TWAIN Support In ObjectPAL

By Paul Cronk

13-February-2002

Revision History

Date	Version	Author	Description
13-February-2001	1.4	Paul Cronk	Edits for Paradox 10 SP2
08-August-2001	1.3	Paul Cronk	3rd revision
01-July-2001	1.2	Paul Cronk	2nd revision
28-June-2001	1.1	Paul Cronk	1st revision
07-June-2001	1.0	Paul Cronk	Initial draft

Reviewers

Dennis Santoro
Steve Caple
Oak Hall
Tony McGuire
Liz Woodhouse

TABLE OF CONTENTS

Definitions

TWAIN-compliant data source

an acquisition source that complies with the TWAIN protocol.

source manager

a middle-ware that communicates between applications and TWAIN-compliant data sources.

TWAIN session

a state where a TWAIN variable has established a connection with the source manager. This process is known as ‘Opening a TWAIN session’.

acquiring

the act of transmitting an image from the data source to the application.

Requirements

TWAIN support was added to Paradox in version 9, Service Pack 1. ObjectPAL support for TWAIN came later in version 10.

- You must be running Paradox 10 (or later).
- The file pxtwn32.dll must be located in the same directory as pdxwin32.exe
- The file twain_32.dll must be present in your operating system directory (c:\winnt, c:\windows)
- At least one TWAIN-complaint data source installed on the machine.

Twain Support in ObjectPAL

Introduction

There are many advantages of putting TWAIN support into your application. This introduction discusses some of the common questions regarding TWAIN, and its uses.

What is TWAIN and TWAIN support?

TWAIN is an interface that enables applications to transmit images from imaging hardware devices. These devices primarily include digital cameras and page scanners. Images when acquired are scanned directly into the application for direct manipulation.

TWAIN support is a term for the client application. Applications which conform to the TWAIN interface and acquire images into their application are said to have TWAIN support. A data source which complies with the TWAIN interface is said to be TWAIN-compliant.

TWAIN is not an acronym. The word TWAIN is from Kipling's, "The Ballad of East and West" – "… and never the twain shall meet." However, many people believe that it is indeed an acronym. Perhaps the best entry for an acronym was "Technology Without An Interesting Name."

What are the benefits of TWAIN support?

Nearly every application that supports some form of photo-identification will benefit by including TWAIN support with their application. A health club that has photo-ID for its members or a hospital and employee identification tags. Even things as simple as a plant collection, fish database, and coin collections can benefit by having the ability to scan images into a table.

TWAIN Support Overview

In Paradox 9, interactive TWAIN support was added in SP1. This gave the ability to acquire images from scanners and cameras into tables and forms. However, not until Paradox 10 did the support become available to the developer. The TWAIN type in ObjectPAL gives the developer the ability to acquire images from scanners and cameras, and place those images into files, graphic types, and tables. The TWAIN type also contains functions to enumerate through the list of data sources on the computer. With the addition of the TWAIN type developers can add the ability to acquire and manage scanned images.

A TWAIN variable type is used to acquire images from TWAIN-compliant devices. Communication between Paradox (or more precisely the TWAIN variable) and the source manager is established by opening a session with the source manager.

The source manager is installed when the first TWAIN-compliant device is installed onto the workstation. The latest version of the twain_32.dll, which contains the source manager, is placed into the operating system directory by the installation program of a TWAIN-complaint device.

For Windows ME, and Windows 2000, Microsoft came out with WIATWAIN, an acronym for Windows Imaging Acquisition TWAIN. This means that all Windows ME and 2000 machines come with the source manager installed as part of the operating system. WIATWAIN is a windows driver and interface for data sources. Regardless of the operating system, Paradox still behaves the same way.

TWAIN support is treated like a plug-in for Paradox. The file ‘pxtwn32.dll’ contains the interface that Paradox uses to communicate with the source manager. If this DLL is not present (or not plugged-in), then all the functionality related to TWAIN support is effectively disabled. Note that you cannot use the Paradox 9 SP1 version of pxtwn32.dll in the absence of its existence in Paradox 10. The only similarity between the two files is the filename. The version shipped with Paradox 9 SP1 does not provide an interface to ObjectPAL.

After a connection has been established with the source manager, the application can enumerate the available data sources, acquire images, get the default source for the machine, and set the data source for the session.

