



DALSA • 7075 Place Robert-Joncas, Suite 142 • St-Laurent, Quebec, Canada • H4M 2Z2
<http://www.imaging.com>

Sapera LTTM

ActiveX Manual

Edition 6.00

Part number OC-SAPM-AXCP1



NOTICE

© 2004-2006 DALSA Corp. All rights reserved.

This document may not be reproduced nor transmitted in any form or by any means, either electronic or mechanical, without the express written permission of DALSA Corp. Every effort is made to ensure the information in this manual is accurate and reliable. Use of the products described herein is understood to be at the user's risk. DALSA Corp. assumes no liability whatsoever for the use of the products detailed in this document and reserves the right to make changes in specifications at any time and without notice.

Microsoft® is a registered trademark; Windows®, Windows NT®, Windows® 2000, and Windows® XP, are trademarks of Microsoft® Corporation.

All other trademarks or intellectual property mentioned herein belong to their respective owners.

Part number OC-SAPM-AXCP1

Printed on August 1, 2006

Printed in Canada

Contents

INTRODUCTION	1
OVERVIEW OF THE MANUAL	1
USING THE MANUAL	1
GETTING STARTED	3
ABOUT SAPERA LT ACTIVE X CONTROLS	3
BASIC HIERARCHY CHART	4
REQUIREMENTS	4
<i>Minimum System Requirements</i>	4
<i>Board Requirements (optional)</i>	4
INSTALLATION PROCEDURE	4
FILE LOCATIONS	5
USING THE SAPERA LT ACTIVE X CONTROLS	7
DEMO PROGRAMS	7
USING THE CONTROLS	8
<i>Compiling Demos in Visual Basic 6</i>	8
<i>Using Active X Controls in Visual Basic 6</i>	8
<i>Compiling Demos in Visual Studio .NET</i>	10
<i>Using Active X Controls in Visual Studio .NET</i>	10
<i>Upgrading Existing Applications in Visual Studio .NET</i>	13
<i>Installing Packages in Delphi</i>	13
<i>Compiling Demos in Delphi</i>	14
<i>Using Active X Controls in Delphi</i>	14
PROGRAMMING EXAMPLES	16
<i>Error Management</i>	16
<i>Acquiring Images</i>	18
<i>Displaying Images</i>	20
<i>Working with Buffers</i>	21
ACQUISITION CONTROL	23
OVERVIEW	23
PROPERTY PAGES	24
<i>General Tab</i>	25
<i>Events Tab</i>	26
<i>Transfer Settings</i>	27
<i>Bayer Conversion</i>	28
<i>Buffer Tab</i>	29

<i>Display Tab</i>	30
SAPACQUISITION	31
<i>SapAcquisition Member List</i>	31
<i>SapAcquisition Member Description</i>	33
SAPBUFFER	72
<i>SapBuffer Member List</i>	72
<i>SapBuffer Member Description</i>	73
SAPDISPLAY	88
<i>SapDisplay Member List</i>	88
<i>SapDisplay Member Description</i>	88
SAPLUT	94
<i>SapLut Member List</i>	94
<i>SapLut Member Description</i>	95
SAPRECTANGLE	108
<i>SapRectangle Member List</i>	108
<i>SapRectangle Member Description</i>	108
SAPDATAMONO	110
<i>SapDataMono Member List</i>	110
<i>SapDataMono Member Description</i>	110
SAPDATARGB	111
<i>SapDataRGB Member List</i>	111
<i>SapDataRGB Member Description</i>	111
SAPDATAYUV	112
<i>SapDataYUV Member List</i>	112
<i>SapDataYUV Member Description</i>	113
SAPDATAHSI	114
<i>SapDataHSI Member List</i>	114
<i>SapDataHSI Member Description</i>	114
SAPDATAHSV	116
<i>SapDataHSV Member List</i>	116
<i>SapDataHSV Member Description</i>	116
SAPDATAFRGB	117
<i>SapDataFRGB Member List</i>	117
<i>SapDataFRGB Member Description</i>	118
DALSA CONTACT INFORMATION	121
SALES INFORMATION	121
TECHNICAL SUPPORT	122
GLOSSARY OF TERMS	123
INDEX	127

Introduction

Overview of the Manual

The *Sapera LT ActiveX manual* covers the following topics:

- **Getting Started**
Sapera LT ActiveX description and requirements.
- **Using the Sapera LT ActiveX Controls**
Description of the procedures to follow for various development environments in creating an application based on ActiveX controls. Description of the programming techniques to follow in writing applications.
- **Acquisition Control**
Description of the main ActiveX control including all of its sub-components.
- **Hierarchy Chart**
Illustrated hierarchy diagram of the complete library.
- **DALSA Contact Information**
Phone numbers, web site, and important email addresses.

This manual exists in printed, Windows Compiled HTML Help, and Adobe Acrobat (PDF) formats. The Help and PDF formats make full use of hypertext cross-references. The PDF format offers links to DALSA's home page on the Internet, located at <http://www.imaging.com/>.

DALSA's Web site contains documents, updates, demos, errata, utilities, and more.

Using the Manual

File names, directories, and Internet sites will be in bold text (for example, **setup.exe**, **c:\windows**, **<http://www.imaging.com>**). Function parameters will be in italics (for example, *xlen*). Source code, code examples, text file listings, and text that must be entered using the keyboard will be in typewriter-style text (for example, [PixelClock]).

Menu and dialog actions will be indicated in bold text in the order of the instructions to be executed, with each instruction separated by bullets. For example, going to the File menu and choosing Save would be written as **File•Save**.

Getting Started

About Sopera LT ActiveX Controls

An ActiveX Control (originally called an OLE Control) is a software component with a binary standard interface. It can be compared to a dynamic-link library (DLL) with a standardized method of accessing its implementation.

An ActiveX Control must be contained within an application called an ActiveX Container (OLE Container). Typically, ActiveX Controls are inserted into a form or a dialog window and used as a standard control, such as a button or an edit box. They are represented by icons depicting their respective functions.

