Table of Contents

Index

1 About MegaPipe ATL ActiveX

1.1 Introduction

MegaPipe ATL ActiveX is a reliable and powerful library for handling serial communication, modem operation and file-transfer (XModem Checksum, XModem CRC, XModem 1K, YModem, YModem-G, ZModem and Kermit), it is usable with a variety of popular application development environments such as VC++, VB, VB.Net, C#.Net, Access, Delphi and Borland C++.

MegaPipe ATL ActiveX supports multiple port/phone line (up to 8) communications simultaneously, each communication channel is uniquely identified by a channel ID (0-7).

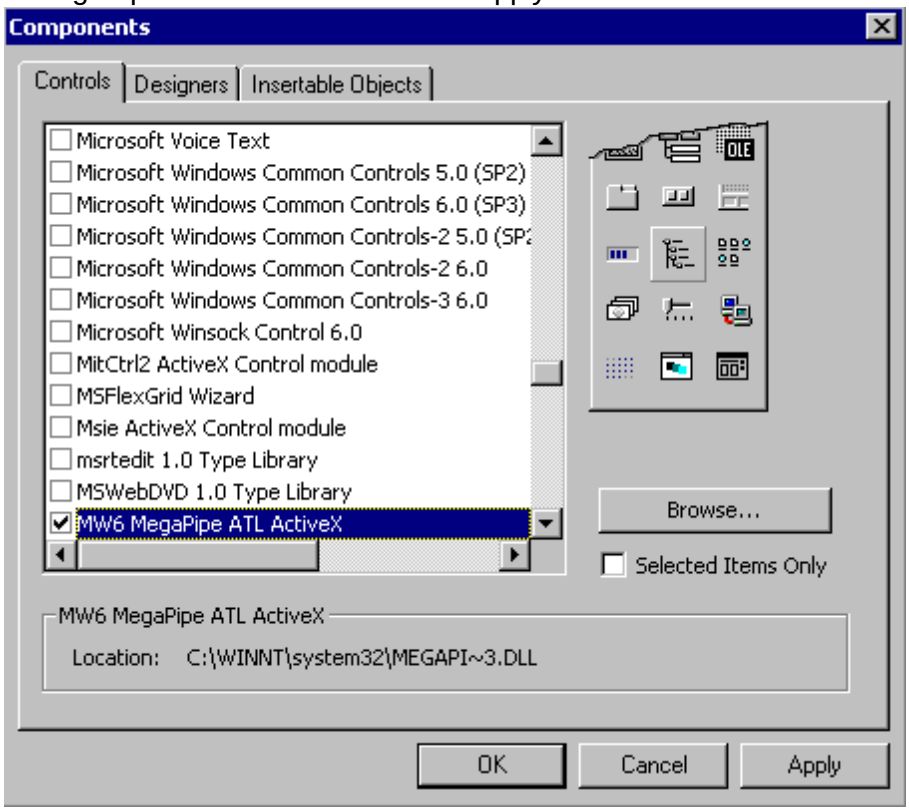
1.2 How to Use It

1.2.1 Trial Version

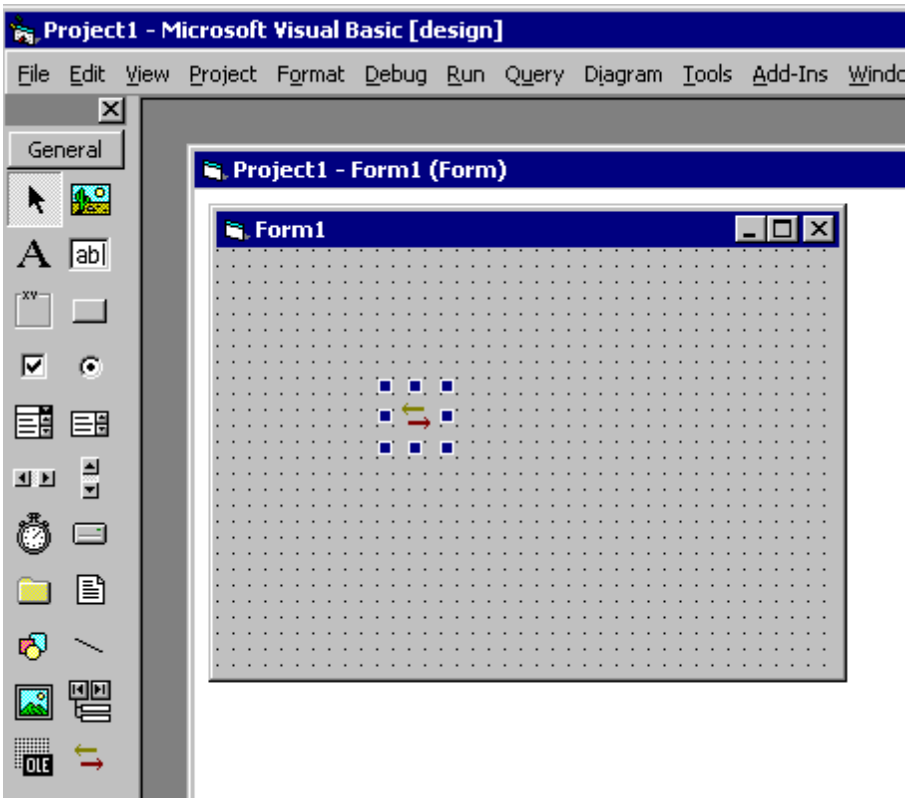
It is very easy to add the MegaPipe ATL ActiveX to your project in any ActiveX compliant IDE such as Visual Basic 6.0, Visual C++ 6.0, Access or Visual FoxPro.

