DAPtools OCX Manual

DAPtools OCX property,
method, and event reference.

Version 1.01

Microstar Laboratories, Inc.

Microstar Laboratories, Inc.
2265 116 Avenue N.E.
Bellevue, WA 98004
Tel: (425) 453-2345
Fax: (425) 453-3199
http:// www.mstarlabs.com

Part Number MSDTOCXM101-9710 1

Contents

1. Introduction

The DAPtools OCX includes the DAP OCX, examples, and documentation for using a Data Acquisition Processor with environments that support OCXs such as Visual Basic and Delphi.

The DAP OCX is compatible with Windows 95 and Windows NT. The DAP OCX requires the DAPIO32 DLL which is shipped with DAPIO32 for Windows, Accel32, and DAPcell.

The DAP OCX translates commands between the programming environment and the Data Acquisition Processor. It also automates several parts of Data Acquisition Processor communication.

The DAP OCX:

- Opens and closes Data Acquisition Processor communication pipes
- Sends custom commands and DAPL command files on startup
- Flushes existing data on startup and shutdown
- Polls the Data Acquisition Processor for data

2. Installation

Requirements

- Windows 95 with Windows DAP Driver and DAPIO32 for Windows, or Accel32, or DAPcell.

OR

- Windows NT with Accel32 or DAPcell.

Steps

1. Insert Disk 1 into the floppy disk drive.
2. Run `SETUP.EXE` and follow the instructions of the installation program.

The install program installs the DAPtools OCX files and program examples and updates the registry.

Supported environments

- Visual Basic 4.0, Visual Basic 5.0
- Delphi 2.0, Delphi 3.0

More Information

Before using the DAPtools OCX, please read the `README.TXT` file that is in the main directory of your DAPtools OCX installation. The `README.TXT` file contains important information about the DAPtools OCX.

3. DAP OCX Interface Reference

The following pages contain a complete alphabetical listing of all DAP OCX properties, methods, and events.

DAP Control

DAP is the underlying control of the Data Acquisition Processor board. **DAP** supports the properties, methods, and events discussed in the following pages.

Properties

Accel32Version
AutomaticBinaryDataRead
AutomaticTextDataRead
CCFile
CCStackSize
CharData
CtlVersion
DAPLFile
DapName
FloatData
FlushBinaryInput
FlushOnShutdown
FlushOnStartup
FlushTextInput
InputAvail
IntData
LongData
MinBytesToRead
MinPollingInterval
OutputSpace
StringData

Methods

Int16BufferGet
Int16BufferPut

Events

NewBinaryData
NewTextData

Topics

Accel32Version Property

Get the Accel32 version.

Applies To
DAP

Declaration
*object.*Accel32Version
Type: Integer

Default
N/A

Access Restrictions
Read Only

Remarks
Accel32Version must be divided by 100 to get the Accel32 version. For example, reading 100 would mean that the current Accel32 version is 1.00.

Accel32Version will return 0 as the version number if Accel32 is not installed. This will occur using the Windows DAP Driver under Windows 95.

Example
```
Private Sub Command1_Click()
  'Get Accel32 version and display as
  'caption of label
  Label1.Caption = "Accel32 version: " + _
    Str$(DAP1.Accel32Version / 100)
End Sub
```

See Also
CtlVersion Property

AutomaticBinaryDataRead Property

Specify whether the DAP OCX automatically polls the Data Acquisition Processor for binary data.

Applies To

DAP

Declaration

*object.*AutomaticBinaryDataRead [= Boolean]

Type: Boolean

Default

True

Access Restrictions

None

Remarks

If **AutomaticBinaryDataRead** is true, the **NewBinaryData** event is fired when **MinBytesToRead** bytes are available from the Data Acquisition Processor binary output communication pipe, `$BinOut`. If it is false, **NewBinaryData** is not fired and the application must use another mechanism to poll data.

See Also

AutomaticTextDataRead Property, **MinBytesToRead** Property, **NewBinaryData** Event

AutomaticTextDataRead Property

Specify whether the DAP OCX automatically polls the Data Acquisition Processor for text data.

Applies To
DAP

Declaration
object.AutomaticTextDataRead [= Boolean]
Type: Boolean

Default
True

Access Restrictions
None

Remarks
If **AutomaticTextDataRead** is true, the **NewTextData** event is fired for each line of text from the Data Acquisition Processor text output communication pipe, `$SysOut`. If it is false, **NewTextData** is not fired and the application must use another mechanism to poll data.

See Also
AutomaticBinaryDataRead Property, **NewTextData** Event

CCFile Property

Specify a single custom command binary to download to the Data Acquisition Processor.

Applies To

DAP

Declaration

object.CCFile [= String]

Type: String

Default

Empty string

Access Restrictions

None

Remarks

CCFile must contain the full path and filename of the custom command binary, usually with .BIN as the extension of the filename. It downloads a single custom command binary to the Data Acquisition Processor, with a stack size specified by the **CCStackSize** property. **CCFile** sends a RESET command to the Data Acquisition Processor, and therefore must be used before the **DAPLFile** property.