The ActiveX Control interface is composed of three main parts:

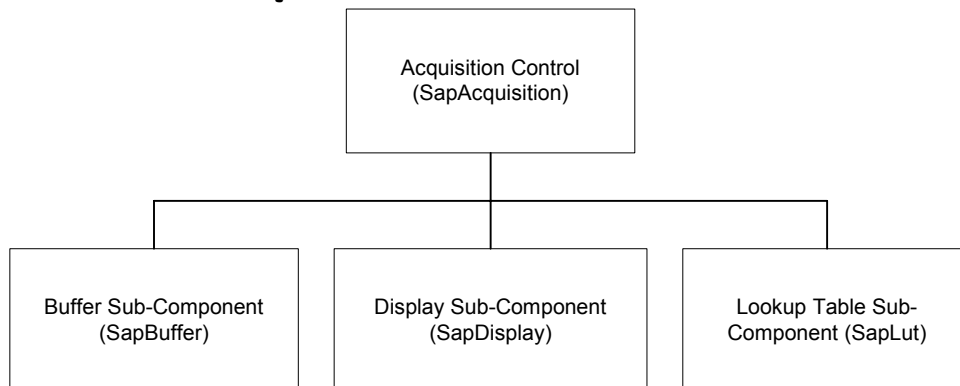
- **Properties** are variables that describe the control state. Properties encapsulate the Sopera LT parameters and the control options.
- **Methods** are functions used to call control tasks. They are mapped to Sopera LT functions.
- **Events** are signals sent to the control's container to inform it of events occurring within the control.

There are several advantages to using Sopera LT ActiveX Controls versus the Sopera++ API:

- Their language-independent interface supports several development tools, such as Visual Basic 6, Visual Studio .NET, and Borland Delphi.
- There is no need for header files and import libraries, thus facilitating the integration of Sopera LT into your application.
- They can be easily integrated along with third-party components within the same application.
- ActiveX persistence storage allows the storage and retrieval of property values to and from the disk without writing any code.
- Sopera LT resource allocation is hidden by the control. This allows the modification of Sopera LT resource parameters at design-time.

Sopera LT ActiveX Controls can contribute significantly to reducing your application development time. They are an essential part of a Rapid Application Development (RAD). A few lines of code are sufficient to build a simple Sopera LT program when using ActiveX.

Basic Hierarchy Chart



Requirements

Sapera LT can operate on systems with or without DALSA boards installed. Operating Sapera LT on the system processor without a DALSA board installed provides only minimal functionality and performance. Below is a list of the required components for the two operation modes.

Minimum System Requirements

- PCI-bus IBM PC or compatible with Pentium class or later processor
- Windows® 2000, or Windows® XP (**Note:** Windows NT 4.0 and Embedded are no longer officially supported)
- One of the following development tools:
 - Microsoft® Visual Basic 6
 - Microsoft® Visual Studio .NET 2003 or later
 - Borland® Delphi 7 or later

Board Requirements (optional)

- One of several Sapera LT compatible boards.
- Corresponding Sapera LT compatible board drivers.

Installation Procedure

Sapera LT ActiveX Controls are installed automatically while installing Sapera LT. If you have not yet installed Sapera LT, see the installation procedures within the *Sapera LT User's Manual*.

File Locations

The table below lists the different file groups and locations:

Description	Location
Executables for ActiveX controls	\$(SAPERADIR)\Components\ActiveX\Bin
Borland Delphi support	\$(SAPERADIR)\Components\ActiveX\Delphi
Visual Basic 6 demos	\$(SAPERADIR)\Demos\ActiveX\VB6
Visual Studio .NET demos (C# and VB.NET)	\$(SAPERADIR)\Demos\ActiveX\NET
Borland Delphi demos	\$(SAPERADIR)\Demos\ActiveX\Delphi

Using the Sapera LT ActiveX Controls

Demo Programs

Sapera LT includes demos that use the ActiveX controls. A Visual Basic 6.0 project group (`\\Sapera\Demos\ActiveX\Vb6\SapActiveXDemos.vbg`), Visual Studio .NET solution (`\\Sapera\Demos\ActiveX\NET\SapActiveXDemos.sln`), and Delphi 7 package solution (`\\Sapera\Demos\ActiveX\Delphi\SapActiveXDemos.bpg`) include all ActiveX demo projects.

The following describes the different Sapera LT ActiveX demo programs.

Acquisition Demo (Visual Basic 6.0)

`$(SAPERADIR)\Demos\ActiveX\Vb6\AcqDemo\AcqDemo.vbp`

This program demonstrates the basic acquisition functions included in the Sapera LT ActiveX controls. It allows you to load an acquisition file and then acquire images, either in continuous or in one-shot mode. Clicking the right mouse button over the image display area allows you to modify the acquisition control properties as well as loading and saving images.

The minimum requirements to run this demo are a Sapera-compatible frame grabber and an analog or digital camera.

The supplied executable is built using Visual Basic 6.

Acquisition Demo (VB.NET)

`$(SAPERADIR)\Demos\ActiveX\NET\VbAcqDemo\VbAcqDemo.vbproj`

This program demonstrates the basic acquisition functions included in the Sapera LT ActiveX controls. It allows you to load an acquisition file and then acquire images, either in continuous or in one-shot mode. Clicking the right mouse button over the image display area allows you to modify the acquisition control properties as well as loading and saving images.

The minimum requirements to run this demo are a Sapera-compatible frame grabber and an analog or digital camera.

Acquisition Demo (C#)

\$(SAPERADIR)\Demos\ActiveX\NET\VsAcqDemo\VsAcqDemo.csproj

This program demonstrates the basic acquisition functions included in the Spera LT ActiveX controls. It allows you to load an acquisition file and then acquire images, either in continuous or in one-shot mode. Clicking the right mouse button over the image display area allows you to modify the acquisition control properties as well as loading and saving images.

The minimum requirements to run this demo are a Spera-compatible frame grabber and an analog or digital camera.

Acquisition Demo (Delphi 7)

\$(SAPERADIR)\Demos\ActiveX\Delphi\AcqDemo\AcqDemo.dpr

This program demonstrates the basic acquisition functions included in the Spera LT ActiveX controls. It allows you to load an acquisition file and then acquire images, either in continuous or in one-shot mode. Clicking the right mouse button over the image display area allows you to modify the acquisition control properties as well as loading and saving images.

The minimum requirements to run this demo are a Spera-compatible frame grabber and an analog or digital camera.

