WinMark v5 ActiveX Interface

The new WinMark ActiveX control, available with WinMark version 5 and up, introduces new methods, properties and events to support multiple marking head types, and contains additional options in support of our new Flyer marking head. The v5 ActiveX control is a completely different control from that of versions 1 - 4, so it is named "Synrad WinMark Pro Control Version 5.0" to differentiate it from it's predecessor.

The purpose of this document is to:

1. explain of the differences and similarities between our earlier control and the new control
2. describe the new methods and events grouped by function
3. provide a alphabetical listing of the new methods and events with detailed descriptions

What is different?

The v5 control is different from, and the same as, the previous control used in WinMark versions 1 through 4:

- The new control contains all of the methods and properties used in the earlier control. Updating your existing application to use the v5 control does not mean that all of your existing code must be re-written… you will need to recompile the code with the new control referenced, add code to determine which head is being used (even with just one head available), but the old methods and properties will work the same.
- The new control provides support for selection of one head out of many that may be available to the PC running your ActiveX application. The v5 control can select from, and connect to, one (and ONLY one) DH/FHIN/FHTR/Fenix marker connected through a Fiber Link Controller Card, as well as from multiple Flyer heads connected through USB and/or Ethernet ports. Only one head may be controlled at a time, but the control allows you to search for available heads and connect to any head that is not already communicating to another application.
- The new control provides support for operating the Flyer head in Stand Alone (SA) mode… you can transfer files to and from the head, initiate or abort SA marking, or check on the progress of a SA mark session.
- At the point of this writing, you cannot place multiple instances of the control into a single process. You may create a multi-threaded application to support multiple instances of the control, but each process thread must have just one instance of the control.

The V5 ActiveX control is a work in progress, so be sure to visit our website (www.winmark.com) to see if a new build of WinMark might include an improved ActiveX control that provides functions you need that are not described in this document. The website also has sample projects that illustrate the use of the new control features.

The WinMark ActiveX Help file, available from the WinMark Help menu, also provides the latest details on the control and its use.

Instantiating the Control

Short ConnectPreviewWnd(Long pParent, Long pPreview)

Assign the V5 ActiveX control handle to the handle of another object, such as a Picture Box.

One of the rules of the new ActiveX control is that only one instance of the control may exist in a single process. If you're in the habit of placing more than one WinMark ActiveX control in a process (some customers have done this, to our chagrin), you won't be able to do so anymore. In VB, this also means that if you place the control right onto your form, you'll need to close the form before running the application – VB instantiates the control when it is viewed on the form, then tries to instantiate it again at run time and VB then crashes.
There are two ways around this: close the form before running the app, or place a picture box on the form and link the ActiveX control to the picture box in the form load procedure.

Dealing with multiple heads

The most significant change in the v5 ActiveX control is the fact that multiple marking heads may be 'connected' to the application. Among these 'heads' could be

- one (and only one) Fiber Link Controller Card installed in the PC - even if there is not a DH/FHIN/FHTR/Fenix/FenixII marker connected and powered up, the FLCC is detected as a 'head'
- one or more Flyer/Fenix Flyer heads powered up and connected through the USB port(s) on the PC
- one or more Flyer/Fenix Flyer heads powered up and connected through the Ethernet port(s) on the PC – the heads will be detected only if they've been used before and their Ethernet addresses have been stored in the WinMark registry. Use the ConnectToEthernet method to register new heads through the Ethernet.
- a 'Simulation Head' – if the control doesn't detect any heads when the application starts up, it will default to simulation mode (enabling you to create mark files without a head connected) and report this head type as an unknown head (type = -1).

A very important distinction must be made between *connected* and *available*:

- *connected* means that a data connection has been established between the head and the application
- *available* means that the head is connected and not being controlled by another process

Each head may be controlled by one process at a time. We don't want a marking head running two jobs at once.

As an example, consider the system in Figure 1 that has:

- a Flyer head (Flyer1) connected to an Ethernet network, but not powered up
- a Flyer head (Flyer2) connected to the Ethernet network, powered up but not running a job
- a Flyer head (Flyer3) connected to the Ethernet network, powered up and running a job
- a PC (PC1) connected to the network, running WinMark v5, controlling Flyer3
- a PC (PC2) connected to the network, running a v5 ActiveX application but not controlling a head
- a PC (PC3) on the network running the ActiveX application, also connected to a Flyer head (Flyer4) through it's USB port, also with an FLCC connected to a Fenix marker (Fenix1)

An application running on PC1 would see:

- Flyer1 is not connected
- Flyer2 is connected and available
- Flyer3 is connected and available
- *This PC would not see Flyer4 or Fenix1, since they are not directly connected to the network*

An application running on PC2 would see:

- Flyer1 is not connected
- Flyer2 is connected and available
- Flyer3 is connected but not available
- *This PC would not see Flyer4 or Fenix1, since they are not directly connected to the network*

An application running on PC3 would see:

- Flyer1 is not connected
- Flyer2 is connected and available
- Flyer3 is connected but not available
- Flyer4 is connected and available
- Fenix1 is connected and available

To summarize all of this: a head that is being controlled by a process is available only to that process.