1. After the installation for the trial version is finished, your ActiveX compliant IDE should be able to recognize the MegaPipe ATL ActiveX.

2. Assume you are using Visual Basic 6.0, click "Project" > "Components", choose "MW6 MegaPipe ATL ActiveX" and click "Apply".



3. Drag and drop the MegaPipe ATL ActiveX on your windows form.



1.2.2 Full Version

Follow the instructions listed below to add the full version MegaPipe ATL ActiveX to your project:

1. Uninstall the trial version and complete the installation for the full version.

2. The license key method SetKey should be called before you call any other MegaPipe method.

```
SetKey(BSTR lpKey, VARIANT_BOOL *ValidKey)
```

Parameters

Key

10 digits license key obtained from us.

ValidKey

If the license key is valid, the value of variable pointed to by ValidKey is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

Examples

```
[Visual Basic]

Dim ValidKey As Boolean
MegaPipeObj.SetKey("XXXXXX-XXXX", ValidKey)
```

1.3 How to Distribute It

If you want to redistribute the MegaPipe ATL ActiveX as part of your application, please follow the instructions below:

1) For 32-bit version Windows OS, put **MegaPipeCtrl.dll** into the windows 32-bit system folder (e.g. "c:\windows\system32" or "c:\winnt\system32") on the target machine and run "regsvr32 MegaPipeCtrl.dll" to register it.

2) For 64-bit version Windows OS, put **MegaPipeCtrl.dll** into the SysWOW64 folder (e.g. "c:\windows\SysWOW64") on the target machine, and run the following commands to register it:
   - cd \windows\SysWOW64
   - regsvr32 MegaPipeCtrl.dll

3) For Windows Vista or above, you need to use an elevated Command Prompt to run *regsvr32.exe* command, click "**Start"** > "**All Programs**" > "**Accessories**", right-click "**Command Prompt**", and then click "**Run as administrator**".

2 Reference Guide

2.1 Serial Communication

2.1.1 ClosePort Method

Closes the serial port for a communication channel (direct serial connection or modem connection).

```
ClosePort(short ChannelID, VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

See Also

OpenPort Method

2.1.2 EscapeCommFunc Method

Performs an extended method for a communication channel (direct serial connection or modem connection).

```
EscapeCommFunc(short ChannelID, long Func, VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

Func

Extended method to be performed, this parameter can be one of the following values:

dwFunc Value	Comment
3	Sets RTS (request-to-send) line
4	Clears RTS (request-to-send) line
5	Sets DTR (data-terminal-ready) line
6	Clears DTR (data-terminal-ready) line
7	Resets device if possible
8	Sets the device break line
9	Clears the device break line

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

2.1.3 GetCommStatus Method

Retrieves modem control-register values for a communication channel (direct serial connection or modem connection).

```
GetCommStatus(short ChannelID, long *ModemStat, VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

ModemStat

A long pointer to a 32-bit variable that specifies the current state of the modem control-register values. This parameter can be a combination of the following values:

Value	Comment
MS_CTS_ON	The CTS (Clear To Send) signal is on.
MS_DSR_ON	The DSR (Data Set Ready) signal is on.
MS_RING_ON	The ring indicator signal is on.
MS_RLSD_ON	The RLSD (Receive Line Signal Detect) signal is on.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

2.1.4 GetFeedback Method

Retrieves the information for a communication channel (direct serial connection or modem connection) when an error or a warning occurs.