If **CCFile** is set at design-time, the custom command is downloaded automatically on startup after the Data Acquisition Processor is initialized but before the DAPL command file is downloaded. If it is set at run-time, the custom command is downloaded immediately. If **CCFile** encounters a problem, the DAP OCX will raise the DAPIO32 DLL error:

```
'Could not download custom command file, CCFile.'
```

CCFile does not support downloading custom command lists (.TXT or .LST extension). If there are multiple custom commands to download to the Data Acquisition Processor, it should be done at run-time by calling the **CCFile** property multiple times.

The filename of the custom command can be any length, but only the first 11 characters are significant in DAPL. This means that the filename must be a valid DAPL symbol name. A valid DAPL symbol name must begin with a letter, which can be followed by letters, numbers, and underscore characters. DAPL is case insensitive. For more information on this topic, please refer to the Command Syntax chapter of the DAPL Manual.

See Also

`CCStackSize` Property, **`DAPLFile`** Property

CCStackSize Property

Specify the stack size in bytes for the **CCFile** property.

Applies To
DAP

Declaration
object.CCStackSize [= Integer]
Type: Integer

Default
1000

Access Restrictions
None

Remarks
CCStackSize is the stack size in bytes for a custom command specified by the **CCFile** property.

The default stack size of 1000 bytes is sufficient for most custom commands. **CCStackSize** must be in the range 1000 to 32764.

See Also
CCFile Property

CharData Property

Read a character value from the Data Acquisition Processor. Write a character value to the Data Acquisition Processor.

Applies To

DAP

Declaration

object.CharData [= Integer]

Type: Integer

Default

N/A

Access Restrictions

Run-time only

Remarks

On read of **CharData**, a character value is read from the Data Acquisition Processor text output communication pipe, `$SysOut`. There is a 10 second time-out if there is no character value to read. When this happens, the DAP OCX will raise the error:

```
'Unable to read CharData from DAP.'
```

On write of **CharData**, the assigned character value is sent to the Data Acquisition Processor text input communication pipe, `$SysIn`. There is a 10 second time-out if there is no space to write the character value. When this happens, the DAP OCX will raise the error:

```
'Unable to write CharData to DAP.'
```

Example

```
Private Sub Command1_Click()
  'Read character value from DAP and display as caption
  'of label.
  Label1.Caption = "Character Data: " + Chr$(DAP1.CharData)
End Sub
```

See Also

FloatData Property, **IntData** Property, **LongData** Property, **StringData** Property

CtlVersion Property

Get the DAP Control version.

Applies To
DAP

Declaration
object.CtlVersion
Type: Integer

Default
N/A

Access Restrictions
Read only

Remarks
CtlVersion must be divided by 100 to get the DAP Control version. For example, reading 100 would mean that the current DAP Control version is 1.00.

Example
```
Private Sub Command1_Click()
  'Get DAP Control version and display as caption of label.
  Label1.Caption = "Control version: " + _
   Str$(DAP1.CtlVersion / 100)
End Sub
```

See Also
Accel32Version Property

DAPLFile Property

Specify the DAPL command file to configure the Data Acquisition Processor.

Applies To
DAP

Declaration
object.DAPLFile [= String]
Type: String

Default
Empty string

Access Restrictions
None

Remarks
DAPLFile must contain the full path and filename of the DAPL command file, usually with .DAP as the extension of the filename. **DAPLFile** must be used after **CCFile** because **CCFile** sends a RESET command to the Data Acquisition Processor.

If **DAPLFile** is set at design-time, the DAPL command file is sent automatically on startup after the Data Acquisition Processor is initialized and the custom command is downloaded. If it is set at run-time, the DAPL command file is sent immediately. If **DAPLFile** encounters a problem, the DAP OCX will raise the DAPIO32 DLL error:

```
'Could not download DAPL command file, DAPLFile.'
```

See Also
CCFile Property

DapName Property

Specify the machine name and Data Acquisition Processor name for text and binary communication with the Data Acquisition Processor.

Applies To

DAP

Declaration

object.DapName [= String]

Type: String

Default

`\\.\Dap0`

Access Restrictions

None

Remarks

DapName consists of two portions, a machine name and Data Acquisition Processor name. It is led by two backslashes, with the second portion delimited by one backslash. The naming method is based on the Universal Naming Convention (UNC).

Using the default as an example, `\\.` denotes the local machine and `\Dap0` is the name of the Data Acquisition Processor. If you are connected to another machine named PC45, that contains `Dap0`, then the **DapName** is `\\PC45\Dap0`.

DapName opens handles to four communication pipes on the Data Acquisition Processor: text input communication pipe `$SysIn`, text output communication pipe `$SysOut`, binary input communication pipe `$BinIn`, and binary output communication pipe `$BinOut`. There is no access to each of the handles.

If **DapName** is set at design-time, all the communication pipes are opened when the application is run. If it is set at run-time, the previous communication pipes are closed, and the new **DapName** communication pipes are opened. If **FlushOnShutdown** is true, the previous communication pipes are flushed and then closed. If **FlushOnStartup** is true, the new **DapName** communication pipes are

opened and then flushed. If **DapName** encounters a problem, the DAP OCX will raise the DAPIO32 DLL error:

```
'Could not access DAP Driver.'
```

Usually **DapName** is set at design- time. If your application needs access to more than one Data Acquisition Processor, use multiple instances of the DAP OCX. Only in advanced applications should **DapName** be set at run-time.