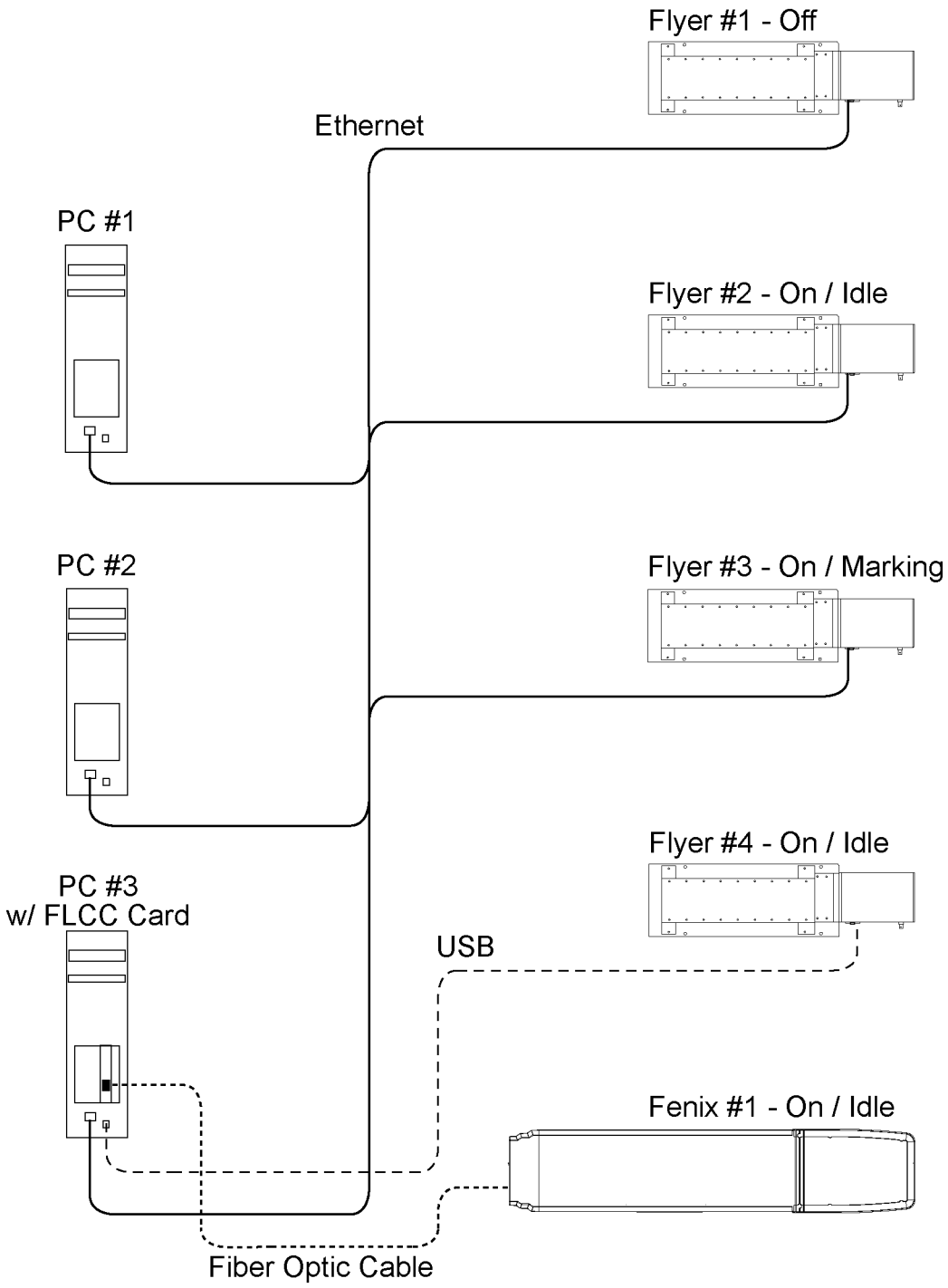

Figure 1: Multiple marking heads connected through a network

Head indexing and selection

This section will provide a brief overview of the marking head detection, identification and selection methods and events. For the complete details of the methods and events, refer to the full descriptions at the end of this document.

The new control assigns Index values to each of the various heads that are (or have been) connected and available to the application. The Index values are 0 based - the first head recognized is given Index 0 and they increment from there. This data is saved in the Windows registry - if a head was connected in the past but is not currently available, it is still treated as a potential head.

If you open your application without a head connected and running, and with no potential heads stored in the Windows registry, the ActiveX control will default to using a simulation head.

So, even if your application will be connected to just a single marking head, you can (and should) use these methods to verify that the head is connected and running.

Methods

Short GetMarkHeadCount()

Use this method to determine how many individual heads have been identified by the WinMark driver.

Short GetCurrentMarkingHead()

Use this method to retrieve the Index value of the currently selected head.

Short GetMarkHeadName(Short Index, BSTR pName)*

Use this method to retrieve the name of a head. For Flyer heads, the name is retrieved from the head. For other head types, the name is retrieved from the Windows registry.

Short GetMarkHeadStatus(Short Index)

Use this method to determine whether a head is available or in use.

Short GetMarkHeadType(Short Index)

Use this method to retrieve the head type.

Short SetCurrentMarkingHead(Short Index)

Use this method to select a head index as the current head.

Short ConnectToEthernetFlyer(BSTR strAddress)

Use this method to find and connect to a Flyer head using an Ethernet connection.

Short DeviceSimulation(Short mhType)

Use this method to create a simulated head in your application.

Events

FlyerConnectNotification(ByVal Index As Integer, ByVal bConnecting As Long)

Use this event to detect a change in the connection status of a Flyer using the USB port. This event will fire every time a head is connected or disconnected, providing the Index value of the head involved. For instance, if a new head is connected, the Index value will be incremented.

Reading from and writing to the discrete I/O

We've also added some new methods and events to provide more flexibility in dealing with the discrete I/O. In previous versions of the WinMark ActiveX control, you used GetDigitalBit and SetDigitalBit to handle I/O operations. To read all of the inputs from the Fiber Link Controller Card (FLCC) or from the marking head, you'd have to call GetDigitalBit once for each bit to be read. To monitor the status of an input bit, you would have called the GetDigitalBit method from a program loop or a timer control routine. The new control contains methods to read or write all of the inputs or outputs in a single method call. We've also added methods and events to command the control to report when a particular input state is present at the Flyer head, or when any of a group of inputs has changed state.

All of the I/O string values (pData, strBits, strMask) are formatted using '0', '1', and/or '-' characters: values of '0' mean that the I/O bit is inactive, '1' means that the bit is active, and '-' means don't care (used when defining required input state values). The strings are ordered so that bit 0 is on the left side of the string, bit 7 on the right.

Methods:

Short GetDigitalString(BSTR pData)*

Use this method to return the current state of all input bits on the Flyer head.

Short SetDigitalString(BSTR pData)*

Use this method to set the head's outputs to a desired state.

Short TrigWaitDigEvent(String strBits)

Use this method to command the Flyer head to fire the FlyerWaitDigital event when the input condition defined by strBits is true. For instance, a strBits value of '0001----' will cause the FlyerWaitDigital event to fire off only if IN0, IN1, and IN2 are inactive and IN3 is active. The '-' values mean that the states of IN4 – IN7 are ignored.