```
GetFeedback(short ChannelID, BSTR *Msg, long *Size, VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

Msg

A pointer to a string containing the information.

Size

A pointer to the variable that receives the size (in characters) of the information.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

2.1.5 GetInputDataCount Method

Retrieves the number of incoming data bytes available in the input buffer for a communication channel (direct serial connection or modem connection).

```
GetInputDataCount(short ChannelID, long *Count);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

Count

A pointer to the variable that receives the number of incoming data bytes available in the input buffer.

See Also

ReceiveData Method

2.1.6 GetPortStatus Method

Retrieves the port status for a communication channel (direct serial connection or modem connection).

```
GetPortStatus(short ChannelID, long *Status);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

Status

The value of 1 for the variable pointed to by Status indicates that the port is open, the value of 0 for the variable pointed to by Status indicates that the port is closed.

2.1.7 OpenPort Method

Opens the serial port for a communication channel (direct serial connection or modem connection).

```
OpenPort(short ChannelID, VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

See Also

ClosePort Method

2.1.8 ReceiveData Method

Reads data from the input buffer for a communication channel (direct serial connection or modem connection).

```
ReceiveData(
  short ChannelID,
  short *Buffer,
  long NumberOfBytesToReceive,
  long *NumberOfBytesReceived,
  VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

Buffer

A pointer to the buffer that receives the data read from the input buffer.

NumberOfBytesToReceive

The minimum number of bytes to read.

NumberOfBytesReceived

A pointer to the variable that receives number of bytes successfully read from the input buffer.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

2.1.9 SendData Method

Transmits data for a communication channel (direct serial connection or modem connection).

```
SendData(
  short ChannelID,
  short *Buffer,
  long NumberOfBytesToSend,
  long *NumberOfBytesSent,
  VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

Buffer

A pointer to the buffer containing the data to be transmitted to the remote side.

NumberOfBytesToSend

Number of bytes to be transmitted to the remote side.

NumberOfBytesSent

A pointer to the variable that receives number of bytes successfully transmitted.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

2.1.10 SetPortParam Method

Sets up the serial port parameters for a communication channel (direct serial connection) or assigns a communication channel to a modem.

```
SetPortParam(
  short ChannelID,
  BSTR PortName,
  long BaudRate,
  short DataBits,
  short StopBits,
  short Parity,
  short FlowControl,
  BOOL UseTAPI,
  short ModemIndex,
  VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

PortName

A string containing the serial port name (e.g. "COM1").

BaudRate

The baud rate of the transmission (e.g. 9600).

DataBits

The data bits of the transmission, this parameter can be one the following values:

Value	Comment
4	4 data bits
5	5 data bits
6	6 data bits
7	7 data bits
8	8 data bits

StopBits

The stop bits of the transmission, this parameter can be one the following values:

Name	Comment
0	1 stop bit
1	1.5 stop bits
2	2 stop bits

Parity

The parity of the transmission, this parameter can be one the following values:

Name	Comment
0	No parity
1	Odd parity
2	Even parity
3	Mark parity
4	Space parity

FlowControl

The flow control of transmission, this parameter can be one the following values:

Value	Comment
0	No flow control
1	Xon/Xoff software control
2	Hardware control

UseTAPI

Indicates whether a communication channel uses modem related Methods to handle the phone communication or not, if it is FALSE, ignore nModemIndex parameter.

ModemIndex

Used to assign a communication channel to a modem, this parameter is a 0-based index and a valid value must be between 0 and total number of modems - 1.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

Remarks

If you use modem-related method (e.g. MakeCall Method and WaitForCall Method) to handle the phone communication, your application doesn't need to care about lpPortName, dwBaudRate, nDataBits, nStopBits, nParity and nFlowControl parameters, Micrsoft TAPI will take care of them automatically.

2.2 Modem

2.2.1 CloseTAPI Method

Closes TAPI after you finish TAPI-related modem operation(s) for all communication channels.

```
CloseTAPI(VARIANT_BOOL *Result);
```

Parameters

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

Remarks

If InitTAPI API is called, this Method must be called in order to shut down TAPI methods properly.

If a phone line connection is established, be sure to set the DropCall property to true before call this method.

