetFileType/GetKeySize/GetRecordSize

```
Function GetFileType(Lun As Long) As Long
Function GetKeySize(Lun As Long) As Long
Function GetRecordSize(Lun As Long) As Long
```

Each of these methods return their respective piece of information about the specified file.

Where:

> **Lun** is the LUN under which the file was opened.

For Example:

```
      If (GetFileType(Lun) = ftKeyed)
         MsgBox "This is a keyed file"
      End If
```

## Data Formatting

### WriteInit

```
Function WriteInit(Lun As Long) As Boolean
```

This method clears the file record buffer to all spaces. Call this method prior to placing data into the record buffer. In cases where you want to perform an update that involves changing only part of a record, you would **NOT** use this method. Instead, you would first read the record you want to update, change the field, and then write the record back to the file.

### PutRecField

```
Sub PutRecField(Lun As Long, FieldData As String, FieldPosition As Long,
                FieldLength As Long)
```

This method copies the specified string into the record buffer at the specified FieldPosition for the specified FieldLength. If the specified string is shorter than the indicated FieldLength, the field is padded with spaces.

For Example:

```
      WriteInit(Lun)
      PutRecField(Lun, "T00", 0, 3)       ' Put terminal# (key) into begin of record
      PutRecField(Lun, "LP1", 3, 3)       ' Printer name is next
      Write(Lun, "T00")                   ' Write the record using terminal as key
```

### GetRecField

```
Function GetRecField(Lun As Long, FieldPosition As Long, FieldLength As Long,
                     As String
```

This method retrieves a field from the record buffer at the specified FieldPosition for the specified FieldLength. When using this method keep in mind that any spaces added to pad the field will also be returned in the string.

For Example:

```
      If (Read(Lun, "T00"))
         TerminalNum = GetRecField(0, 3)  ' Get terminal# (same as key)
         PrinterName = GetRecField(3, 3)  ' Get printer name
      End If
```

### StrNumToIB

```
Function StrNumToIB(StrNum As String, Length As Long, Precision As Long) As String
```

This method converts a VB string numeric into a Internet Basic compatible string numeric. The conversion provides any necessary padding or precision and also moves the sign to the right. **This method should always be used when writing data to a file that will also be read from an Internet Basic program.** If an error occurs during conversion a NULL (“”) string will be returned and LastCometError will indicate the reason for the error.

For Example:

```
Dim MyNum As Double
Dim MyIBNumStr As String
Const IBNumLen = 7                  ‘ Length of numeric
Const IBNumPrec = 4                 ‘ Precision of numeric
Const IBNumFieldLen = IBNumLen+2    ‘ Allow room for decimal point And sign

MyNum = -123.45
MyIBNumStr = StrNumToIB(Str(MyNum), IBNumLen, IBNumPrec)
If (MyIBNumStr <> “”) Then
   ‘ Resulting string should be “123.4500-“
   PutRecField(Lun, MyIBNumStr, 0, IBNumFieldLen)
End If
```

## IBToStrNum

Function **IBToStrNum**(IBStrNum As String) As String

This method will convert a Internet Basic numeric string into a VB numeric string. The conversion is very simple in that it merely moves the sign to the left and strips blank spaces. **This method should always be used when reading data from a file that was written by an Internet Basic program.** If an error occurs during conversion a NULL (“”) string will be returned and LastCometError will indicate the reason for the error.

For Example:

```
Dim MyNum As Double
Dim MyIBNumStr As String
Dim MyNumStr As String
Const IBNumLen = 7                  ‘ Length of numeric
Const IBNumPrec = 4                 ‘ Precision of numeric
Const IBNumFieldLen = IBNumLen+2    ‘ Allow room for decimal point And sign

MyIBNumStr = GetRecField(Lun, 0, IBNumFieldLen)
MyNumStr = IBToStrNum(MyIBNumStr)
If (MyNumStr <> “”) Then
      MyNum = Val(MyNumStr)
End If
```

## Directory Status

```
Function DabStat(Dab As Long, Path As String) As String
```

Each Directory that a user has access to after connecting to CometLib is assigned a DAB (Directory Access Block) and is numbered sequentially from 0 to 49. This method returns the Directory Name and Path associated with the specified Dab. A NULL (“”) string may be returned for a DBA that currently has nothing assigned to it. This is not considered an error. The follow code demonstrates how to enumerate all of your DABs.