Short InputChangeDigital (String strMask)

Use this method to enable or disable the Flyer head's FlyerChangeDigital event. The strMask value uses '0' and '1' characters to indicate which input bits are of interest. For instance, using a strMask value of '11100000' means that the FlyerChangeDigital event fires off every time a change in state is detected on IN0, IN1 or IN2. The '0' values mean that changes to inputs IN3-7 are ignored. Use a strMask value of '00000000' to disable the FlyerChangeDigital event.

Events:

FlyerWaitDigital(ByVal strMessage As String)

The event fires off on the first detection of an input combination that satisfies the requirements of strBits passed through the TrigWaitDigEvent method. The strMessage value returns the state of all eight inputs at the time of the event.

FlyerChangeDigital(ByVal strMessage As String)

This event fires off on every change of state detected on the inputs of interest as defined by the strMask value passed through the InputChangeDigitial method. The strMessage value returns the state of all eight inputs at the time of the event.

Flyer 'System' properties
Many of the system properties that would normally reside in the WinMark registry are stored in FLASH memory in the Flyer head. For example, the head's IP address, the properties used for marking on the fly, and the head name are all system properties.

Use these methods to retrieve the property count, names and values, or to update the values of the system properties.

Short GetSystemPropertyCount()
Short GetSystemProp(BSTR strProp, BSTR pStrValue)*
Short GetSystemPropertyNameByIndex(Short PropIndex, BSTR pStrName)*
Short SetSystemProp(BSTR strProp, BSTR strValue)

You may also use these functions to read and write user-defined 'property' data in the Flyer head. If you use the SetSystemProp method to write data to a property name that is not recognized by the Flyer head, it will store the property name and the corresponding data to a separate data register in the Flyer's FLASH memory. You may then retrieve the value using the GetSystemProp method.

The use of these user-defined data registers should be limited to small amounts of data – less than 1024 strings of 255 characters or less per string. Since these properties are stored in FLASH memory, the data should be fairly static in nature… this is not a good place to store data that is frequently changing such as serial numbers or time codes that change on every mark.

Miscellaneous Methods

Short GetFlyerTemperature(Float pFront, Float* pBack, Boolean bOvertemp)*
The Flyer head contains two thermal sensors that allow you to monitor the head temperature. Use this method to retrieve the Flyer head's current temperature values and the status of the head's overtemp flag.

Short GetHeadUptime (none)
Use this method to retrieve the number of seconds that the Flyer head has been powered up. Since this value is returned as a Long Integer, the value rolls over at over 2.1 billion seconds.

Short GetNetworkMount (bNetworked as Long)
The Flyer head may be configured to read mark, font and data files from a network share. Use this method to query the head as to the status of the network share connection.

Short RebootHead (none)
Use this method to force a reboot of the Flyer head. The head takes about 30 seconds to reboot, during which time it will be unavailable. If you're using the USB port, the *FlyerConnectNotification* event will fire off after the head has finished the boot up process.

Stand-Alone marking functions (Flyer only)
The Flyer head supports Stand-Alone marking, meaning that WinMark mark files (*.MKH) may be downloaded to the FLASH memory in the head, loaded into RAM, then marked without the head being connected to a WinMark or ActiveX application. Even if a connection is established and maintained between an ActiveX application and the Flyer head, marking in SA mode may provide faster mark cycle times due to the reduction in data passing back and forth between the app and the head. Marking in SA mode also presents the possibility of managing multiple marking heads from a single application, with the app just telling each head which file to load and when to start or stop marking.

The Filestore
The Flyer head is equipped with 8MB (or more) of FLASH memory to store MKH files and font files (TTF or OTF types). We refer to this memory as the Filestore. The head is running a Linux operating system and uses the Journaling FLASH File System (JFFS2) to manage the Filestore. With the JFFS2, you can treat the Filestore as a normal hard drive, creating subdirectories, copying files to and from the Filestore and deleting files as needed.

Files in the Filestore root directory are preceded with a '/' character. For instance, the Canvas.mkh file is referenced as '/Canvas.mkh'.

Note that when downloading files to the Filestore, you are storing the file in FLASH memory… not loading it into the Flyer's RAM memory. If you load a file from the Filestore into RAM, the upload the file, modify it and save it back to the Filestore, the file in RAM is unchanged until you reload the file. This can be VERY confusing if you forget this basic nature of the Filestore.

Methods
Short GetFilestoreInfo (BufferSize as Long, Buffer as String)
Use this method to retrieve the names of the files currently residing in the Filestore.

Short GetFilestoreUsage (BytesUsed as Long, BytesAvailable as Long)
Use this method to retrieve the Filestore memory usage and capacity.

Short ReformatFileStore (none)
Use this method to reformat the Filestore. This will erase all MKH, DAT, TTF and MCF files within the Flyer FLASH memory.

Short CopyFile (strPathFrom as String, strPathTo as String)
Use this method to copy a file within the Filestore. If you copy the file to a filename that is already in use, the previous file will be overwritten.

Short DeleteFSObject (strPath as String)
Use this method to delete a file from the Filestore. Note that there is no 'wastebasket' function in the Filestore… once you delete a file, it is gone.

Short DownloadFile (strPathFrom as String)
Use this method to download a file to the Filestore. The file will be named 'Canvas.mkh' and saved to the Filestore's root directory.

Short MakeDir (strDirPath as String)
Use this method to create a subdirectory within the Filestore.

Short OpenFile (strPath as String)
Use this method to load a file from the Filestore into Flyer RAM to prepare for marking.

Short RenameFSObject (strOldPath as String, strNewPath as String)
Use this method to change the filename of one of the files in the Filestore.

Short SaveToFileStore (strPathFrom as String, strPathTo as String)

Use this method to download a file to the Flyer Filestore, specifying the name the file should have within the Filestore.

Short SetAsMarkOnStartup (strPath as String)
Use this method to configure the Flyer head to load and start marking a mark file from the Filestore when the Flyer head boots up.

Short SAGetCurrentFilePath (strFileName as String)
Use this method to read the filename and path of the mark file currently loaded into the Flyer RAM for marking.

Short UploadFile (strPathFrom as String, strPathTo as String)
Use this method to upload a file from the Filestore to the ActiveX application.

Events
FlyerFileUpDown(ByVal fileType As Integer, ByVal bytesTransferred As Integer, ByVal error As Integer)
Use this event to determine when a file transfer into or out of the Flyer head is complete.