Example

```
Private Sub Command1_Click()
  'Open communication pipes on the local machine
  ' to the third DAP, named Dap2.
  DAP3.DapName = "\\.\Dap2"
End Sub
```

FloatData Property

Read a floating point value from the Data Acquisition Processor. Write a floating point value to the Data Acquisition Processor.

Applies To
DAP

Declaration
object.FloatData [= Single]
Type: Single

Default
N/A

Access Restrictions
Run-time only

Remarks
On read of **FloatData**, a floating point value is read from the Data Acquisition Processor binary output communication pipe, `$BinOut`. There is a 10 second time-out if there is no floating point value to read. When this happens, the DAP OCX will raise the error:

```
'Unable to read FloatData from DAP.'
```

On write of **FloatData**, the assigned floating point value is sent to the Data Acquisition Processor binary input communication pipe, `$BinIn`. There is a 10 second time-out if there is no space to write the floating point value. When this happens, the DAP OCX will raise the error:

```
'Unable to write FloatData to DAP.'
```

DAPL has no built-in commands which use floating point data. **FloatData** is provided for use with custom commands.

Example

```
Private Sub Command1_Click()
  'Read floating point value from DAP and display as
  'caption of label.
  Label1.Caption = "Float Data: " + Str$(DAP1.FloatData)
End Sub
```

See Also

CharData Property, **IntData** Property, **LongData** Property, **StringData** Property

FlushBinaryInput Property

Flush existing binary data from the Data Acquisition Processor.

Applies To
DAP

Declaration
object.FlushBinaryInput
Type: Boolean

Default
N/A

Access Restrictions
Run-time only; Read only

Remarks
FlushBinaryInput is true if all old binary data were flushed from the Data Acquisition Processor binary output communication pipe, `$BinOut`. It is false if all old binary data were not flushed.

Example
```
Private Sub Command1_Click()
  'Flush binary data from DAP and check if successful
  'with message box.
  If DAP1.FlushBinaryInput Then
    MsgBox "Binary data were successfully flushed"
  Else
    MsgBox "Binary data were not flushed"
  End If
End Sub
```

See Also
FlushOnShutdown Property, **FlushOnStartup** Property, **FlushTextInput** Property

FlushOnShutdown Property

Specify whether existing Data Acquisition Processor data values are automatically flushed on shutdown.

Applies To

DAP

Declaration

object.FlushOnShutdown [= Boolean]

Type: Boolean

Default

True

Access Restrictions

None

Remarks

If **FlushOnShutdown** is true, a RESET command is sent to the Data Acquisition Processor on shutdown, and existing data values are flushed.

See Also

FlushBinaryInput Property, **FlushOnStartup** Property, **FlushTextInput** Property

FlushOnStartup Property

Specify whether existing Data Acquisition Processor data values are automatically flushed on startup.

Applies To
DAP

Declaration
object.FlushOnStartup [= Boolean]
Type: Boolean

Default
True

Access Restrictions
None

Remarks
If **FlushOnStartup** is true, a RESET command is sent to the Data Acquisition Processor on startup, and existing data values are flushed.

See Also
FlushBinaryInput Property, **FlushOnShutdown** Property, **FlushTextInput** Property

FlushTextInput Property

Flush existing text data from the Data Acquisition Processor.

Applies To
DAP

Declaration
object.FlushTextInput
Type: Boolean

Default
N/A

Access Restrictions
Run-time only; Read only

Remarks
FlushTextInput is true if all old text data were flushed from the Data Acquisition Processor text output communication pipe, `$SysOut`. It is false if all old text data were not flushed.

Example
```
Private Sub Command1_Click()
  'Flush text data from DAP and check if successful
  'with message box.
  If DAP1.FlushTextInput Then
    MsgBox "Text data were successfully flushed"
  Else
    MsgBox "Text data were not flushed"
  End If
End Sub
```

See Also
FlushBinaryInput Property, **FlushOnShutdown** Property, **FlushOnStartup** Property

InputAvail Property

Get the number of bytes available for reading from the Data Acquisition Processor.

Applies To
DAP

Declaration
object.InputAvail
Type: Integer

Default
N/A

Access Restrictions
Run-time only; Read only

Remarks
InputAvail returns the number of bytes available for reading from the Data Acquisition Processor binary output communication pipe, `$BinOut`.

If **InputAvail** encounters a problem, the DAP OCX will raise the DAPIO32 DLL error:

```
'Could not get the number of bytes for reading.'
```

Example
```
Private Sub Command1_Click()
  'Get number of bytes available for reading from DAP
  'and display as caption of label.
  Label1.Caption = "Number of bytes for reading: " + _
   Str$(DAP1.InputAvail)
End Sub
```

See Also
Int16BufferGet Method, **OutputSpace** Property

IntData Property

Read an integer value from the Data Acquisition Processor. Write an integer value to the Data Acquisition Processor.