Using the Controls

This section offers information concerning demo and application compilation using ActiveX Controls for different development environments.

Note: The ActiveX demos have the same look and functionality for the supported development environments. Spera LT includes executables for the ActiveX demos using the Visual Basic 6.0 version. If you need to compile the ActiveX demos, use the following procedures.

Compiling Demos in Visual Basic 6

To compile the demos you must open the following project group in Microsoft Visual Basic 6.0

<SAPERADIR>\Demos\ActiveX\VB6\SapActiveXDemos.vbg

You can also directly open this project group by selecting **DALSA | Spera LT | Demos | ActiveX | All Demos (Visual Basic 6.0)** from the Start menu.

You can compile all the demos at once by selecting **File | Make Project Group**.

You can also compile the demos individually by right-clicking the desired project in the project group window, selecting **Set as Start Up**, and then selecting **File | Make AcqDemo.exe...**

Using ActiveX Controls in Visual Basic 6

Before you can use any Spera LT ActiveX control, you must enable it in your project by selecting **Project | Components**. Names for all the controls start with *Spera LT*. After the control is selected it

appears as an icon in your toolbox. Open your toolbox window if it is not visible by selecting **View | Toolbox**.

Inserting ActiveX Controls in a Form (Design-time)

To insert a control in a form, click the icon of your choice in the toolbox and then drag a rectangle in your form at the desired location.

Once the control is inserted, a corresponding variable is automatically created. The name of this variable is *<ControlName>1*, for example, **SapAcquisition1**. You can then use it to invoke properties and methods from your control.

Creating ActiveX Controls Dynamically (Runtime)

Most of the time an ActiveX control is created at design-time using an icon in a form. There may be situations however where the control must be created dynamically at runtime, for example, when you do not know in advance how many controls will be required.

A control is created dynamically using the **CreateObject** method by specifying the *Prog ID*, a string identifying the control and containing the following syntax:

```
<LibraryName>.<ControlName>Ctrl.1
```

An example of a Prog IDs is *SapAcquisition.SapAcquisitionCtrl.1*

Here is an example on how to dynamically create and dispose of an acquisition control.

```
' Create the control
Dim acq As Object
Set acq = CreateObject("SapAcquisition.SapAcquisitionCtrl.1")

'Use properties and methods of the control
acq.LoadConfigDlg (hWnd)
' . . .

'Destroy the control
Set acq = Nothing
```

Note that the resulting acquisition control has no visual representation, so you cannot use it to display acquired images.

Creating Objects (Runtime)

Structures in ActiveX controls are presented as COM objects. For example, *SapRectangle* and *SapDataMono* are COM objects. An object is created using the **New** statement as seen in the example below:

```
Dim NewROI As New SapRectangle

NewROI.Left = 120
NewROI.Top = 80
NewROI.Width = 240
NewROI.Height = 160
```

Trapping events

To install a handler for one of the events in the Sapera LT ActiveX controls, first select the form that will handle the event, then select **View | Code** to bring up the source code window. In the list at the top left, select the name of the required control, for example, *SapAcquisition1*. In the list at the top right, select the name of the required event, for example, *Xfer*. Here is an example handler:

```
Private Sub SapAcquisition1_Xfer(ByVal eventType As SapAcquisitionLIB.enumXferEvent,
    ByVal eventCount As Long, ByVal bufIndex As Long, ByVal isTrash As Boolean,
    ByVal pairIndex As Long)

    ' Event handling code goes here

End Sub
```

Compiling Demos in Visual Studio .NET

To compile demos you must open the following solution in Microsoft Visual Studio .NET 2003:

```
<SAPERADIR>\Demos\ActiveX\.NET\SapActiveXDemos.sln
```

You can also directly open this solution by selecting **DALSA | Sapera LT | Demos | ActiveX | All Demos (Visual Studio .NET)** from the Start menu.

You can compile all demos at once by selecting **Build | Build Solution**.

You can also compile the demos individually by right-clicking the desired project in the Solution Explorer, selecting **Set as StartUp Project** and then selecting **Build | Build CSAcqDemo**.

Using ActiveX Controls in Visual Studio .NET

Before you can use any ActiveX control in your form, you must enable the required control(s) in your project by selecting **Tools | Add/Remove Toolbox Items | COM Components**. The controls are listed under the names *Sap...Control* of the COM **Components** tab. After the controls are selected they appear as icons in the **Windows Forms** tab of your toolbox. Open your toolbox window if it is not visible by selecting **View | Toolbox**. Notice that at least one form must be open so that the controls are enabled in the toolbox.

Inserting ActiveX Controls in a Form (Design-time)

To insert a control in a form, click the icon of your choice in the toolbox and drag a rectangle in your form at the desired location.

Once the control is inserted a corresponding variable is automatically created. The name of this variable is *ax<ControlName>1*, for example **axSapAcquisition1** for C# and *Ax<ControlName>1*, for example **AxSapAcquisition1** for Visual Basic. You can then use it to invoke properties and methods from your control.

Creating ActiveX Controls Dynamically (Runtime)

Most of the time an ActiveX control is created at design-time using an icon in a form. However, there may be situations where the control must be created dynamically at runtime, for example, when you do not know in advance how many controls will be required.

Before you can create any control, you must add a reference to the library that contains the control by right-clicking the **References** folder from the *Solution Explorer*, selecting **Add References**, and then choosing the desired control(s) from the **COM** tab. The controls are listed under the names *Sapera LT...Control*. After the controls are selected they appear as subfolders under the **References** folder.

In C#, a control is created dynamically by allocating an instance of its corresponding class using the following syntax:

```
<LibName>.<ControlName> VarName = new <LibName>.<ControlName>Class();
```

Below is a C# example of how to dynamically create an acquisition control. The control will automatically be marked for garbage collection when the object goes out of scope.

```
Using SapAcquisitionLIB;  
  
// Create the control  
SapAcquisition acq = new SapAcquisitionClass();  
  
// Use properties and methods of your control  
acq.Online = true;  
...
```

In VB.NET, a control is created dynamically the same way it is created in Visual Basic 6.0.

Creating Objects (Runtime)

All the structures in ActiveX controls are presented as COM objects. For example *SapRectangle* and *SapDataMono* are COM objects. The syntax for creating an object in C# is identical to that used for creating controls dynamically.