Real Time Clock Management

Methods

The Flyer head contains a battery-backed real time clock, with daylight savings time options that are stored in the battery backed memory. The head supports all automated date and time code marking functions found in the WinMark software, including the ability to mark using custom date and shift codes.

Short GetHeadDateTime (Date as Date, bUTC as Boolean)
Short SetHeadDateTime (Date as Date, bUTC as Boolean)
Use these methods to read or write the current date and time values in the Flyer head.

Short GetHeadTimeInfoString (DSTString as String)
Short SetHeadTimeInfoString (strDSTString as String)
Use these methods to read or write the Daylight Savings Time definition data in the Flyer head.

Stand Alone Mark Control and Monitoring

Methods

Short GetMarkMode (bStandalone as Long, bStandaloneMarking as Long)
Use this method to retrieve the current mark mode and marking status of the Flyer head.

Long GetSAMarkStats (minCycleTime as Long, maxCycleTime as Long, curCycleTime as Long, curPieceCount as Long, PieceCountTotal as Long)
Use this method to retrieve the cycle time and cycle count statistics from the Flyer head.

Short SAMark (none)
Use this method to command the Flyer head to mark the file currently loaded into RAM.

Short ForceAbort (bState as Boolean)
Use this method to command the Flyer head to abort the current mark session.

Boolean CheckForAbort ()
Use this method to confirm that the mark session has been aborted.

Short SetSAConfiguration (bEnable as Boolean)
Use this method to configure the Flyer head to enter or exit Stand Alone marking mode.

Long SAMarkLogLevel (nLevel as Long)
The Flyer head creates a Mark Log that is stored in the head, containing information regarding mark start and stop times, piece count, progress through stages of automation, and error messages. The head also reports this information as log messages returned to the PC. The SAMarkLogLevel value determines the amount of this data is to be logged and reported.

Short UseSAControlFile (bUse as Boolean)
Use this method to configure the Flyer head to use a Master Control File to control the loading and marking of files based upon the status of the head's discrete I/O (refer to the Flyer head user's manual for full details of MCF marking).

Events

FlyerEOMStatus(ByVal iStatus as Long)
Use this event to detect that the Flyer head has finished a Stand Alone mark session. If the mark count is something other than 1 you will not see this event fire after each mark, but just at the end of the session.

FlyerLog(ByVal strMessage As String)
Use this event to detect and process log messages as they are sent from the Flyer head. The log messages provide mark session status information such as mark start/end, total session duration, automation messages, etc, depending on the setting of the Flyer's SAMarkLogLevel property.

FlyerProgress(ByVal strMessage As String)
Use this event to detect and process progress messages as they are sent from the Flyer head. The progress message provides the updated count of mark cycles completed in the mark session. If the mark file's mark count is a non-zero value, the message will be 'n of x' where n is the current count and x is the desired mark count. If the mark count is set to zero, then only the current count is returned.

Detailed Method Descriptions
All the methods that have been added to the V5 ActiveX control, listed alphabetically. Typical return values are given where appropriate, but you can find the full list of error codes in Appendix B.

Boolean CheckForAbort ()
Call this method immediately after calling ForceAbort to verify that the Stand Alone mark session was aborted. If the session was aborted, a True value is returned.

Note that the True result is returned only on the first call after the abort.

Short ConnectPreviewWnd(long pParent, long pPreview)
Reads the window handle of an object (such as a PictureBox in VB), then assigns the WinMark ActiveX preview window handle to this object for display of the marking preview window at run-time.

This is useful as it removes the development crashes in VB when placing the SynMhAtx object onto a form, viewing the form and starting the application. (Viewing the form opens up the ActiveX control within the VB6 IDE and running the debug version also opens up another instance within the same process. This causes the VB6 IDE to crash and developers to lose any unsaved data.)

Note that another way to avoid this issue in VB is to just close the form view before running the code.

Currently the pParent variable is not used, but may be used in the future.

Short ConnectToEthernetFlyer(BSTR strAddress)
Establishes an Ethernet connection to a Flyer head at a given IP address. The value of strAddress must be a string representing an IP address such as "192.168.0.5".

Typical return values:
≥ 0 (the head Index) successful retrieval of data value
- 3 the head is unavailable

Short CopyFile (strPathFrom as String, strPathTo as String)
FH Flyer only. Copies one file within the Flyer Filestore to another file within the Filestore.

Pass the strPathFrom and strPathTo values with a preceding '/' character. If the file is to be copied from or to a subdirectory, include the preceding slash character along with the directory name, another slash and the filename, such as '/Directory1/test.mkh'.

If you copy the file to a filename that is already in use, the previous file will be overwritten.

Typical return values:
0 file copy succeeded
- 12 file copy failed

Short DeleteFSObject (strPath as String)
Deletes a file from the Flyer Filestore. Pass the name of the file to be deleted with a leading '/' character, such as '/Canvas.mkh' to delete the 'Canvas.mkh' file that is located in the Filestore's root directory.

Note that there is no 'wastebasket' function in the Filestore… once you delete a file, it is gone.

Typical return values:
0 file delete succeeded
- 16 file delete failed

Short DeviceSimulation(Short mhType)
Creates a simulated head in your application.

Pass the mhType as:
1 for FH style heads
2 for SmartFH heads
3 for Scanlab heads
4 for Flyer heads

The short integer returned by the method is currently not implemented… the value is always zero.

Short DownloadFile (strPathFrom as String)
Downloads a mark file from the PC to the Flyer Filestore, saving the file as 'Canvas.mkh' in the Filestore's root directory. The new file will overwrite whatever is already contained in the Canvas.mkh file in the Filestore. This method is useful when fine tuning mark parameters, using the Canvas.mkh file as a temporary file buffer. Once the file settings are satisfactory, the CopyFile method may be used to save the file to a permanent and meaningful file name.

Pass the strPathFrom value as the entire path and filename for the source file, such as 'C:\Program Files\WinMark\Test.mkh'.

Typical return values:
0 file download succeeded
- 7 file download failed

Short ForceAbort (bState as Boolean)
Commands the Flyer head to abort the current stand-alone mark session.

The bState Boolean value determines whether the abort event is entered into the Flyer head log file or not.

Short GetCurrentMarkingHead()
Returns the index of the currently selected marking head, 0 based.

Typical return values:
- 1 no marking head is selected
- 2 the currently selected index is invalid (less than zero, or greater than the highest valid index)
- 3 the head is unavailable

Short GetDigitalString(BSTR pData)*
Returns a string representation (pData) of the current state of the marking head inputs.