See Also

InitTAPI Method

2.2.2 DropCall Method

Cuts off current established phone line connection for a communication channel inexplicitly identified by a modem index.

```
DropCall(short ChannelID, VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

Remarks

Use SetPortParam method to define a relationship between a modem index and a communication channel ID (0-7).

See Also

SetPortParam Method | GetModemCount Method

2.2.3 InitTAPI Method

Initializes TAPI before your application conducts TAPI-related modem operation(s) for all communication channels.

```
InitTAPI(VARIANT_BOOL *Result);
```

Parameters

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

2.2.4 GetLineStatus Method

Retrieves phone line status for a modem communication channel.

```
GetLineStatus(short ChannelID, long *Status);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

Status

The value of variable pointed to by Status can be one of the following values:

Value	Comment
1	There is an incoming call.
2	The call is proceeding.
3	The line is connected.
4	The line is disconnected.
5	The line is busy, please dial later.
6	No dial tone was detected.
7	The remote side does not answer.

Remarks

Use SetPortParam method to define a relationship between a modem index and a communication channel ID (0-7).

See Also

SetPortParam Method | GetModemCount Method

2.2.5 GetModemCount Method

Retieves the number of modems installed on PC.

```
GetModemCount(long *Count);
```

Parameters

Count

A pointer to the variable that receives the number of modems installed on PC.

2.2.6 GetModemName Method

Retrieves the name of a modem explicitly identified by a modem index.

```
GetModemName(short ModemIndex, BSTR *ModemName, long *Size, VARIANT_BOOL *Result);
```

Parameters

ModemIndex

This parameter is a 0-based index and a valid value must be between 0 and total number of modems - 1.

ModemName

A pointer to a buffer that receives a null-terminated character string containing the modem name.

Size

A pointer to the variable that receives the size, in characters, of the modem name string.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

Remarks

Use SetPortParam method to define a relationship between a modem index and a communication channel ID (0-7).

See Also

SetPortParam Method | GetModemCount Method

2.2.7 GetModemPort Method

Retrieves the name of the port associated with a modem explicitly identified by a modem index.

```
GetModemPort(short ModemIndex, BSTR *PortName, long *Size, VARIANT_BOOL *Result);
```

Parameters

ModemIndex

This parameter is a 0-based index and a valid value must be between 0 and total number of modems - 1.

PortName

A pointer to a buffer that receives a null-terminated character string containing the port name.

Size

A pointer to the variable that receives the size, in characters, of the port name string.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

Remarks

Use SetPortParam method to define a relationship between a modem index and a communication channel ID (0-7).

See Also

SetPortParam Method | GetModemCount Method

2.2.8 MakeCall Method

Makes a phone call for for a modem communication channel.

```
MakeCall(short ChannelID, BSTR PhoneNumber, VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

PhoneNumber

A pointer to a null-terminated string containing the remote side phone number to dial in.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

Remarks

Use SetPortParam method to define a relationship between a modem index and a communication channel ID (0-7).

See Also

SetPortParam Method | GetModemCount Method

2.2.9 WaitForCall Method

Waits for a call for a modem communication channel.

```
WaitForCall(short ChannelID, VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

Remarks

Use SetPortParam method to define a relationship between a modem index and a communication channel ID (0-7).

See Also

SetPortParam Method | GetModemCount Method

2.3 File Transfer

2.3.1 GetXferStatus Method

Retrieves file-transfer status for a communication channel (direct serial connection or modem connection).

```
GetXferStatus(short ChannelID, long *Status);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

Status

The value of variable pointed to by Status can be one of the following values:

Value	Comment
1	A file-transfer session is aborted.
2	A file-transfer session is doing initialization.

3	Start to upload or download a file now.
4	One block of data are transferred successfully.
5	A file is uploaded or downloaded successfully.
6	A file-transfer session is finished successfully.

2.3.2 XferAddFile Method

Informs MegaPipe of the name of file which will be uploaded to the remote side for a communication channel (direct serial connection or modem connection).

```
XferAddFile(short ChannelID, BSTR FileName, VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

FileName

A pointer to a null-terminated string containing the relevant path name of file which will be uploaded to the remote side.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

Remarks

All XModem protocols can only upload 1 file during one file-transfer session, your application only needs to call this API once for a communication channel.

YModem, YModem-G, ZModem and Kermit can upload multiple files during one file-transfer session, so your application maybe needs to call this API a few times if multiple files are uploaded for a communication channel.

2.3.3 XferClearAllFiles Method

Clears file name information in MegaPipe memory on the upload side for a communication channel (direct serial connection or modem connection).

```
XferClearAllFiles(short ChannelID, VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

Remarks
Call this API before you call XferAddFile() API.

2.3.4 XferGetCurrBytes Method

Retrieves the number of data bytes sent/received so far for a communication channel (direct serial connection or modem connection).