Here is an example using VB.NET:

```
Dim NewROI As New SapAcquisitionLIB.SapRectangle  
NewROI.Left = 120  
NewROI.Top = 80  
NewROI.Width = 240  
NewROI.Height = 160
```

Trapping events

To install a handler for one of the events in the Sapera LT ActiveX Controls using C#, first select the ActiveX control in the Form Designer, then select **View | Properties** to bring up the properties window. Select the lightning-shaped icon at the top to bring up the list of available events. Double-click the name of the desired event, for example, *Xfer*. Here is an example handler:

```
private void axSapAcquisition1_Xfer(object sender,
    AxSapAcquisitionLIB.SapAcquisitionEvents_XferEvent e)
{
    // Event handling code goes here
    ...
}
```

If the control was created dynamically, then the above does not apply. You need to explicitly define the event handler function and enable it in the source code.

```
// Enable the event handler
acq.Xfer += new SapAcquisitionEvents_XferEventHandler(acq_XferEvent);
...

private void acq_XferEvent(enumXferEvent eventType, Int32 eventCount,
    Int32 bufIndex, Boolean isTrash, Int32 pairIndex)
{
    // Event handling code goes here
}
```

For VB.NET, first select the form that will handle the event, then select **View | Code** to bring up the source code window. In the list at the top left, select the name of the desired control, for example, *AxSapAcquisition1*. In the list at the top right, select the name of the desired event, for example, *Xfer*. Here is an example handler:

```
Private Sub AxSapAcquisition1_Xfer(ByVal sender As Object,
    ByVal e As AxSapAcquisitionLIB.SapAcquisitionEvents_XferEvent)
    Handles AxSapAcquisition1.Xfer

    ' Event handling code goes here

End Sub
```


Upgrading Existing Applications in Visual Studio .NET

If you have a .NET application using ActiveX controls of a previous version of Sapera LT, the following symptoms may occur after installing a new version:

- The ActiveX control windows disappear from the form they were inserted in. Also, the References folder has bad links of the type AxSap... or AxInterop.Sap... These links are marked by a yellow triangle icon.
- The ActiveX control windows are still present, but you experience abnormal behavior when running the application.

This means that your .NET wrappers are out-of-date and must be updated to the most current wrappers. To do so, use the following procedure:

- Close all the forms in your application.
- Delete bad links from the References folder.
- Right-click on the References folder and select **Add References**.
- Click the **Browse** button and select the <SAPERADIR>\Components\ActiveX\Ms.net directory.
- Select all wrapper files required by your application. The table below shows the syntax of the wrapper files with respect to their corresponding component.

Component	Wrapper Files
SapAcquisition	Interop.SapAcquisitionLIB.Dll AxInterop.SapAcquisitionLIB.Dll
Sap...	Interop.Sap...LIB.Dll AxInterop.Sap...LIB.Dll

- Click **Open** then click **OK**. The selected .NET wrappers will then be added to the References folder.
- Your .NET wrappers are now up-to-date. You can reopen the form(s) to verify that the ActiveX windows are present. If they are still missing, you can have to close your solution and then reopen it again.

Installing Packages in Delphi

Before opening or creating any project using ActiveX controls in Delphi, you must install the required package. A package is a Delphi-compatible component giving access to the ActiveX control in Pascal language.

To install the Sapera LT package, select **Components | Install Packages**, click the **Add** button in the *Design Packages* box and then select the **SapActiveXDelphi.bpl** file in the <Windows>\System32 directory. The package will appear in the list above under *Sapera LT ... Package*.

Compiling Demos in Delphi

Note: Be sure to install the Sapera LT Delphi package prior to opening a project containing ActiveX controls.

To compile the demos you must open the following project group in Delphi:

`<SAPERADIR>\Demos\ActiveX\Delphi\SapActiveXDemos.bpg`

You can also directly open this project group by selecting **DALSA | Sapera LT | Demos | ActiveX | All Demos (Delphi 7)** from the Start Menu.

You can compile all demos at once by selecting **Project | Build All Projects**.

You can also compile demos individually by right-clicking the desired project in the Project Manager window and selecting **Build**.

Using ActiveX Controls in Delphi

Note: Be sure to install the Sapera LT Delphi package before creating or opening a project containing ActiveX controls.

Setting Project Options

In order for your application to compile properly using the ActiveX controls, you must add the package path to the project options. You can do so by adding `$(SAPERADIR)\Components\ActiveX\Delphi` in **Project | Options | Directories/Conditionals | Search Path**. This path is required for compiling controls created at either design-time or runtime.

Inserting ActiveX Controls in your Application (Design-time)

Once the packages are installed the controls appear as icons in the **ActiveX** tab of your **Component Palette**. Open your palette if not visible by selecting **View | Toolbars | Component Palette**.

To insert a control in a form, click the icon of your choice in the component palette and then drag a rectangle in your form at the desired location.

A corresponding variable is automatically created once the control is inserted. The name of this variable is `<ControlName>1`, for example **SapAcquisition1**. You can then use it to invoke properties and methods from your control.

Creating ActiveX Controls Dynamically (Runtime)

Most of the time an ActiveX control is created at design-time using an icon within a form. There can be situations however where the control must be created dynamically at runtime, for example, when you do not know in advance how many controls will be required.