Note that repeated calling of this method can bog down the communications to the Flyer head. If you intend to use this method to continuously poll the inputs to wait for a particular input state before progressing through your code, we encourage you to investigate the use of the TrigWaitDigEvent method/FlyerWaitDigital event or InputChangeDigital method/FlyerChangeDigital event as alternative approaches.

The function prefers one of 2 string lengths, 8, or 16 characters terminated by a NULL character:

- If pData is an eight character string, it will represent the input byte with the first character as bit 0 and the last as bit 7.
- If pData is a 16 character string, the first character represents input 0 and the 16th character will be input 15.
- If pData is any other length that is greater than 8 bytes but not 8 or 16 bytes, the function will presume an 8 character string as the default.

If the marking head has fewer input bits than used in the character string, the invalid bits will be set to zero.

Typical return values:

≥ 0 (the head Index) successful retrieval of data value
- 1 no head is currently selected
- 3 the head is unavailable
- 4 incorrect data format (for example, pData is less than 8 characters in length)

Short GetFilestoreInfo (BufferSize as Long, Buffer as String)
Retrieves a string containing the names of the files currently residing in the Filestore.

The BufferSize argument sets the length of the string returned in Buffer.

The string value returned in Buffer is a list of the files and directories found in the Flyer filestore. Each file or directory name is preceded by a '/' character. For instance, a Buffer value of:

'/File1.mkh/File2.mkh/Directory1/File1.mkh'

would indicate that File1.mkh and File2.mkh reside in the filestore root directory, with a subdirectory named Directory1 also containing a File1.mkh file.

Typical return values:

0 successful retrieval of file information
- 20 invalid filestore size

Short GetFilestoreUsage (BytesUsed as Long, BytesAvailable as Long)
Retrieves the filestore memory usage and capacity in bytes.

Typical return values:

0 successful retrieval of filestore information
- 9 data response failure

Short GetFlyerTemperature(float pFront, float* pBack, boolean bOvertemp)*
Returns the Celsius temperature values read from the Flyer's internal temperature sensors: the value returned as pFront is read from the Power Amps temp sensor, the value returned as pBack is read from the CPU temp sensor. If either temperature exceeds the factory programmed overtemp point, bOvertemp returns True.

Typical return values:

≥ 0 (the head Index) successful retrieval of data
- 1 no head is currently selected
- 3 the head is unavailable
- 4 the strProp or strValue are not valid
- 5 head not a Flyer, no temp available

Short GetHeadDateTime (Date as Date, bUTC as Boolean)
Reads the current date and time values from the Flyer head. Set bUTC to True to retrieve the Date value in Universal Coordinated Time, set bUTC to False to retrieve the Date value as local time.

Typical return values:

0 successful retrieval of filestore information
- 9 data response failure

Short GetHeadTimeInfoString (DSTString as String)
Retrieves the Daylight Savings Time definition from the Flyer head.

The DSTString value is formatted as something like "PST+8PDT+7,M3.2.0/03:00,M11.1.0/02:00", where:

'PST8' indicates the standard time bias – in this case Pacific Standard Time will be used, with an -8 hour offset from UTC time

'PDT7' indicates the daylight time bias – in this case Pacific Daylight Time will be used, with a -7 hour offset from UTC time

'M3.2.0/03:00' indicates when the changeover from standard to daylight time occurs – in this case M3.2.0 indicates the second Sunday (day 0) of the third month (March). The '/03:00' indicates that the change occurs at 3AM on the specified date.

'M11.1.0/02:00' indicates when the changeover from daylight to standard time occurs – in this case M11.1.0 indicates the first Sunday (day 0) of the eleventh month (November). The '/02:00' indicates that the change occurs at 2AM on the specified date.

The short integer returned by the method is currently not implemented… the value is always zero.

Long GetHeadUptime (none)

Retrieves the number of seconds since the Flyer has been running since the last boot up sequence. This value resets every time the head is powered up or rebooted, so it indicates just the duration of the current running state, whether the head is marking or idle. Since this value is returned as a long integer, the value rolls over at 2,147,483,647 seconds (a little over 68 years).

Typical return values:

> 0 successful retrieval of uptime data

- 1 head not available

Short GetMarkHeadCount()

Returns the number of heads currently available on the system, including heads in use by other controls or WinMark. The function may indicate that 5 heads are connected, but only two may actually be free to be used by this control instance (because the other three are in use by other controls and/or WinMark). To determine which heads are available in a multiple head configuration, query each head index using the GetMarkHeadStatus method.

Short GetMarkHeadName(short Index, BSTR pName)*

Retrieve the name of a marking head by its index value. For Flyer heads, this is the name stored in its system properties and is limited to 255 characters (which is the size that pName should be set to accept). For other head types, this is the value that is stored in the WinMark registry. If the index is valid, this method returns the head's name through pName, otherwise the method will return an error code.

Typical return values:

- 2 the currently selected index is invalid (less than zero, or greater than the highest valid index)
- 3 the head is unavailable

Note that the Name of the head for Flyer can be changed through the SetSystemProp command.

Short GetMarkHeadStatus(short Index)

Returns the status of a head by its index value.

Typical return values:

1 the Index is valid and the head is available

0 the head is in use by another control or by WinMark

- 2 the currently selected index is invalid (less than zero, or greater than the highest valid index)
- 3 the head is unavailable

Short GetMarkHeadType(short Index)

Returns the type of the marking head by its index value.

Typical return values:

1 for FH style heads
2 for SmartFH heads
3 for Scanlab heads
4 for Flyer heads
- 1 for simulated heads
- 2 the currently selected index is invalid (less than zero, or greater than the highest valid index)
- 3 the head is unavailable

Short GetMarkMode (bStandalone as Long, bStandaloneMarking as Long)
Returns the current mark mode and marking status from the Flyer head.

The bStandalone value returns 1 if the head is in stand alone mode, 0 if it is not.

The bStandaloneMarking value is 1 if the head is in standalone mode and is marking, 0 if it is not.

The short integer returned by the method is currently not implemented… the value is always zero.

Short GetNetworkMount (bNetworked as Long)
Returns the status of the network share connection. Refer to the FH Flyer operator's manual for details on setting up the network share. The share may be disconnected for a number of reasons: the Ethernet cable to the head is disconnected, the share drive is down, or any of the share parameters in the Flyer head are incorrect.