Images acquired via a TWAIN variable can be transferred to three different formats. ObjectPAL can acquire to a file, a graphic type, or a graphic field that has been placed in edit-mode. There is also an option to display the user-interface for the data source before acquiring.

Checking for Twain Support

To check for TWAIN support, the application would typically call *isTwainAvailable*() on the startup of the application, and store the result for later reference.

```
#FormData1::var
var
      loTwain           Logical
endvar

#FormData1::init
method init(var eventInfo Event)

loTwain = isTwainAvailable()

endMethod
```

Whenever the application is about to display user-interface items regarding acquisition, the variable loTwain is referenced. If this value returns FALSE, then the user-interface items are disabled, hidden, or removed.

This allows applications to take advantage of workstations that have TWAIN-compliant devices attached and still function with those workstations with no TWAIN-compliant devices.

Each developer/application will have their own methods of handling application settings. This example is not to point out the best method, but to suggest that storing the return value of *isTwainAvailable*() for later reference is a good idea.

Opening a Session

As stated previously, communication between the source manager and the TWAIN variable type is handled by a session. Typically, this is known as 'opening a TWAIN session'. The session is the only means to talk to the source manager. To establish a session, the function *open*() is called.

While not efficient, this does provide another means to verify that TWAIN support is available. If the return code from *open*() is false, then the TWAIN session could not be established.

Lastly, developers can code an *isAssigned*() call to determine if the session was established. Normally, conditional code would be wrapped around the *open*() call.

Like most session style objects, the *close*() function will terminate the session. Like the TCursor object, when the variable goes out of scope, the variable is unassigned, and the session is terminated. A simple example is described below.

```
btnExample::pushbutton
method pushbutton(var eventInfo Event)
var
      twSes Twain
endvar

;// open a session to the source manager, and report
;// an error if one is returned.
if not twSes.open() then
      errorShow()
      return
endif

if twSes.isAssigned() then

      ;// acquire, select source, etc…
endif

;// close the session.
twSes.close()

endMethod
```

Default Sources

The term default source refers to the system-wide TWAIN-compliant data source that is used by default for acquiring images. The only supported way of changing the system default data source is through the Select Source dialog. If the source for a TWAIN session is not explicity set, the session inherits the system's default data source for the acquisition.

The following example describes how to get the default source for the machine. This information is useful if you are coding your own Select Source dialog. When coding your own Select Source dialog, the application is responsible for saving, and retrieving the selected data source. The application is also responsible for ensuring the data source still exists on the system.

```
btnExample::pushbutton
method pushbutton(var eventInfo Event)
var
      twSes       Twain
      strSource   String
endvar

if not twSes.open() then
      errorShow()
      return
endif

strSource = twSes.getDefaultSource()
msgInfo ( “Default Source”, strSource )

twSes.close()
endMethod
```

Showing the Select Source Dialog

The Select Source dialog is used to select a data source to acquire from. The data source that is selected from the Select Source dialog becomes the default data source for the machine.

```
btnExample::pushbutton
method pushbutton(var eventInfo Event)
var
      twSes             Twain
      strOldSource      String
      strNewSource      String
endvar

if not twSes.open() then
      errorShow()
      return
endif

strOldSource = twSes.getDefaultSource()
twSes.ShowSelectSourceDlg()
strNewSource = twSes.getDefaultSource()
```

```
if strOldSource = strNewSource then
      msgInfo ( “Twain”, “The data source has not changed.” )
else
      msgInfo ( “Twain”, “The data source has changed” )
endif

twSes.close()

endMethod
```

Enumerating Data Sources

A TWAIN session can enumerate all the data sources on a machine. This is as simple as opening a TWAIN session and calling *enumSourceNames*(). This information returns the source names of the data sources located on the machine. The index of each data source is stored in the key of the returned dynArray entry. The index should only be used as a means to retrieve the data source name, and should not be stored for later use.

```
btnExample::pushbutton
method pushbutton(var eventInfo Event)
var
      twSes             Twain
      strOldSource      String
      dynSourceNames    DynArray[] String
endvar

if not twSes.open() then
      errorShow()
      return
endif

twSes.enumSourceNames( dynSourceNames )
dynSourceNames.view()

twSes.close()

endMethod
```