A control is created dynamically using the **CreateOleObject** method by specifying the *Prog ID*, a string identifying the control and containing the following syntax:

```
<LibraryName>.<ControlName>Ctrl.1
```

An example of a Prog IDs is *SapAcquisition.SapAcquisitionCtrl.1*

Below is an example on how to dynamically create an Acquisition Control. The control will be automatically destroyed when the image object goes out of scope.

```
{ Add the following references to the 'uses' statement }
uses ..., ComObj, SapAcquisitionLIB_TLB;

var
    Acq : SapAcquisition;

begin
    { Create the ActiveX control object }
    Acq := SapAcquisition(CreateOleObject('SapAcquisition.SapAcquisitionCtrl.1'));

    { Use object properties and methods }
    Acq.LoadConfigDlg(0);

    { Free the ActiveX control object }
    Acq := SapAcquisition(IDispatch(Unassigned));
end;
```

Creating Objects (Runtime)

Structures in ActiveX controls are presented as COM objects. For example, *SapRectangle* and *SapDataMono* are COM objects. An object is created using the **CreateComObject** function as seen in the example below:

```
{ Add the following references to the 'uses' statement }
uses ..., ComObj, SapAcquisitionLIB_TLB;

var
    Roi : SapRectangle;

begin
    { Create the COM object }
    Roi := SapRectangle(CreateComObject(CLASS_SapRectangle));

    { Use object properties and methods }
    Roi.Left := 64;

    { Free the COM object }
    Roi := SapRectangle(IUnknown(Roi));
end;
```

Trapping events

To install a handler for one of the events in the Sapera LT ActiveX controls using Delphi, first select the ActiveX control in the Form Designer., then select **View | Object Inspector** to bring up the corresponding window. Select the **Events** tab to view the list of available events. For each required event, enter the name of the handler function. Here is an example handler:

```
procedure TForm1.Acq_Xfer(ASender: TObject; eventType: TOleEnum;
    eventCount, bufIndex: Integer; isTrash: WordBool; pairIndex: Integer);
begin
    { Event handling code goes here }
end;
```

Programming Examples

This section presents Visual Basic 6 code samples that demonstrate common tasks when programming with the Sapera LT ActiveX Controls.

Error Management

Error reporting

All methods in the Sapera LT ActiveX Controls return a Boolean result to indicate success or failure. However, the actual error conditions are still reported as soon as they happen (before the methods return), using one of five predefined reporting modes:

- Error messages are sent to a popup window (the default)
- Error messages are sent to the DALSA Log Viewer
- Error messages are sent to the active debugger, if any
- Error messages are generated internally
- Error messages are sent to the application through an event. See the sections *Using ActiveX Controls in ...* for details about event handlers on the various supported platforms.

Use the **ErrorMode** property to set the current reporting mode, as follows:

```

' Send error messages to the Log Viewer
SapAcquisition1.ErrorMode = ErrorModeLog

' Send error messages to the debugger
SapAcquisition1.ErrorMode = ErrorModeDebug

' Simply generate error messages internally
SapAcquisition1.ErrorMode = ErrorModeCustom

' Send errors to application through an event
SapAcquisition1.ErrorMode = ErrorModeEvent

' Restore default reporting mode
SapAcquisition1.ErrorMode = ErrorModeNotify

```

For more details, see the reference section of this manual.

Retrieving Error Descriptions

No matter which reporting mode is currently active, it is always possible to retrieve the latest error message. If the error happened when calling a low-level Sapera function, then a related numeric code is also available. To retrieve this information, use the `LastErrorDesc` and `LastErrorValue` properties, as follows:

```

Dim Result as Boolean
Result = SapAcquisition1.Grab

If Result = False Then
    ' Get the latest error message
    ErrorDescText.Caption = SapAcquisition1.LastErrorDesc

    ' Get the latest error code
    ' See the Sapera Basic Modules Reference Manual for details
    ErrorValueText.Caption = SapAcquisition1.LastErrorValue
End If

```

In addition, the DALSA Log Viewer utility program, included with Sapera LT, provides an easy way to view error messages. It includes a list box that stores these messages as errors occur. Available options allow you to modify the different fields for display.

During development, it is recommended to start the Log Viewer before your application and then let it run so it can be referred to anytime to view detailed error descriptions. However, errors are also stored by a low-level service (running in the background), even if the utility is not running. Therefore, it is possible to run it only when a problem occurs within your application.

Acquiring Images

Acquisition Example

The example below demonstrates how to grab a live image into a buffer allocated in system memory using a Bandit-II board.

Acquiring an image requires one file (the CCF file) to configure the acquisition hardware. It defines both the characteristics of the camera and how it will be used with the acquisition hardware. Use the CamExpert tool provided with Sapera LT to create CCF files. Note that resource parameters can also be accessed individually (as described in the following sections).

```
Dim Result as Boolean

' Configure the acquisition hardware
' These settings will only be effective when writing the Online property
SapAcquisition1.Server = "Bandit_II_1"
SapAcquisition1.Device = "Standard Composite Video & Y/C"
SapAcquisition1.ConfigFile = "c:\CameraFiles\MyCamera.ccf"

' Allocate internal acquisition resources
SapAcquisition1.Online = True

' Start continuous acquisition
Result = SapAcquisition1.Grab

' Do other things ...

' Stop continuous acquisition
Result = SapAcquisition1.Freeze
Result = SapAcquisition1.Wait(5000, False)

' Free internal acquisition resources
SapAcquisition1.Online = False
```

For more details, see the reference material in this manual and the source code for the ActiveX demos included with Sapera LT.

Capabilities and Parameters

Sapera LT ActiveX Controls already include all the functionality necessary for most Sapera LT applications; however, some features are only available in the supporting low-level API. Capabilities and parameters are of particular interest in this case. Together they define a resource's ability and current state.

See the *Sapera Basic Modules Reference Manual* and the *Sapera Acquisition Parameters Reference Manual* for a description of all capabilities and parameters, and their possible values.

A capability is a value or set of values that describe what a resource can do. It is used to determine the possible valid values that can be applied to a resource's parameters. Capabilities are read only. Their value can be obtained from the acquisition control by using the **GetCap** and **GetCapEx** methods.

A parameter describes a current characteristic of a resource. Parameters can be read/write or read only. Their value can be obtained or set by using the **GetParam**, **GetParamEx**, **SetParam**, and **SetParamEx** methods.

Modifying Acquisition Parameters Individually

Acquisition parameters can be modified individually by using the **SetParam** and **SetParamEx** methods. When a new parameter value is requested, that value is verified against the current state of the acquisition hardware and its corresponding capability. If the modification request is denied because the parameter is dependent on other parameters, then all the parameters in question must be modified by group.

```
Dim Result as Boolean

' Using camera #2
Result = SapAcquisition1.SetParam(CORACQ_PRM_CAMSEL, 1, False)
```

For more details, see the reference material in this manual.