Applies To

DAP

Declaration

object.IntData [= Integer]

Type: Integer

Default

N/A

Access Restrictions

Run-time only

Remarks

On read of **IntData**, an integer value is read from the Data Acquisition Processor binary output communication pipe, `$BinOut`. There is a 10 second time-out if there is no integer value to read. When this happens, the DAP OCX will raise the error:

```
'Unable to read IntData from DAP.'
```

On write of **IntData**, the assigned integer value is sent to the Data Acquisition Processor binary input communication pipe, `$BinIn`. There is a 10 second time-out if there is no space to write the integer value. When this happens, the DAP OCX will raise the error:

```
'Unable to write IntData to DAP.'
```

Example

```
Private Sub Command1_Click()
  'Read integer value from DAP and display as caption
  'of label.
  Label1.Caption = "Integer Data: " + Str$(DAP1.IntData)
End Sub
```

See Also

CharData Property, **FloatData** Property, **LongData** Property, **StringData** Property

LongData Property

Read a long value from the Data Acquisition Processor. Write a long value to the Data Acquisition Processor.

Applies To

DAP

Declaration

object.LongData [= Long]

Type: Long

Default

N/A

Access Restrictions

Run-time only

Remarks

On read of **LongData**, a long value is read from the Data Acquisition Processor binary output communication pipe, `$BinOut`. There is a 10 second time-out if there is no long value to read. When this happens, the DAP OCX will raise the error:

```
'Unable to read LongData from DAP.'
```

On write of **LongData**, the assigned long value is sent to the Data Acquisition Processor binary input communication pipe, `$BinIn`. There is a 10 second time-out if there is no space to write the long value. When this happens, the DAP OCX will raise the error:

```
'Unable to write LongData to DAP.'
```

Example

```
Private Sub Command1_Click()
  'Read long value from DAP and display as caption
  'of label.
  Label1.Caption = "Long Data: " + Str$(DAP1.LongData)
End Sub
```

See Also

CharData Property, **FloatData** Property, **IntData** Property, **StringData** Property

MinBytesToRead Property

Specify the minimum number of bytes that must be available from the Data Acquisition Processor before firing the **NewBinaryData** event.

Applies To

DAP

Declaration

object.MinBytesToRead [= Integer]

Type: Integer

Default

Two

Access Restrictions

None

Remarks

MinBytesToRead is used only if **AutomaticBinaryDataRead** is true. The **NewBinaryData** event is fired when **MinBytesToRead** bytes are available from the Data Acquisition Processor binary output communication pipe, `$BinOut`.

MinBytesToRead must be a multiple of two in the range 2 to 32766. In most cases, it should be set to two bytes and the **NewBinaryData** event handler should read all the bytes specified by the **NewBinaryData** *BytesAvailable* parameter. If you plan to use another value, refer to NewBinaryData Restrictions.

See Also

AutomaticBinaryDataRead Property, **MinPollingInterval** Property, **NewBinaryData** Event, NewBinaryData Restrictions

MinPollingInterval Property

Specify the minimum interval in milliseconds between checks for available data from the Data Acquisition Processor.

Applies To
DAP

Declaration
object.MinPollingInterval [= Integer]
Type: Integer

Default
Zero

Access Restrictions
None

Remarks
MinPollingInterval is used only if either **AutomaticBinaryDataRead** or **AutomaticTextDataRead** is true. It applies to the polling interval for the **NewBinaryData** and **NewTextData** events.

If **MinPollingInterval** is set to zero, polling occurs every one millisecond.

See Also
AutomaticBinaryDataRead Property, **AutomaticTextDataRead** Property, **MinBytesToRead** Property

OutputSpace Property

Get the number of bytes that can be buffered for writing to the Data Acquisition Processor.

Applies To

DAP

Declaration

object.OutputSpace

Type: Integer

Default

N/A

Access Restrictions

Run-time only; Read only

Remarks

OutputSpace returns the number of bytes that can be buffered for writing to the Data Acquisition Processor binary input communication pipe, `$BinIn`.

If you write more bytes than that returned by **OutputSpace**, the write operation does not complete until the Data Acquisition Processor has read sufficient data.

If **OutputSpace** encounters a problem, the DAP OCX will raise the DAPIO32 DLL error:

```
'Could not get the number of bytes for writing.'
```

Example

```
Private Sub Command1_Click()
  'Get number of bytes that can be buffered for writing
  'to DAP and display as caption of label.
  Label1.Caption = "Number of bytes for writing: " + _
    Str$(DAP1.OutputSpace)
End Sub
```

See Also

InputAvail Property, **Int16BufferPut** Method

StringData Property

Read a string from the Data Acquisition Processor. Write a string to the Data Acquisition Processor.

Applies To

DAP

Declaration

object.StringData [= String]

Type: String

Default

N/A

Access Restrictions

Run-time only

Remarks

On read of **StringData**, characters are read from the Data Acquisition Processor text output communication pipe, `$SysOut` until a carriage-return is reached. The carriage-return is removed before the string is returned. If a line-feed followed the carriage-return, it is also removed. There is a 10 second time-out if there is no string to read. When this happens, the DAP OCX will raise the error:

```
'Unable to read StringData from DAP.'
```

On write of **StringData**, the assigned string followed by a carriage-return, is sent to the Data Acquisition Processor text input communication pipe, `$SysIn`. There is a 10 second time-out if there is no space to write the string. When this happens, the DAP OCX will raise the error:

```
'Unable to write StringData to DAP.'
```

Example

```
Private Sub Command1_Click()
  'Read string from DAP and display as caption of label.
  Label1.Caption = "String Data: " + DAP1.StringData
End Sub
```

See Also

CharData Property, **FloatData** Property, **IntData** Property, **LongData** Property

Int16BufferGet Method

Read a buffer of data from the Data Acquisition Processor into an array.