Setting the source for a session

To set a data source for the TWAIN session, the caller would use *setSource*(). If the source is not found in the source list, then the *setSource*() returns FALSE. The following example will select the first source name in the source name list.

```
BtnExample::pushbutton
Method pushbutton(var eventInfo Event)
var
      twSes             Twain
      strOldSource      String
      dynSourceNames    DynArray[] String
endvar

if not twSes.open() then
      errorShow()
      return
```

```
endif

twSes.enumSourceNames ( dynSourceNames )

;// if there was at least one source name, then set it.
if dynSourceNames.size() >= 1 then
      twSes.setSource ( dynSourceNames[ 1 ] )
else
      msgInfo ( “Twain”, “There are no sources available.” )
endif

twSes.close()

;// ...

endMethod
```

Getting the source for the session

The source for the current session can be set by calling *setSource*(). Likewise, to retrieve the selected source use *getSource*(). The difference between *getSource*(), and *getDefaultSource()*, is that in the latter we are retrieving the default source for the system, and *getSource*() retrieves the source for the session. In the event that the source has explicitly been set by the developer, calling *getSource*() without setting the session source will return the same result as *getDefaultSource*(). For this example, we'll assume we always want to acquire from the camera, regardless of what they have selected.

```
BtnExample::pushbutton
Method pushbutton(var eventInfo Event)
var
      twSes             Twain
      strSourceName     String
      dynSourceNames    DynArray[] String
endvar

if not twSes.open() then
      errorShow()
      return
endif

strSourceName = twSes.getSource()
if strSourceName <> “QV QuickCam 2.33 32bit” then
      if not twSes.setSource ( “QV QuickCam 2.33 32bit” ) then
            msgInfo ( “Twain”, “The camera isn’t installed.” )
            return
      endif
endif

twSes.close()

endMethod
```

Getting the number of sources

In Setting the source for a session, the example set the source based on results from the *enumSourceNames*() method of the Twain type. The code relies on the *size*() function of the DynArray type to return the number of sources available in the array. By using *getSourceCount*(), the number of TWAIN-compliant data sources is retrieved.

Since at least one TWAIN-compliant device is required for TWAIN support to be available, this function should never return 0. Thus, we can safeguard our code by getting the source count after opening the TWAIN session. The example below clearly illustrates how we can produce more efficient code:

```
BtnExample::pushbutton
Method pushbutton(var eventInfo Event)
var
      twSes             Twain
      strOldSource            String
      dynSourceNames    DynArray[] String
endvar

if not twSes.open() then
      errorShow()
      return
endif

if twSes.getSourceCount() = 0 then
      errorShow()
      return
endif

twSes.enumSourceNames ( dynSourceNames )
twSes.setSource ( dynSourceNames[ 1 ] )

twSes.close()

endMethod
```

Acquiring Images

With the Twain type, developers can acquire images into three different formats. For each format, the user-interface can be shown or not shown.

Paradox does not support multiple image handling. Certain cameras can buffer images in the camera memory, and send them in succession. Paradox does not support the ADF (automatic document feeders), such as scanners. The reason is quite simple. It makes little sense to scan 40 images into the same graphic field in a table. A graphic type can only hold one graphic, and a file can only have one filename.

Paradox does not support 32bit images, regardless if they are scanned in, or imported from files. With other topics such as graphics handling, 24bit scanned images can be saved to disk in .gif, .jpg formats to keep the file size down.

Paradox 10 SP1 does not support acquiring to a graphic type. Attempting to acquire to a graphic type will result in an error stating that an unassigned variable was referenced. In fact, upon further investigation, it appears that it is calling the ‘acquire to file’ TWAIN function. Paradox 10 SP2 corrects this behavior.