For Example:

```
    Dim Result As Long
    Dim DataBuff As String
    Dim Path As String
    Dim AllDabs As String

    On Error GoTo BadDab
    For Result = 0 To 49
        DataBuff = CometDB.DabStat(Result, Path)
        If (DataBuff <> "") Then _
            AllDabs = AllDabs + Chr(10) + "(" + _
            Right(Str(Result + 00), 2) + ")  " + DataBuff + ":" + Path
    Next Result

    MsgBox "Directories:" + Chr(10) + Chr(10) + AllDabs
    DabStatTest = True
    GoTo Done

BadDab:
    DabStatTest = False
    MsgBox "DabStat Error: " & CometDB.GetLastErrorString

Done:
```

## Error Handling

### GetCometExcpString

Function **GetCometExcpString**(Excp As Long) As String

This methods returns the ASCII string associated with the specified Comet exception.

For Example:

```
      On Error Goto ErrorFunc
      .
      .
      .
ErrorFunc:
      MsgBox "You got a Comet error: " & GetCometExcpString(LastCometError)
End Sub
```

### GetLastErrorString

Function **GetLastErrorString**() As String

This method returns a pre-formatted error string following a CometLib error. This string may be more useful under certain circumstances because it may contain additional information about an error.

For Example:

```
      On Error Goto ErrorFunc
      .
      .
      .
ErrorFunc:
      MsgBox "Error: " & GetLastErrorString()
End Sub
```

## Properties

### IsConnected

**IsConnected** As Boolean (read-only)

This property will have a value of True if CometLib is currently connected to Comet, False if not.

### LastCometError

**LastCometError** As Long (read-only)

This property contains the last Comet error number generated by CometLib.  A return value of (-1) would indicate that the last error generated was not a Comet (clibCometError or clibDOSError) error but was some other generic system error (see LastError).

### LastError

**LastError** As Long (read-only)

This property contains the last error of any kind that was generated by CometLib. Although most of the time errors are caused by a file system exception (LastError = clibCometError), occasionally an error will be generated for other reasons.

## CometFileStat Object

### Properties

```
AccessDate As String (read-only)
Attributes As Long (read-only)
CreateDate As String (read-only)
DataFileSize As Long (read-only)
Directory As String (read-only)
ExtractCount As Long (read-only)
FileType As Long (read-only)
KeyFileSize As Long (read-only)
KeySize As Long (read-only)
Locked As Boolean (read-only)
ModifyDate As String (read-only)
NextFile As String (read-only)
RecordSize As Long (read-only)
UserCount As Long (read-only)
```

## CometErrors Object

### Enums

```
excp02EndOfFile
excp03DiskFull
excp04KeyRequired
excp05DirRequired
excp06NoDiskSpace
excp07LockViolation
excp10FileNotAvailable
excp11FileNotFound
excp12FileExists
excp13DOSFileExists
excp24RenameSameDisk
excp26DOSEndOfFile
excp27DOSCallError
excp28WriteProtected
excp29DriverError
excp30DeviceInoperable
excp31DeviceUnavailable
excp32KeyNotFound
excp33Extracted
excp34LunInUse
excp35NoFileOpen
excp36FileIsOpen
excp37NoPermission
excp38InvalidFunction
excp43NoExtensionAllowed
excp44BufferOverflow
excp45InvalidParameter
excp46NonNumericData
excp47ParamTooLarge
excp48InvalidCreateParameter
excp49InvalidLun
excp53InvalidFileAccess
excp54InvalidFileFunction
excp56KeyFoundOnInsert
```

## CometFileTypes Object

### Enums

```
ftSequential
ftContiguous
ftKeyed
ftText
ftProgram
```

## CometLibErrors Object

### Enums

```
clibNoError = 0               ` No error
clibNoFileSystem = 1          ` CFAM not running
clibLoginRefused = 2          ` Login refused by CFAM
clibSharedMemory = 3          ` Couldn't create shared memory
clibControlWinError = 4       ` Error creating control window
clibBufferBusy = 5            ` Buffer was busy during send
clibLinkTimeout = 6           ` CFAM Link timeout error - gone?
clibNoCOnnection = 7          ` Connection to CFAM not established
clibShortBuffer = 8           ` Insufficient bytes returned from CFAM
clibCometError = 9            ` A Comet FS error occurred
clibDOSError = 10             ` Error in a DOS function
clibUserAborted = 11          ` User (app) sent an abort
clibLoginFailed = 12          ` Login refused because Comet is not connected
clibNoLicense = 13            ` CometLib server is not licensed
```