The bNetworked value returns:

0 the network share is not connected
1 the network share is connected

The short integer returned by the method is currently not implemented… the value is always zero.

Long GetSAMarkStats (minCycleTime as Long, maxCycleTime as Long, curCycleTime as Long, curPieceCount as Long, PieceCountTotal as Long)
Retrieves the cycle time and cycle count statistics from the Flyer head.

The minCycleTime, maxCycleTime and curCycleTime values are all returned in tens of milliseconds, so that a value of 48 would indicate a cycle time of 0.48 seconds.

The curPieceCount value returns the number of completed mark cycles within the current mark session.

The PieceCountTotal value returns the mark count defined from the mark file. If the mark count is set to zero, the PieceCountTotal will return a value of zero and the file will continue marking until the session is aborted.

The short integer returned by the method is currently not implemented… the value is always zero.

Short GetSystemProp(BSTR strProp, BSTR pStrValue)*
Returns the property's value in a string format as pStrValue. Be sure to allocate an adequate string variable size, no greater than 255 bytes, to pStrValue that will be sufficient to hold the return data value.

Refer to Appendix A for a complete list of the current Flyer properties.

Typical return values:

≥ 0 (the head Index) successful retrieval of data value
- 1 no head is currently selected
- 3 the head is unavailable
- 4 the strProp or strValue are not valid

Short GetSystemPropertyCount()
Returns the count of system properties on the current Flyer head.

Typical return values:

≥ 0 successful retrieval of data
- 1 no head is currently selected
- 3 the head is unavailable

Short GetSystemPropertyNameByIndex(short PropIndex, BSTR pStrName)*

For a given property index value, returns the name of the Flyer head system property as pStrName. Use GetSystemProperyCount to verify that the property index you are using is within the range available in the head.

Typical return values:

≥ 0 (the head Index) successful retrieval of data
- 1 no head is currently selected
- 3 the head is unavailable
- 4 PropIndex is out of range or pStrName is NULL

Short InputChangeDigital (String strMask)

Use this method to enable or disable the Flyer head's FlyerChangeDigital event.

Pass the value strMask as an eight character string made up of '0'or '1' characters:

'1' report any change to the input
'0' ignore changes to the input

The strMask value is ordered with input bit 0 on the left end, input bit 7 on the right. For instance, using a strMask value of '11100000' commands the Flyer head to fire the FlyerChangeDigital event every time a change in state is detected on IN0, IN1 or IN2. The '0' values commands the Flyer head to ignore changes to inputs IN3-7.

Use a strMask value of '00000000' to disable the FlyerChangeDigital event.

The event fires on every change of value on the input(s) of interest, although the head has a maximum event firing rate of 3 to 4 events per second. There is no timeout value on the InputChangeDigital trigger.

Typical return values:

≥ 0 (the head Index) method successful
- 1 no head is currently selected
- 3 the head is unavailable

Short MakeDir (strDirPath as String)

Creates a subdirectory within the Filestore. Subdirectory names are preceded with a forward slash character (/) and may be nested. For instance, the root directory may contain a directory named /SubDirectory1 which then contains /Subdirectory1a.

Note that the directory name 'Flyer Fonts' is reserved for the use of downloaded font files.

Typical return values:

0 directory creation successful
- 22 the directory creation failed – possibly a directory of the same name already exists

Short OpenFile (strPath as String)

Loads a file from the Filestore into Flyer RAM to prepare for marking. File names must be preceded with a forward slash character (/).

Typical return values:

0 file load successful
- 17 file load failed

Short RebootHead (none)
Forces a 'soft' reboot of the Flyer head. The head takes about 30 seconds to reboot, during which time it will be unavailable. If you're using the USB port, the *FlyerConnectNotification* event will fire off after the head has finished the boot up process.

Typical return values:
- 0 head reboot successful
- 14 head reboot failed

Short ReformatFileStore (none)
Erases all MKH, DAT, TTF and MCF files within the Flyer FLASH memory.

Typical return values:
- 0 reformat successful
- 10 reformat failed

Short RenameFSObject (strOldPath as String, strNewPath as String)
Renames one of the files in the Flyer Filestore. File names must be preceded with a forward slash character (/).

If the file specified by strNewPath already exists, it will be overwritten.

Typical return values:
- 0 file rename successful
- 12 file rename failed

Short SAGetCurrentFilePath (strFileName as String)
Reads the filename and path of the mark file currently loaded into the Flyer RAM for marking.

Typical return values:
- 0 get file path successful
- 17 get file path failed

Short SAMark (none)
Commands the Flyer head to mark the file currently loaded into RAM.

Typical return values:
- 0 SAMark successful
- 11 object timeout period exceeded – did not get a response from the head
- 18 head is busy in standalone mode

Long SAMarkLogLevel (nLevel as Long)
Sets the level of log data stored in the Flyer head marking log and reported through the FlyerLog event.

The nLevel argument corresponds to the Flyer's 'Standalone Mark Log Level' property:
- 0 = 'Errors Only'
- 1 = 'Light'
- 2 = 'Normal'
- 3 = 'Verbose'
- 4 = 'Everything'

Typical return values:
- 0 SAMarkLogLevel set successfully
- 15 file properties are invalid

Short SaveToFileStore (strPathFrom as String, strPathTo as String)
Downloads a file to the Flyer Filestore, specifying the name the file should have within the Filestore.

Pass the file name in strPathFrom formatted with the full path and filename, such as 'C:/Program Files/WinMark/MyMark.mkh'.

Pass the file name in strPathTo preceded with a forward slash character (/MyFile.mkh) to place the file in the root directory. To place the file in a subdirectory, use the forward slash, directory name and another slash (/Directory1/MyFile.mkh).

If the strPathTo filename already exists in the Filestore, it will be overwritten by the SaveToFileStore operation.

Typical return values:
0 file download successful
- 7 file download failed

Short SetAsMarkOnStartup (strPath as String)
Configures the Flyer head 'StartUpDrawing' and 'MarkOnStart' properties to mark the specified file at startup.

Pass the file name in strPath preceded with a forward slash character (/MyFile.mkh) if the file is in the root directory. If the file is in a subdirectory, use the forward slash, directory name and another slash (/Directory1/MyFile.mkh).