Acquiring to a file

To acquire to a file, only the filename is required. Like other ObjectPAL functions, if the filename is not preceded with an absolute path, the working directory is assumed.

```
BtnExample::pushbutton
Method pushbutton(var eventInfo Event)
var
      twSes             Twain
      strFileName       String
endvar

if not twSes.open() then
      errorShow()
      return
endif

strFileName = “acquired.bmp”

;// acquire to the filename, acquired.bmp.
;// show the user interface.
if not twSes.acquire ( strFileName, true ) then
      errorShow()
endif

twSes.close()

endMethod
```

Acquiring to a graphic object

Acquiring to a graphic object is straight-forward. The first parameter of the *acquire*() method is the uiObject that refers to the graphic object. The graphic object can be in run or design mode.

```
BtnExample::pushbutton
Method pushbutton(var eventInfo Event)
var
      twSes             Twain
      uiGraphic         UIObject
endvar

if not twSes.open() then
      errorShow()
      return
endif

;// acquire to the field, fldGraphic.
;// show the user interface.
uiGraphic = fldGraphic

if not twSes.acquire (uiGraphic, true ) then
```

```
      errorShow()
endif

twSes.close()

endMethod
```

Acquiring to a graphic field object

Acquiring to a graphic field is a little more complex than a graphic object. The form must be in edit mode. The uiObject refers to the graphic field. If these two conditions are not met, then the acquire method will log an error to the error stack.

```
BtnExample::pushbutton
Method pushbutton(var eventInfo Event)
var
      twSes             Twain
      uiGraphic         UIObject
endvar

if not twSes.open() then
      errorShow()
      return
endif

;// place the form in edit mode.
;// acquire to the graphic field object, fldGraphic.
;// show the user interface.
fldGraphic.edit()

uiGraphic = fldGraphic

if not twSes.acquire (uiGraphic, true ) then
      errorShow()
endif

twSes.close()

;// end Edit mode which will post the record.
fldGraphic.endEdit()

endMethod
```

Acquiring to a graphic type

This example does not work in Paradox 10 SP1. Acquiring to a graphic type returns an error stating usage of an unassigned variable. This issue has been resolved in Paradox 10 SP2.

However, for completeness, the example, plus an explanation is listed here. Its intended functionality is the ability to acquire an image from a data source and place the image into the graphic variable. Then the graphic variable can be written to disk, stored into a table, or displayed on screen.

```
BtnExample::pushbutton
```

```
Method pushbutton(var eventInfo Event)
var
      twSes             Twain
      gAcquired         Graphic
endvar

if not twSes.open() then
      errorShow()
      return
endif

;// acquire to the graphic type, gAcquired.
;// show the user interface.
if not twSes.acquire (gAcquired, true ) then
      errorShow()
endif

qAcquired.writeToFile ( “c:\\temp\\acquired.bmp” )

twSes.close()

endMethod
```

Function Reference

isTwainAvailable *Twain*

```
proc isTwainAvailable() Logical

Parameters  :     none
Returns     :     Logical
```

This function returns TRUE if TWAIN support is available, and FALSE if TWAIN support is unavailable at the time of the function call.

IsTwainAvailable() will check the existence of the TWAIN interface plug-in (pxtwn32.dll). If this is found, then a connection to the source manager (twain_32.dll) is attempted. If the connection was established successfully, a further is made to check to ensure that there is at least one source available for acquiring before returning TRUE to the caller.

acquire (to a UIObject type) Twain

```
Method acquire( const ui uiObject, const bShowUI Logical ) Logical

Parameters  :     ui          UIObject to acquire to
                  bShowUI     show the user interface of the source
Returns     :     Logical
```

This function acquires an image into the passed UI object. The passed UI object must be a graphic object, or a graphic field object that is in edit mode. If either of these two conditions are not met, the function fails.

If the acquisition dialog is cancelled, the function still returns TRUE.

acquire (to a graphic type) Twain

```
Method acquire( const gAcquired Graphic, const bShowUI Logical )
Logical

Parameters  :     gAcquired   graphic object to acquire to
                  bShowUI     show the user interface of the source
Returns     :     Logical
```

This method requires Paradox 10 SP2 to function properly. In Paradox 10 SP1, this method states an error about the usage of an unassigned variable. For completeness, however, the summary is listed below.

This function acquires an image into the passed graphic type. If the acquisition dialog is cancelled, the function still returns TRUE. To determine if dialog was cancelled, verify that the graphic type is assigned. If the graphic type is unassigned, then the dialog was cancelled.

acquire (to a file) Twain

```
Method acquire( const strFileName String, const bShowUI Logical )
Logical

Parameters  :     strFileName name of the file to acquire to.
                  bShowUI     show the user interface of the source
Returns     :     Logical
```

This function acquires an image into the passed filename. If the filename is not preceded with an absolute path, the working directory is assumed. The file can be prefixed with an alias, to store the file at the location of the passed alias.