Applies To

DAP

Declaration

Visual Basic

object.Int16BufferGet(*Length* As Long, *Buffer* As Integer) As Long

Delphi

function *object*.Int16BufferGet(*Length*: Integer; var *Buffer*: Smallint): Integer;

Parameters

object

Evaluates to the object in the 'Applies To' section.

Length

The number of 16-bit integers requested from the Data Acquisition Processor. *Length* is a 32-bit integer.

Buffer

A reference to one element in an array of 16-bit integers.

Return Values

The number of 16-bit elements successfully read from the Data Acquisition Processor. The return value is a 32-bit integer.

Remarks

Int16BufferGet reads a buffer of data from the Data Acquisition Processor binary output communication pipe, `$BinOut` into an array.

Length is the number of 16-bit integers requested from the Data Acquisition Processor. *Buffer* is a reference to one element in an array of 16-bit integers. It is important that you DO NOT read more elements than you have allocated in the array.

Int16BufferGet returns the number of 16-bit elements successfully read from the Data Acquisition Processor. There is a 10 second time-out if it cannot read all the requested *Length* elements. When this happens, the DAP OCX will raise the error:

```
'Unable to read BufferData from DAP.'
```

It might be helpful to check **InputAvail** before reading *Length* elements in **Int16BufferGet**. Note that **InputAvail** returns the number of bytes and not the number of 16-bit elements.

Example

```
Private Sub Command1_Click()
  'Declare 100 elements in BufData array.
  Dim BufData(100) As Integer
  Dim Elements As Long
  Dim ElementsGet As Long
  Elements = 100  'Specify 100 elements to read.
  'Read 100 elements of data from DAP into BufData array.
  ElementsGet = DAP1.Int16BufferGet(Elements, BufData(1))
End Sub
```

See Also

InputAvail Property, **Int16BufferPut** Method, **NewBinaryData** Event

Int16BufferPut Method

Write a buffer of data from an array to the Data Acquisition Processor.

Applies To

DAP

Declaration

Visual Basic

object.Int16BufferPut(*Length* As Long, *Buffer* As Integer) As Long

Delphi

function *object*.Int16BufferPut(*Length*: Integer; var *Buffer*: Smallint): Integer;

Parameters

object

Evaluates to the object in the 'Applies To' section.

Length

The number of 16-bit integers to write to the Data Acquisition Processor. *Length* is a 32-bit integer.

Buffer

A reference to one element in an array of 16-bit integers.

Return Values

The number of 16-bit elements successfully written to the Data Acquisition Processor. The return value is a 32-bit integer.

Remarks

Int16BufferPut writes a buffer of data from an array to the Data Acquisition Processor binary input communication pipe, $BinIn.

Length is the number of 16-bit integers to write to the Data Acquisition Processor. *Buffer* is a reference to one element in an array of 16-bit integers. It is important that you DO NOT write more elements than you have allocated in the array.

Int16BufferPut returns the number of elements successfully written to the Data Acquisition Processor. There is a 10 second time-out if it cannot write all *Length* elements. When this happens, the DAP OCX will raise an error:

```
'Unable to write BufferData to DAP.'
```

It might be helpful to check **OutputSpace** before writing *Length* elements in **Int16BufferPut**. Note that **OutputSpace** returns the number of bytes and not the number of 16-bit elements.

Example

```
Private Sub Command1_Click()
  'Declare 100 elements in BufData array.
  Dim BufData(100) As Integer
  Dim Elements As Long
  Dim ElementsPut As Long
  Elements = 100   'Specify 100 elements to write.
  For I = 1 To 100 'Initialize BufData array.
    BufData(I) = I * 5
  Next I
  'Write 100 elements of data from BufData array to DAP.
  ElementsPut = DAP1.Int16BufferPut(Elements, BufData(1))
End Sub
```

See Also

Int16BufferGet Method

NewBinaryData Event

Respond to new binary data from the Data Acquisition Processor.

Applies To

DAP

Declaration

Private Sub *object*_NewBinaryData(*BytesAvailable:*As Integer)

Parameters

object

Evaluates to the object in the 'Applies To' section.

BytesAvailable

The number of bytes available, which is greater than or equal to **MinBytesToRead**.

Remarks

NewBinaryData runs only if **AutomaticBinaryDataRead** is true. It is fired at most every **MinPollingInterval** milliseconds when **MinBytesToRead** bytes are available from the Data Acquisition Processor binary output communication pipe, `$BinOut`.

The *BytesAvailable* parameter is the number of bytes available, which is greater than or equal to **MinBytesToRead**. The DAP OCX does not read data from the Data Acquisition Processor when **NewBinaryData** is fired. Reading data is the responsibility of the event handler.

Note that the DAP OCX behaves differently with the **NewTextData** event.