Typical return values:
0 properties successfully set
- 15 file properties are invalid

Short SetCurrentMarkingHead(short Index)
Select a head, by its index value, to be the current marking head.

Typical return values:
≥ 0 (the head Index) head selection successful
- 1 no marking head is selected
- 2 the currently selected index is invalid (less than zero, or greater than the highest valid index)
- 3 the head is unavailable

Short SetDigitalString(BSTR pData)
Sets the outputs of the current head according to the string representation (pData) passed to the function.

The function prefers one of 2 string lengths, 8, or 16 characters terminated by a NULL character:

- If pData is an eight character string, it will represent the input byte with the first character as bit 0 and the last as bit 7.
- If pData is a 16 character string, the first character represents input 0 and the 16th character will be input 15.
- If pData is any other length that is greater than 8 bytes but not 8 or 16 bytes, the function will presume an 8 character string as the default.

Valid characters used in pData are:
- do not change output bit from previous value
0 clear/turn off the output bit
1 set/turn on the output bit

If the marking head has fewer output bits than used in the character string, the invalid bits will be set to zero.

Typical return values:

≥ 0 (the head Index) successful retrieval of data value
- 1 no head is currently selected
- 3 the head is unavailable
- 4 pData is less than 8 characters in length or contains invalid characters

Short SetHeadDateTime (Date as Date, bUTC as Boolean)
Writes the current date and time values to the Flyer head.

Pass bUTC as true to indicate that Date is UTC time (Universal Coordinated Time, formerly known as GMT), false to indicate that Date is local time.

Typical return values:
0 date/time successfully set
- 3 head not available

Short SetHeadTimeInfoString (strDSTString as String)
Writes the Daylight Savings Time definition data to the Flyer head.

The DSTString value is formatted as something like "PST+8PDT+7,M3.2.0/03:00,M11.1.0/02:00", where:
'PST8' indicates the standard time bias – in this case Pacific Standard Time will be used, with an -8 hour offset from UTC time
'PDT7' indicates the daylight time bias – in this case Pacific Daylight Time will be used, with a -7 hour offset from UTC time
'M3.2.0/03:00' indicates when the changeover from standard to daylight time occurs – in this case M3.2.0 indicates the second Sunday (day 0) of the third month (March). The '/03:00' indicates that the change occurs at 3AM on the specified date.
'M11.1.0/02:00' indicates when the changeover from daylight to standard time occurs – in this case M11.1.0 indicates the first Sunday (day 0) of the eleventh month (November). The '/02:00' indicates that the change occurs at 2AM on the specified date.

The short integer returned by the method is currently not implemented… the value is always zero.

Short SetSAConfiguration (bEnable as Boolean)
Commands the Flyer head to enter (bEnable = True) or exit (bEnable = False) Stand Alone marking mode.

Typical return values:
0 head configuration successful
- 15 head configuration failed

Short SetSystemProp(BSTR strProp, BSTR strValue)
Sets the Flyer head's property named in strProp to the value given as strValue.

You may also use this function to read and write user-defined 'property' data (no more that 255 bytes per value, no more than 1024 values total) in the Flyer head. If you use the SetSystemProp method to write data to a property name that is not recognized by the Flyer head, it will store the property name and the corresponding data to a separate data register in the Flyer's FLASH memory. Since these values are written to FLASH, this is not a good place to store data that is frequently changing such as serial numbers that change on every mark.

Refer to Appendix A for a complete list of the current Flyer properties.

Typical return values:
≥ 0 (the head Index) successful retrieval of data value
- 1 no head is currently selected
- 3 the head is unavailable
- 4 the strProp or strValue arguments are not valid

- 5 strValue is invalid for the property data type or range
- 6 the property is read only

For example, the function would return – 5 if strProp = "EncoderResolution" and strValue = "thirty" rather than "30.0".

Short TrigWaitDigEvent(String strBits)
Use this method to command the Flyer head to fire the FlyerWaitDigital event when the input condition defined by strBits is true.

Pass the value strBits as an eight character string made up of '0', '1' or '–' characters:
'1' input must be active
'0' input must be inactive
'–' don't care… input may be active or inactive

The strBits value is ordered with input bit 0 on the left end, input bit 7 on the right. For instance, a strBits value of '100-----' would command the Flyer head to fire the WaitDigital event whenever input 0 is active, inputs 1 and 2 are inactive, with no regard as to the states of inputs 3-7.

The event fires just once upon the first input match to the strBits value. There is no timeout value on the WaitDigital trigger.

Typical return values:
≥ 0 (the head Index) method successful
- 1 no head is currently selected
- 3 the head is unavailable

Short UploadFile (strPathFrom as String, strPathTo as String)
Uploads a file from the Filestore to the PC.

Pass the strPathFrom value with a preceding '/' character. If the file is to be copied from a subdirectory, include the preceding slash character along with the directory name, another slash and the filename, such as '/Directory1/test.mkh'.

Pass the strPathTo value as the entire path and filename for the destination file, such as 'C:\Program Files\WinMark\UploadTest.mkh'.

Typical return values:
0 file upload succeeded
- 8 file upload failed

Short UseSAControlFile (bUse as Boolean)
Commands the Flyer head to use (bUse = True) or not use (bUse = False) a Master Control File to control Stand Alone marking mode.

Typical return values:
0 head configuration successful
- 15 head configuration failed

Detailed Event Descriptions

FlyerChangeDigital(ByVal strMessage As String)

This event fires off on every change of state detected on the inputs of interest as defined by the strMask value passed through the InputChangeDigitial method.

The strMessage value returns the state of all eight inputs at the time of the event, with input bit 0 on the left, input bit 7 on the right. For instance, if strMessage = "10110000", input bits 0, 2 and 3 are all active (on); input bits 1, 4-7 are all inactive (off).

FlyerConnectNotification(ByVal Index As Integer, ByVal bConnecting As Long)

This event fires every time the Flyer USB driver detects a change in the USB connection status. The event returns the index of the head involved, and whether the change was due to a head connecting (bConnecting = 1) or disconnecting (bConnecting = 0).

FlyerEOMStatus(ByVal iStatus As Long)

Fires off when the Flyer head has completed a mark session in Stand Alone mode. If the mark count is something other than 1 you will not see this event fire after each mark, but just at the end of the session.