This method can acquire 32bit images to disk.

Specifying an extension to the filename (such as .jpg, or .gif) will not convert the file. The file will be stored as a bitmap regardless.

If the acquisition dialog is cancelled, the function still returns TRUE. To determine if dialog was cancelled, verify existence of the file on disk. If the file is not to be found, then the dialog was cancelled.

close Twain

```
Method close() Logical

Parameters  :     none
Returns     :     Logical
```

This function returns TRUE when the session is closed. It will return FALSE, if the variable is in a state where it cannot be aborted (such as acquiring an image).

enumSourceNames Twain

```
Method enumSourceNames( var dyn DynArray[] String ) Logical

Parameters  :     dyn         dynamic array to hold results
Returns     :     Logical
```

This function enumerates the sources that exist on the current machine, and places them into the passed dynamic array. These same names can be used to set the source for the session.

getSource Twain

```
Method getSource() String

Parameters  :     none
Returns     :     String      current data source name
```

This function returns the source name for the session. If the source name for the session has not been set, then the default source for the machine is used.

getDefaultSource Twain

```
Method getDefaultSource() String

Parameters  :     none
Returns     :     String      system default data source
```

This function returns the system default TWAIN data source. This data source is the selected entry in the system Select Source dialog. This function is useful for resetting/setting or comparing the current session data source to that of the system.

getSourceCount Twain

Method **getSourceCount**() LongInt

```
Parameters  :     none
Returns     :     LongInt     returns the number of data sources
```

This function returns the number of TWAIN-compliant data sources that exist on the machine. This function should never return 0. If there are no data sources existing on the machine, then the function *isTwainAvailable*() would return FALSE, and open() would not establish a session.

isAssigned Twain

Method **isAssigned**() Logical

```
Parameters  :     none
Returns     :     Logical
```

This function returns TRUE if the twain session has been opened, and FALSE if not. Generally, developers should code around the *open*() call to deal with errors or problems establishing the session.

open Twain

Method **open**() Logical

```
Parameters  :     none
Returns     :     Logical
```

This function returns TRUE if a TWAIN session could be established. An application can have multiple TWAIN sessions occurring at the same time. However, one and only one can be communicating with the TWAIN manager at any time.

setSource Twain

```
method setSource( const strSourceName String ) Logical

Parameters  :     strSourceName     source name to set
Returns     :     Logical
```

This function sets the source name for the session. If the source name could not be found, then the function returns FALSE.

Note: As of Paradox 10 SP2 - if the function fails to set the passed source name, coding an *errorShow*() to display the error will have no effect. No error dialog will be displayed.

showSelectSourceDlg Twain

```
Method showSelectSourceDlg() Logical

Parameters  :     none
Returns     :     Logical
```

This function displays the Select Source dialog from the source manager. This is the only supported way of changing the default data source for the system. However, for most TWAIN enabled applications, it is best to store the default source for the application, instead of relying on the system data source. By storing the default source for the application, the administrator gains more control over what devices may render images, and what devices cannot.

Sample Projects

Picture Saver

The Picture Saver application mimics a photo-album. It contains thumbnails of all the pictures stored in a table. The table has two fields, the first is the key field, and the second the path and filename of the graphic. Graphics are generally not stored in tables since the BDE expands graphic files into bitmaps before storing the graphic. Graphics may have been acquired into a .jpg or .gif format, and thus, the difference in file size warrants special handling of the graphics. In the Picture Saver application, we store the graphic outside of the table, and store the filename in the table.

Picture Saver Startup Script

Abstract: The Picture Saver script sets the alias for the application and calls the main form. Use the startup script to ensure that the 'PictureSaver' alias is set before reviewing the samples.

Uses: nothing

Picture Saver Dialog

Abstract: The Picture Saver dialog contains a thumbnail view of the pictures in the photo album. To accomplish the thumbnail view, we place the key field and a graphic object in a multi-record object. On the newvalue() event of the key field, we load the appropriate graphic from the file, and assign it to the graphic object.

The dialog has one button, labeled Add, which calls the Add Picture dialog.