```
XferGetCurrBytes(short ChannelID, long *Count);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

Count

A pointer to the variable that receives the number of data bytes sent/received so far.

2.3.5 XferGetCurrFileName Method

Retrieves the name of file being transferred for a communication channel (direct serial connection or modem connection).

```
XferGetCurrFileName(short ChannelID, BSTR *FileName, long *Size, VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

FileName

A pointer to a buffer that receives a null-terminated character string containing the file name.

Size

A pointer to the variable that receives the size, in characters, of the file name string.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

2.3.6 XferGetCurrFileSize Method

Retrieves the size of file being transferred for a communication channel (direct serial connection or modem connection).

```
XferGetCurrFileSize(short ChannelID, long *Size);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

Size

A pointer to the variable that receives the size of file being transferred.

2.3.7 XferSetDstFile Method

Informs MegaPipe of the name of file which will be created on the download side for all XModem protocols for a communication channel (direct serial connection or modem connection).

```
XferSetDstFile(short ChannelID, BSTR FileName, VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

FileName

A pointer to a null-terminated string containing the relevant path name of file which will be created on the download side for all XModem protocols.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

Remarks

YModem, YModem-G, ZModem or Kermit doesn't need to touch this API, since the name of file on the download side is identical to the name of file on the upload side.

2.3.8 XferSetParam Method

Sets up file-transfer direction (upload or download) and file-transfer protocol type for a communication channel (direct serial connection or modem connection).

```
XferSetParam(short ChannelID, short Mode, short Protocol, VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

Mode

This parameters indicates file-transfer direction and it can be one of the following values.

Value	Comment
1	Upload file
2	Download file

Protocol

This parameters indicates file-transfer protocol type and it can be one of the following values.

Value	Comment
0	XModem Checksum protocol
1	XModem CRC protocol
2	XModem 1K protocol
3	YModem protocol
4	YModem-G protocol
5	ZModem protocol
6	Kermit protocol

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

2.3.9 XferSetWorkDir Method

Sets up the work directory for a communication channel (direct serial connection or modem connection).

```
XferSetWorkDir(short Index, BSTR WorkDir, VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

WorkDir

A Pointer to a null-terminated string containing the work directory for a file-upload or a file-download session.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

2.3.10 XferStart Method

Starts a file-transfer session for a communication channel (direct serial connection or modem connection).

```
XferStart(short ChannelID, VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

2.3.11 XferStop Method

Stops a file-transfer session for a communication channel (direct serial connection or modem connection).

```
XferStop(short ChannelID, VARIANT_BOOL *Result);
```

Parameters

ChannelID

A channel ID (0-7) used to identify a communication channel.

Result

If the method succeeds, the value of variable pointed to by Result is VARIANT_TRUE, otherwise the value is VARIANT_FALSE.

3 License

License agreement

This License Agreement ("LA") is the legal agreement between you and MW6 Technologies, Inc. ("MW6") for the software, the font, and any electronic documentation ("Package"). By using, copying or installing the Package, you agree to be bound by the terms of this LA. If you don't agree to the terms in this LA, immediately remove unused Package.

1. License

* The Single Developer License allows 1 developer in your organization the royalty-free distribution (up to 10,000 users) of the software to the third parties, **each individual developer requires a separate Single Developer License as long as he or she needs access to MW6's product(s) and document(s)**.

* The 2 Developer License allows 2 developers in your organization the royalty-free distribution (up to 10,000 users) of the software to the third parties.

* The 3 Developer License allows 3 developers in your organization the royalty-free distribution (up to 10,000 users) of the software to the third parties.

* The 4 Developer License allows 4 developers in your organization the royalty-free distribution (up to 10,000 users) of the software to the third parties.

* The 5 Developer License allows 5 developers in your organization the royalty-free distribution (up to 10,000 users) of the software to the third parties.

* The Unlimited Developer License allows unlimited number of developers in your organization the royalty-free distribution (unlimited number of users) of the software to the third parties.

2. User Disclaimer

The software is provided "as is" without warrant of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or noninfringement. MW6 assumes no liability for damages, direct or consequential, which may result from the use of the software. Further, MW6 assumes no liability for losses caused by misuse or abuse of the software. This responsibility rests solely with the end user.

3. Copyright

The software and any electronic documentation are the proprietary products of MW6 and are protected by copyright and other intellectual property laws.