FlyerFileUpDown(ByVal fileType As Integer, ByVal bytesTransferred As Integer, ByVal error As Integer)

Fires off after a file has been download to or uploaded from the Flyer filestore.

FlyerLog(ByVal strMessage As String)

Fires off whenever the Flyer head generates a log message. The log messages provide mark session status information such as mark start/end, total session duration, automation messages, etc, depending on the setting of the Flyer's SAMarkLogLevel property (refer to the SAMarkLogLevel method for more detail).

FlyerProgress(ByVal strMessage As String)

Fires off whenever the Flyer head generates a progress message. The progress message provides the updated count of mark cycles completed in the mark session. If the mark file's mark count is a non-zero value, the message will be 'n of x' where n is the current count and x is the desired mark count. If the mark count is set to zero, then only the current count is returned.

FlyerWaitDigital(ByVal strMessage As String)

The event fires off on the first detection of an input combination that satisfies the requirements of strBits passed through the TrigWaitDigEvent method. Once the event has fired off, the head clears out the associated input trigger definition – you must again call the TrigWaitDigEvent method to trigger the head to report subsequent input states.

The strMessage value returns the state of all eight inputs at the time of the event, with input bit 0 on the left, input bit 7 on the right. For instance, if strMessage = "10110000", input bits 0, 2 and 3 are all active (on); input bits 1, 4-7 are all inactive (off).

Appendix A – FLYER Head Properties

The current valid list of system properties to be set on Flyer are:

“FlyIpAddress"	Address to be used if the head is set to static, IP address form only like “192.168.0.5”.
"FlyIpMask"	The net mask to be used if the head is set to static IP addressing, typically “255.255.255.0”.
"FlyIpBroadcast"	The broadcast address to be used if the head is set to static IP addressing, typically “255.255.255.255”.
"FlyIpGateway	The gateway address to be used if the head is set to static IP addressing.
"FlyIpDns1"	The primary DNS server address to be used if the head is set to static IP addressing.
"FlyIpDns2"	The secondary DNS server address to be used if the head is set to static IP addressing.
"FlyUseDHCP"	“1” is grab an IP address dynamically from the DHCP server (dynamic), “0” configure the address using the IP information in this configuration (static).
"FlyIpAllow(0-9)”	A way to limit which computers have access to the head. For example “192.168.0.0/255.255.255.0) limits the allowable IPs to the 192.168.0 subnet.
"FlyIpDeny (0-9)”	A way to deny computers access to the head. For example “192.168.0.5/255.255.255.255) denies access of 192.168.0.5 to the head. <u>This is not yet implemented.</u>
"EncoderResolution"	Floating point number in pulses per mm “12.93” for example.
“InvertEncoderDirection"	1 for true, 0 for false.
"RisingEdgePartSense"	1 for true, 0 for false.
"SensorDistance"	Floating point number giving the Tracking sensor distance in inches.
"QuadEncoder"	1 for true, 0 for false.
"UseEncoderless"	1 for true, 0 for false.
"UseFixedPartPitch"	1 for true, 0 for false.
"PartPitch"	Floating point number representing the part to part distance in inches for encoderless tracking.
"LineSpeed"	Floating point number in inches/sec for encoderless tracking.
"MotionVector"	Floating point number between 0-360 giving the orientation of tracking.
"ObjectName"	This is the name of the head and in WinMark is what is shown in the mark button.
"FlyLens"	an integer representing the lens used on the head. The currently allowed selections are: 80 mm lens 6 125 mm lens 5 200 mm lens 2 370 mm lens 4 125HP mm lens 7 User Defined 128 Any number other than these and the head will default to 200 mm lens(2).
“StartUpDrawing"	String containing the path on the head to mark on startup for standalone marking.
"MarkOnStart"	“1” for true, “0” for false, (standalone marking).
"CustomDateCode(0-?)"	String representing the custom date codes (standalone marking)
"ShiftCodes(0-23)"	String representing the shift code (standalone marking)
"FlyUbootVersion"	Read-only string giving the U-boot bootloader version.
"FlyKernelVersion"	Read-only string giving the Linux kernel version.
"FlyVersion"	Read-only string giving the firmware version.
"FlySerialNumber"	Read-only string giving the Flyer serial number.
"FlyMacAddress"	Read-only string giving the Flyer MAC address.
"KeyboardLocked"	“1” lock the keyboard, “0” unlock the keyboard.

"FlyFasiEnable"	Read-only string showing whether the FASI switch is on or off.
"FlyPwmGate"	Read-only string showing whether the PWM output is switched to PWM or a gate signal.
"FlyTickleDisable"	Read-only string showing whether the tickle is switched to disable or normal(with tickle).
"FlyShareUser"	String giving the user name to connect to a windows share on the network, Not fully implemented.
"FlySharePassword"	password for the user name to connect to a windows share on the network, Not fully implemented.
"FlyShareServer"	String giving the IP address or DNS name of the server (if DNS is setup correctly). Not fully implemented.
"FlyShareName"	The "path" to the share "\\server\path". Not fully implemented.
"FlyShareReadOnly"	"1" - connect to the share with read-only access "0" - connect to the share with the same rights normally given to the username in "FlyShareUser" , Not fully implemented.
"ClearingMarkInterval" :	(standalone marking only) Integer representing the clearing mark interval in mark cycles, 0 is off.
"ClearingMarkSessionStart"	(standalone marking only) "1" - run a clearing mark at the mark session start "0" - do NOT run a clearing mark at mark session start

Appendix B – V5 ActiveX Error Codes

-1 NO_HEAD_CONNECTED
-2 HEAD_INDEX_BAD
-3 HEAD_UNAVAILABLE
-4 BAD_FORMAT
-5 BAD_TYPE
-6 TYPE_READ_ONLY
-7 FILE_DOWNLOAD_FAIL
-8 FILE_UPLOAD_FAIL
-9 DATA_RESPONSE_FAIL
-10 REFORMAT_FAIL
-11 OBJECT_TIMEOUT
-12 FILE_COPY_FAIL
-13 FILE_MOVE_FAIL
-14 REBOOT_FAIL
-15 BAD_FILE_PROPS
-16 FILE_DELETE_FAIL
-17 FILE_OPEN_FAIL
-18 STANDALONE_HEAD_BUSY
-19 CORRUPT_FILESYSTEM
-20 INVALID_FILESTORE_SIZE