Example

The **NewBinaryData** event handler uses **Int16BufferGet** to read a buffer of data from the Data Acquisition Processor into the `BufData` array. The `Min` function returns the smaller of `BufData` size in elements and *BytesAvailable* divided by 2.

```
Private Function Min(m As Integer, n As Integer) As Integer
  If m < n Then
    Min = m
  Else
    Min = n
  End If
End Function

Private Sub DAP1_NewBinaryData(BytesAvailable As Integer)
  'Declare 100 elements in BufData array.
  Dim BufData(100) As Integer
  Dim Elements As Long
  Dim ElementsGet As Long

  'The Min() functions returns the smaller of BufData
  'size in elements and BytesAvailable/2.
  Elements = Min(UBound(BufData), BytesAvailable/2)

  'Read Elements from DAP into BufData array.
  ElementsGet = DAP1.Int16BufferGet(Elements, BufData(1))
End Sub
```

See Also

AutomaticBinaryDataRead Property, **MinBytesToRead** Property, **MinPollingInterval** Property, NewBinaryData Restrictions, **NewTextData** Event

NewBinaryData Event Restrictions

The PC side buffer associated with the Data Acquisition Processor binary output communication pipe `$BinOut`, referred here as `maxsize`, must be at least (1020 + **`MinBytesToRead`**) bytes for a-series and e-series Data Acquisition Processors. Otherwise, the **`NewBinaryData`** event may never fire even though the Data Acquisition Processor has data to send.

The number 1020 is Data Acquisition Processor series and device driver dependent. This number may change for future Data Acquisition Processor products. The default setting for `maxsize` is 2048 bytes for the Windows DAP Driver under Windows 95, and 4096 bytes for Accel32 under Windows NT.

See Also

`MinBytesToRead` Property, **`NewBinaryData`** Event

NewTextData Event

Respond to new text data from the Data Acquisition Processor.

Applies To

DAP

Declaration

Private Sub *object*_NewTextData(*DAPString:* As String)

Parameters

object

Evaluates to the object in the 'Applies To' section.

DAPString

The complete string from the Data Acquisition Processor.

Remarks

NewTextData runs only if **AutomaticTextDataRead** is true. It is fired when a complete string is read from the Data Acquisition Processor text output communication pipe, `$SysOut`. A complete string could be spaces or control characters but not an empty string. Empty strings are consumed by the DAP OCX.

The *DAPString* parameter is the complete string from the Data Acquisition Processor. If *DAPString* is not processed by the event handler, the string is lost.

NewTextData is typically used to display error and status messages from the Data Acquisition Processor. To enable error messages, refer to the DAPL command `OPTIONS ERRORQ`.

Note that the DAP OCX behaves differently with the **NewBinaryData** event.

Example

```
Private Sub DAP1_NewTextData(DAPString As String)
  'Display string from DAP, if exists, in a message box.
  MsgBox DAPString, , "String from DAP"
End Sub
```

See Also

AutomaticTextDataRead Property, **MinPollingInterval** Property, **NewBinaryData** Event

4. Updating DAP VBX Project

You can update projects that use the 16-bit DAP VBX to use the 32-bit DAP OCX. This process will be automatic if you follow the steps in Converting DAP VBX Project.

Some of the properties of the DAP VBX are now obsolete in the DAP OCX. Because of this, you may have to make some coding changes after the DAP VBX project has been converted to a DAP OCX project. Converting Obsolete Properties and Obsolete Properties Summary has information on this.

Note that some of the obsolete properties are still implemented for backward compatibility, while others raise errors. For properties that raise errors, use the error-handling features of Visual Basic and Delphi to handle them. Refer to Handling Errors and Error Messages.

- Converting DAP VBX Project
- Converting Obsolete Properties
- Obsolete Properties Summary

See Also

Handling Errors, Error Messages

Converting DAP VBX Project

The steps to converting a DAP VBX project are listed below.

- Update `VB.INI` to include the below lines where `<InstallDir>` is the install directory for the DAP OCX. This should only be done once.

```
[VBX Conversions32]
dap.vbx={53263BAC-4A99-11CF-B0B4-444553540000}#1.0#0;
<InstallDir>\dap.ocx
```

- Save the original DAP VBX form (`.FRM`) in text format using 16-bit Visual Basic. The conversion cannot take place if the form was saved in binary format.
- Open the original DAP VBX project (`.MAK`) under 32-bit Visual Basic. Respond 'Yes' when asked to upgrade the DAP VBX custom control to its newer version.

See Also

Updating DAP VBX Project, Converting Obsolete Properties, Obsolete Properties Summary

Converting Obsolete Properties

The Obsolete Properties Summary section provides a list of replacements for the obsolete properties. If you were using any obsolete properties, use the replacement properties after your DAP VBX project has been converted to a DAP OCX project.

The following topics describe how to handle the special cases of obsolete properties.

Numbered ACCEL Devices

If in the original DAP VBX project, **TextACCELNumber** was set to 0 and **BinaryACCELNumber** was set to 1, **DapName** is set to `\\.\Dap0` in the converted DAP OCX project.

If **TextACCELNumber** or **BinaryACCELNumber** were set to different values, you will see the message below during conversion.

```
Could not convert TextACCELNumber and BinaryACCELNumber.
Please set the DapName property after the project is loaded.
```

If this occurs, **DapName** is set to `\\.\DapX`, and you must update it to the appropriate **DapName** before running your application.