Uses: *isTwainAvailable*()

Add Picture Dialog

Abstract: The Add Picture Dialog is used to acquire images from a data source, or to import images into the table. The Select Source button brings up the Picture Saver Select Source dialog

Uses: *open*(), *close*(), *setSource*(), *acquire*()

Select Source Dialog

Abstract: Our Select Source dialog mimics the behavior and operations of the Select Source dialog displayed by source manager. In this dialog, we ask the source manager for a list of data sources, and populate a list box with them. The code has an OK and a Cancel button, and returns the name of the data source selected.

Uses: *open*(), *close*(), *enumSourceNames*(), *getSourceCount*(), *getDefaultSource*()

Advanced Techniques

In this article, the samples use some advanced techniques which should be explored while designing a TWAIN implementation. These techniques are not mandatory for any TWAIN enabled application. If the application does not employ the advanced techniques, the application will still function.

Saving the Application Default Source

Code should never rely on the system default source for acquisition. A straight *acquire*() could acquire from a different source than expected, and thus yield unexpected results. Creating and maintaining the default source for the application prevents users from acquiring from unwanted or unsupported sources. This concept can be expanded to include user default sources as well.

Handling Graphics

Graphics require special handling. When a graphic is read from a file, the file is converted from its original format to a native bitmap form. This is typical in other applications. In Paradox 10, graphics can be exported to disk in more compressed formats, such as GIF, and JPEG. The largest advantage of leveraging the new graphic features in Paradox 10 is size.

Graphics are generally not stored in tables since the BDE expands graphic files into bitmaps before storing the graphic. Thus, if a developer reads an item into a graphic type, and places the graphic type into the table, the internal format of the file is not preserved. This causes the table to grow increasingly large with each graphic. By storing the filename of the graphic object as a reference to the graphic, the size of the table is reduced, and the number of entries is only limited to the size of the drive where the table is located.

Setting the System Default Source

One trick to get the default TWAIN source without showing the Select Source dialog is to read the profile entries that correspond to the default data source. The filename is stored in the ‘Default Source’ entry of the ‘TWAIN’ section in the ‘win.ini’ file.

```
strDSFileName = readProfileString( “win.ini”, “TWAIN”, “Default
Source” )
```

returns the complete path and filename of the system default data source. It is then left up to the developer to equate the friendly name displayed in the Select Source dialog to the filename returned from the above function.

Conversely, the code

```
writeProfileString( “win.ini”, “TWAIN”, “Default Source”,
“digicam.ds”)
```

writes the filename of a data source to the windows profile. The onus is on the developer to ensure the filename exists on the local machine.

The TWAIN Working Group does not support these methods of getting and setting the default data source. If the system data source changes in the midst of a twain session, it is conceivable that the user could acquire from a different data source than the one selected. Thus, the only supported means of setting and getting the system data source is through the source manager.

Caveats with TWAIN support in ObjectPAL

With the current implementation of Paradox 10 SP1, the function *twain.acquire(var gr graphic, var bShow Logical)* does not work as documented. However, one could acquire to a file, then perform a *graphic.readFromFile()* method after the acquisition was complete to achieve the same results. Paradox 10 SP2 has corrected the behavior of the acquire method.

The table view has no user interface for acquiring images. This appears to be an oversight in the original implementation.

The function *twain.isAssigned()* is not documented in the help.

The help panel for *twain.showSelectSourceDlg*() states that the prototype for this function is showSelectSourceDialog(), not showSelectSourceDlg().

The documentation for TWAIN support is quite limited, and in most cases incorrect. Most of the arguments are actually of const type, not var. The help does not automatically point to the correct help panel from the source code window.

Paradox is not capable of acquiring 32bit graphic images to graphic objects, or graphic field objects. If a 32bit image is acquired to one of these objects, the image will appear slanted, or distorted. 32bit images can be acquired to a file successfully. In general, applications should not acquire images greater than 24bits to adhere to the limitations of Paradox.

Paradox can only receive one image per acquisition. Acquiring multiple images, such as time-lapsed stills, or a roll of film on a digital camera cannot be performed using the Paradox TWAIN implementation. This is working as designed. To acquire time-lapsed stills, perform an acquire with no UI, sleep for a period of time, and acquire the next image. This gives the flexibility of posting a record, or storing the image on disk before the next acquisition takes place.

Where can I find more information about TWAIN?

The TWAIN working group can be found on the web, at www.twain.org.