Modifying Acquisition Parameters by Group

Acquisition parameters can be modified by groups using the optional *UpdateNow* parameter to the **SetParam** and **SetParamEx** methods. When a new set of values is written, all modified parameters are verified against the given state and capabilities of the acquisition object.

```
Dim Left As Long
Dim Top As Long
Dim Width As Long
Dim Height As Long
Dim Result As Boolean

' Set a new cropping area for the acquisition device
Left = 40
Top = 80
Width = 160
Height = 80

' Parameter values only take effect after the fourth call to SetParam
Result = SapAcquisition1.SetParam(CORACQ_PRM_CROP_TOP, Top, False)
Result = SapAcquisition1.SetParam(CORACQ_PRM_CROP_LEFT, Left, False)
Result = SapAcquisition1.SetParam(CORACQ_PRM_CROP_WIDTH, Width, False)
Result = SapAcquisition1.SetParam(CORACQ_PRM_CROP_HEIGHT, Height, True)
```

For more details, see the reference material in this manual.

Using an Input Lookup Table

When you set the **Online** property to True in the SapAcquisition control, an internal lookup table (LUT) object (SapLut) is automatically created inside the object, if the acquisition hardware supports lookup tables.

You can then retrieve the LUT object with the **Lut** property, manipulate it using the methods of the SapLut sub-component, and reprogram it using the **ProgramLut** method.

The internal ILUT object is automatically destroyed when you set the **Offline** property to True, or when destroying the acquisition object.

```
Dim LutData
Dim Result As Boolean

Result = SapAcquisition1.Lut.GetDataArray(LutData)

(modify LUT data)

Result = SapAcquisition1.Lut.SetDataArray(LutData)
```

For more details, see the reference material in this manual.

Displaying Images

Enabling and Disabling Image Display

By default, a display window is automatically created in the SapAcquisition control. Images acquired from a camera, or loaded from disk, are shown in this window. To turn image display on or off, use the **Window** property.

```
' Stop displaying the acquisition control and its display window
SapAcquisition1.Display.Window = 0
SapAcquisition1.Visible = False

' Turn image display back on
SapAcquisition1.Visible = True
SapAcquisition1.Display.Window = 1
```

For more details, see the reference material in this manual.

Setting the Display Scaling Mode

By default, there is a one-to-one correspondence between image buffer data and pixels shown in the display area. You can control this mapping mode using the **Scaling** property.

```
' Displayed image buffer contents are scaled so that they are shown
' completely in the view area, while keeping the aspect ratio
SapAcquisition1.Display.Scaling = ScalingFitToWindow

' Restore the default scaling mode
SapAcquisition1.Display.Scaling = ScalingNone
```

For more details, see the reference material in this manual.

Working with Buffers

Using a Region of Interest

Although the SapAcquisition control automatically creates buffers with dimensions compatible with the acquisition device, it is possible to create an adjustable region of interest (ROI) inside each buffer. A ROI can be used by the acquisition to reduce bandwidth requirements, or by a processing function to process a specific region.

To define a rectangular area, you must create and initialize a SapRectangle object, assign it to the **ROI** property of the buffer sub-component of the acquisition object, then enable it with the **UseROI** property.

The example below illustrates this concept.

```
Dim NewROI As New SapRectangle

NewROI.Left = 120
NewROI.Top = 80
NewROI.Width = 240
NewROI.Height = 160

SapAcquisition1.Buffer.ROI = NewROI
SapAcquisition1.Buffer.UseROI = True
```

For more details, see the reference material in this manual.

Reading and Writing Buffers

The simplest way to read/write data from/to a buffer is to access it element by element. The **GetDataXY** and **SetDataXY** methods fulfill this purpose. The following example demonstrates how to modify data in an 8-bit monochrome buffer.

```
Dim Pixel As Long
Dim Result As Boolean
Dim BufData(100)

For Pixel = 1 To 100
    Result = SapAcquisition1.Buffer.GetDataXY(-1, Pixel, Pixel, BufData(Pixel))
    BufData(Pixel).Mono = BufData(Pixel).Mono + 64
    Result = SapAcquisition1.Buffer.SetDataXY(-1, Pixel, Pixel, BufData(Pixel))
Next Pixel
```

Accessing buffer data in this way is quite straightforward, but, unfortunately, it considerably slows down access time. Alternately, you can access data by reading/writing an array of elements with only one call to each of the **GetDataArray** and **SetDataArray** methods. Here is a sample of their usage.

```
Dim BufData
Dim Result As Boolean

Result = SapAcquisition1.Buffer.GetDataArray(-1, BufData)

(modify pixel data)

Result = SapAcquisition1.Buffer.SetDataArray(-1, BufData)
```

Although this is faster than the previous method, performance is still an issue because of the data copying operations involved.

The fastest way to access buffer data is to obtain direct access through a pointer. The **DataAddress** property provides direct access to buffer data. The drawback of this method is that you need the buffer dimensions, format, and pitch to correctly access the data. The following code illustrates this.

```
Dim I As Long
Dim Result As Boolean

' Use custom function to directly process all buffers
For I = 0 To SapAcquisition1.Buffer.Count - 1
    Result = MyProcessingFunc(SapAcquisition1.Buffer.DataAddress(I))
Next
```

For more details, see the reference material in this manual.

Acquisition Control

Overview

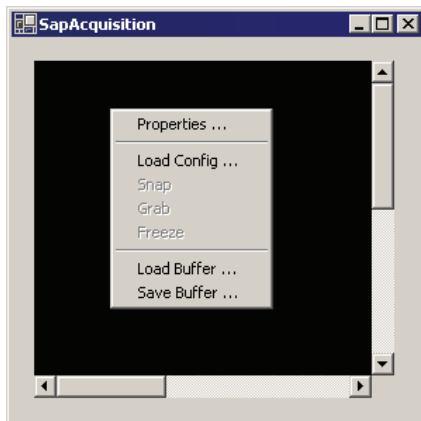
Sapera LT Acquisition Control (SapAcquisition) includes all the necessary functionality to acquire and display images. It can also perform other tasks, such as loading/saving images and management of acquisition LUTs. It includes the following components:

Name	Management of ...
SapAcquisition	image acquisition
SapBuffer	image buffers
SapDisplay	image display
SapLut	lookup tables
SapRectangle	rectangles
SapDataMono	monochrome data
SapDataRGB	RGB data
SapDataYUV	YUV data
SapDataHSI	HSI data
SapDataHSV	HSV data
SapDataFRGB	floating point RGB data

Property Pages

When a SapAcquisition control is inserted into a form, several property pages are available, allowing you to easily adjust the acquisition properties. To open the property pages, right-click on the SapAcquisition control in your form and select **Properties** in the pop-up menu.