Custom Command Lists

CCFile does not support downloading custom command lists. You must update any code that uses a list of custom commands (`.TXT` or `.LST` extension) to a single custom command. There is no longer a default extension with the **CCFile** property, so you have to include the extension. The extension is usually `.BIN`.

If you have several custom commands to download to the Data Acquisition Processor, use the **CCFile** property at run-time. You can call this property several times to download several custom commands. If your custom commands have different stack sizes, you must set **CCStackSize** for each custom command.

DAPL Command Files

There is no longer a default extension with the **DAPLFile** property, so you have to include the extension. The extension is usually `.DAP`.

See Also

Updating DAP VBX Project, Converting DAP VBX Project,
Obsolete Properties Summary

Obsolete Properties Summary

The list below summarizes all the obsolete properties and their replacement properties. The behavior category summarizes whether the obsolete property is still implemented or raises an error.

If the obsolete property raises an error, you must either handle the error using the error-handling features of Visual Basic and Delphi, or eliminate the use of those properties. Refer to Handling Errors and Error Messages.

Obsolete Property	Replacement Property	Behavior
`ACCELVersion`	`Accel32Version`	Still implemented
`AutomaticDataRead`	`AutomaticBinaryDataRead` `AutomaticTextDataRead`	Still implemented
`BinaryACCELNumber` `TextACCELNumber`	`DapName`	Raises an error
`BinaryHandle` `TextHandle`	None	Raises an error
`FlushInputs`	`FlushBinaryInput` `FlushTextInput`	Still implemented
`GetAvail`	`InputAvail`	Still implemented
`IoCtlString`	None	Raises an error
`LoadError`	None	Still implemented
`PutAvail`	`OutputSpace`	Still implemented
`TextHandle`	None	Raises an error

See Also

Updating DAP VBX Project, Converting DAP VBX Project, Converting Obsolete Properties

5. Handling Errors

You can use the error-handling features of Visual Basic to trap errors and take corrective action. When an error occurs, Visual Basic sets the various properties of the Error object, `Err`, such as an error description (`Err.Description`), error number (`Err.Number`), etc.

Use the `Err` object and its properties in an error-handling routine to handle the errors raised by the DAP OCX.

Example

The example below shows one method of trapping errors. In the **`NewBinaryData`** event handler, the **`Int16BufferGet`** method will raise an error after 10 seconds if it cannot read all the requested *`Length`* elements from the DAP.

If you do not want **`Int16BufferGet`** to raise an error, refer to the **`NewBinaryData`** event example.

```
Private Sub DAP1_NewBinaryData(BytesAvailable As Integer)

On Error GoTo TrapError   'Turn on error trapping.
  'Declare 100 elements in BufData array.
  Dim BufData(100) As Integer
  Dim Elements As Long
  Dim ElementsGet As Long
  Elements = 100   'Specify 100 elements to read.

  'Read 100 elements of data from DAP into BufData array.
  ElementsGet = DAP1.Int16BufferGet(Elements, BufData(1))
'Do not execute error handler if no error occurs.
Exit Sub

TrapError:   'Branch here if error occurs.
  'Display error description in a message box.
  MsgBox (Err.Description)

End Sub
```

Error Messages

The following is a list of error messages reported by the DAP OCX. Some of the errors are generated by the DAP OCX while others are generated by the DAPIO32 DLL.

DAP OCX Errors

Unable to read BufferData from DAP.
Unable to write BufferData to DAP.
Unable to read **`CharData`** from DAP.
Unable to write **`CharData`** to DAP.
Unable to read **`FloatData`** from DAP.
Unable to write **`FloatData`** to DAP.
Unable to read **`IntData`** from DAP.
Unable to write **`IntData`** to DAP.
Unable to read **`LongData`** from DAP.
Unable to write **`LongData`** to DAP.
Unable to read **`StringData`** from DAP.
Unable to write **`StringData`** to DAP.
Unable to reserve a system timer.

DAP OCX Obsolete Errors

`BinaryACCELNumber` is obsolete and not supported by the DAP OCX.
`BinaryHandle` is obsolete and not supported by the DAP OCX.
`IoCtlString` is obsolete and not supported by the DAP OCX.
`TextACCELNumber` is obsolete and not supported by the DAP OCX.
`TextHandle` is obsolete and not supported by the DAP OCX.

DAPIO32 DLL Errors

Could not access DAP Driver.
Could not download custom command file, **`CCFile`**.
Could not download DAPL command file, **`DAPLFile`**.
Could not get the number of bytes for reading.
Could not get the number of bytes for writing.

Unable to read BufferData from DAP

Description

- There was a problem reading a buffer of data from the Data Acquisition Processor using the **`Int16BufferGet`** method.

Possible sources

- The requested number of elements, specified by the **`Int16BufferGet`** *`Length`* parameter, are not available.

Unable to write BufferData to DAP

Description

- There was a problem writing a buffer of data to the Data Acquisition Processor using the **`Int16BufferPut`** method.