These properties can also be set at runtime, by right-clicking on the acquisition display window and selecting **Properties** in the pop-up menu.



General Tab

The General tab allows you to specify the acquisition server, device, and configuration file. You can also specify to auto-display acquired image, use an acquisition LUT, flat-field correction, and the error reporting mode.

The screenshot shows a Windows-style dialog box titled "Properties" with a close button (X) in the top right corner. The dialog has several tabs: "General", "Events", "Transfer Settings", "Bayer Conversion", "Buffer", and "Display". The "General" tab is currently selected. Inside the dialog, the "Video Signal Status" is displayed as "N/A". Below this, there is a section with two text boxes: "Server" containing "Anaconda-CL_1" and "Device" containing "CameraLink Full Mono #1". Below these is "Serial Number: N/A" and a "Configuration File" text box containing "C:\DALSA Coreco\Sapera\CamFiles\User\CameraLink.ccf". There is a checkbox labeled "Online" which is unchecked, and a "Camera Selector" dropdown menu showing "0". Below these, there are three checkboxes: "Auto Display Acquired Images" (checked), "Use Acquisition LUT" (unchecked), and "Flat-Field Correction (Software Mode)" (unchecked). To the right of the "Use Acquisition LUT" checkbox is an "Error Reporting Mode" dropdown menu showing "Notify with popup". At the bottom right of the dialog are three buttons: "OK", "Cancel", and "Apply".

Properties

General | Events | Transfer Settings | Bayer Conversion | Buffer | Display

Video Signal Status: N/A

Server: Anaconda-CL_1 Device: CameraLink Full Mono #1

Serial Number: N/A

Configuration File: C:\DALSA Coreco\Sapera\CamFiles\User\CameraLink.ccf

☐ Online Camera Selector: 0

☒ Auto Display Acquired Images Error Reporting Mode: Notify with popup

☐ Use Acquisition LUT

☐ Flat-Field Correction (Software Mode)

OK Cancel Apply

Events Tab

The Events tab allows you to specify the type of acquisition event used to trigger an image capture, as well as the transfer events.

The screenshot shows a 'Properties' dialog box with the 'Events' tab selected. The dialog has a title bar with a close button. Below the title bar are five tabs: 'General', 'Events', 'Transfer Settings', 'Bayer Conversion', and 'Display'. The 'Events' tab is active, showing two sections: 'Acquisition Events' and 'Transfer Events'. The 'Acquisition Events' section contains eight checkboxes: 'Start of Frame', 'Start of Field', 'HSync Lock', 'Pixel Clock', 'End of Frame', 'End of Field', 'HSync Unlock', and 'No Pixel Clock'. The 'Transfer Events' section contains four checkboxes: 'Start of Frame', 'Start of Field', 'End of Transfer', and 'End of Field'. Below these sections are two more options: 'Enable Signal Status Event' and 'Device Reset Timeout (milliseconds)' with a text box containing '20000'. At the bottom of the dialog are three buttons: 'OK', 'Cancel', and 'Apply'.

Acquisition Events			
<input type="checkbox"/> Start of Frame	<input type="checkbox"/> Start of Field	<input type="checkbox"/> HSync Lock	<input type="checkbox"/> Pixel Clock
<input type="checkbox"/> End of Frame	<input type="checkbox"/> End of Field	<input type="checkbox"/> HSync Unlock	<input type="checkbox"/> No Pixel Clock
<input type="checkbox"/> Vertical Sync	<input type="checkbox"/> Data Overflow	<input type="checkbox"/> Frame Lost	
<input type="checkbox"/> External Trigger	<input type="checkbox"/> External Trigger Ignored		

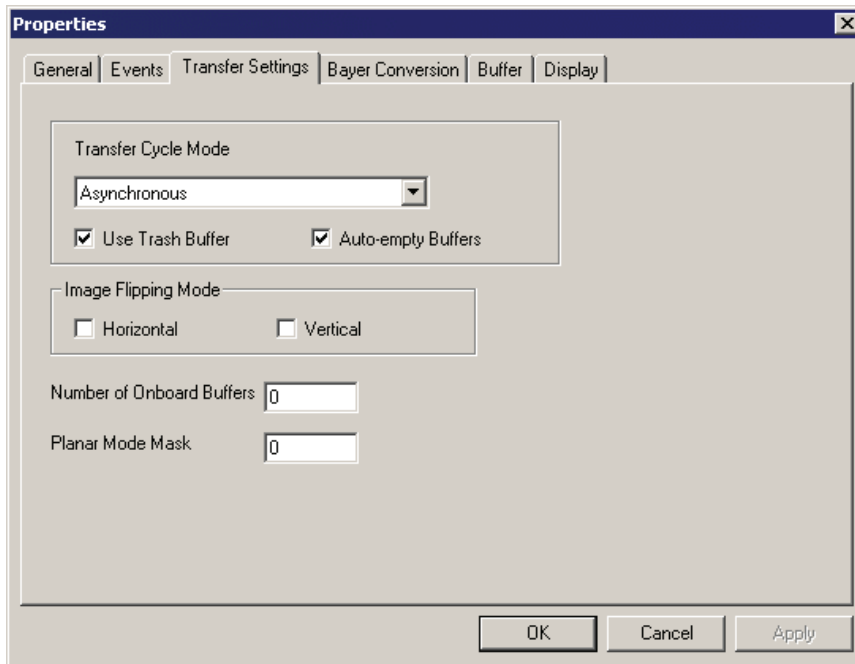
Transfer Events		
<input type="checkbox"/> Start of Frame	<input type="checkbox"/> Start of Field	<input type="checkbox"/> End of Transfer
<input type="checkbox"/> End of Frame	<input type="checkbox"/> End of Field	

☐ Enable Signal Status Event Device Reset Timeout (milliseconds)

OK Cancel Apply

Transfer Settings

The Transfer Settings tab allows you to specify the transfer cycle mode, image flipping mode, the number of onboard buffers, and the planar mode mask.