Possible sources

- The Data Acquisition Processor is not reading data due to:
  An incorrect DAPL command list
  Output procedure underflow
  Failure to start processing procedures

Unable to reserve a system timer

Description

- There was a problem installing a system timer for the **`AutomaticBinaryDataRead`** and **`AutomaticTextDataRead`** properties.

Possible sources

- Your PC system does not have the available resources to install a timer. To conserve system resources, close other applications that may be using timers.

Could not access DAP Driver

Description

- The program could not access the DAP Driver.

Possible sources

- The specified **DapName** is in use by another application. Terminate the application or use another **DapName**.
- The specified **DapName** does not exist. Check that the machine name and Data Acquisition Processor name are correct.
- The Windows DAP Driver for Windows 95 or the Accel32 DAP Driver for Windows NT is not installed.

Could not download custom command file, CCFile

Description

- The program could not download the custom command file specified by the **CCFile** property.

Possible sources

- The custom command file does not exist. Check the path and filename of the custom command file.
- The custom command filename is an invalid DAPL symbol name.
- Incorrect custom command binary format. This is caused by downloading a DAPL 4 custom command on a DAPL 2000 system or vice versa.

Could not download DAPL command file, DAPLFile

Description

- The program could not download the DAPL command file specified by the **DAPLFile** property.

Possible sources

- The DAPL command file does not exist. Check the path and filename of the DAPL command file.

6. Appendix A. Additional Information

The following pages contain a property overview and an alphabetical listing of all obsolete DAP OCX properties.

Property Overview

Many of the DAP OCX properties communicate with the Data Acquisition Processor when they are assigned values. For example, a string that is assigned to the **StringData** property is sent to the Data Acquisition Processor. The string is not stored in the property.

Some properties can only be read. When the **InputAvail** property is read, the DAP OCX gets the number of bytes available from the Data Acquisition Processor. A value cannot be assigned to the **InputAvail** property.

Other properties can be read and written, but may do something different in each case. The **IntData** property reads an integer from the Data Acquisition Processor when its value is read, and sends an integer to the Data Acquisition Processor when its value is set.

Properties can be set at either design-time in the Visual Basic properties list, or while the program is running using Visual Basic code. Some properties are available only at run-time and do not appear in the properties list. For example, the **DAPLFile** property can be specified before the program runs so it is listed in the property list, but the **IntData** property has no meaning until the program is running so it is not listed.

Some of the design-time properties have default values. Unless a specific access restriction is specified, a property can be both read and written.

Obsolete Properties

The properties listed below are obsolete. The obsolete properties are discussed in greater detail in the following pages.

ACCELVersion
AutomaticDataRead
BinaryACCELNumber
BinaryHandle
FlushInputs
GetAvail
IoCtlString
LoadError
PutAvail
TextACCELNumber
TextHandle

ACCELVersion Property

The **ACCELVersion** property is obsolete. It is supported for backward compatibility with the DAP VBX. Please use the **Accel32Version** property instead.

See Also

Accel32Version Property

AutomaticDataRead Property

The **AutomaticDataRead** property is obsolete. It is supported for backward compatibility with the DAP VBX. Please use the **AutomaticBinaryDataRead** and **AutomaticTextDataRead** properties instead.

See Also

AutomaticBinaryDataRead Property, **AutomaticTextDataRead** Property

BinaryACCELNumber Property

The **BinaryACCELNumber** property is obsolete. Using this property will raise the exception:

```
'BinaryACCELNumber is obsolete and not supported by the DAP
OCX.'
```

See Also

DapName Property

BinaryHandle Property

The **BinaryHandle** property is obsolete. Using this property will raise the exception:

```
'BinaryHandle is obsolete and not supported by the DAP OCX.'
```

See Also

Int16BufferGet Method, **Int16BufferPut** Method

FlushInputs Property

The **FlushInputs** property is obsolete. It is supported for backward compatibility with the DAP VBX. Please use the **FlushBinaryInput** and **FlushTextInput** properties instead.

See Also

FlushBinaryInput Property, **FlushTextInput** Property

GetAvail Property

The **GetAvail** property is obsolete. It is supported for backward compatibility with the DAP VBX. Please use the **InputAvail** property instead.

See Also

InputAvail Property, **OutputSpace** Property

IoCtlString Property

The **IoCtlString** property is obsolete. Using this property will raise the exception:

```
'IoCtlString is obsolete and not supported by the DAP OCX.'
```

LoadError Property

The **LoadError** property is obsolete. It is supported for backward compatibility with the DAP VBX. Please use the error-handling features of Visual Basic and Delphi to handle errors.

See Also

Handling Errors, Error Messages

PutAvail Property

The **PutAvail** property is obsolete. It is supported for backward compatibility with the DAP VBX. Please use the **OutputSpace** property instead.

See Also

InputAvail Property, **OutputSpace** Property

TextACCELNumber Property

The **TextACCELNumber** property is obsolete. Using this property will raise the exception:

```
'TextACCELNumber is obsolete and not supported by the DAP
OCX.'
```

See Also

DapName Property

TextHandle Property

The **TextHandle** property is obsolete. Using this property will raise the exception:

```
'TextHandle is obsolete and not supported by the DAP OCX.'
```

See Also

Int16BufferGet Method, **Int16BufferPut** Method

Index