The screenshot shows a Windows-style dialog box titled "Properties" with a close button (X) in the top right corner. The dialog has several tabs: "General", "Events", "Transfer Settings" (which is selected), "Bayer Conversion", "Buffer", and "Display". The "Transfer Settings" tab contains the following controls:

- Transfer Cycle Mode:** A dropdown menu currently showing "Asynchronous".
- Use Trash Buffer:** A checked checkbox.
- Auto-empty Buffers:** A checked checkbox.
- Image Flipping Mode:** A group box containing two unchecked checkboxes: "Horizontal" and "Vertical".
- Number of Onboard Buffers:** A text input field containing the value "0".
- Planar Mode Mask:** A text input field containing the value "0".

At the bottom right of the dialog are three buttons: "OK", "Cancel", and "Apply".

Bayer Conversion

The Bayer Conversion tab allows you to enable Bayer conversion and to specify the type of alignment, white balance coefficients, and conversion method.

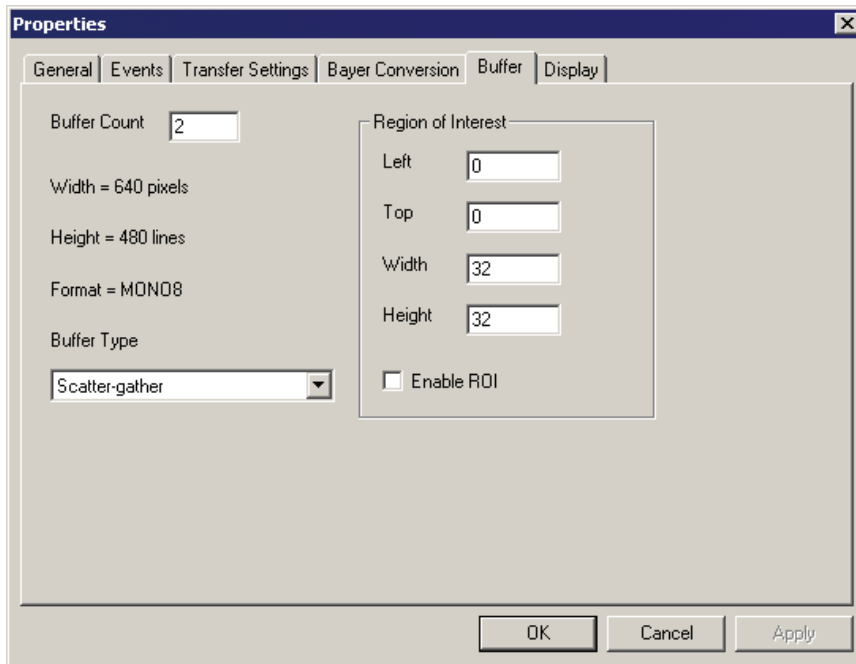
The screenshot shows a 'Properties' dialog box with the 'Bayer Conversion' tab selected. The dialog has a title bar with a close button. Below the title bar are tabs for 'General', 'Events', 'Transfer Settings', 'Bayer Conversion', 'Buffer', and 'Display'. The 'Bayer Conversion' tab contains the following settings:

- ☐ Enable Bayer Conversion (Software Mode)
- Alignment:** A group box containing four radio buttons and corresponding color patches (Red, Green, Blue, and a combined patch). The bottom radio button is selected.
- White Balance Coefficients:** A group box containing two columns of settings: 'Gain' and 'Offset'. Each column has input fields for 'Red', 'Green', and 'Blue'. The 'Gain' values are all '1', and the 'Offset' values are all '0'.
- Conversion Method:** A dropdown menu currently showing 'Method 1'.
- ☐ Use Bayer LUT

At the bottom of the dialog are three buttons: 'OK', 'Cancel', and 'Apply'.

Buffer Tab

The Buffer tab allows you to specify the number of acquisition buffers, their type, and whether to use an ROI.



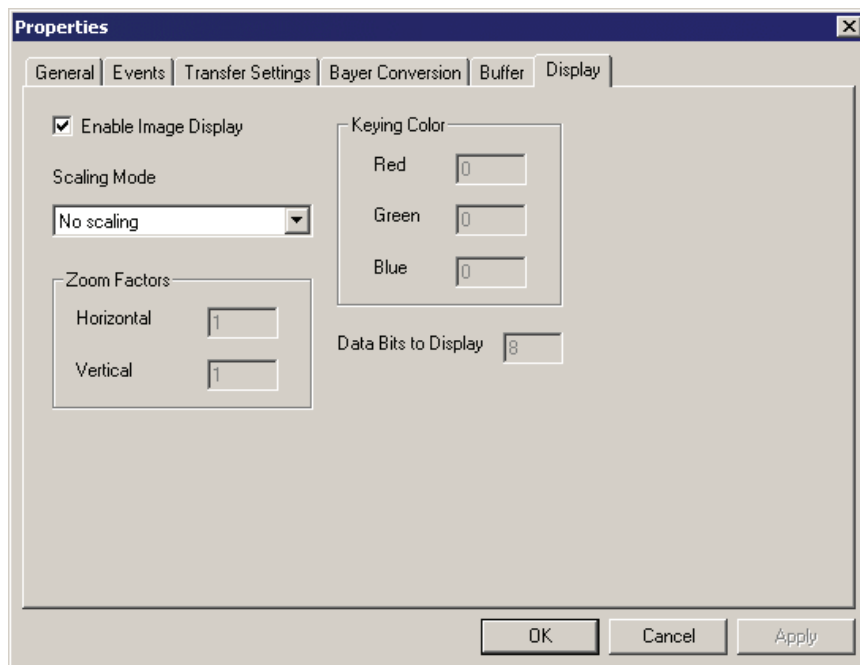
The image shows a 'Properties' dialog box with the 'Buffer' tab selected. The dialog has a title bar with a close button (X). The tabs are General, Events, Transfer Settings, Bayer Conversion, Buffer, and Display. The Buffer tab contains the following settings:

- Buffer Count: 2
- Width = 640 pixels
- Height = 480 lines
- Format = MONO8
- Buffer Type: Scatter-gather (dropdown menu)
- Region of Interest (ROI) section:
 - Left: 0
 - Top: 0
 - Width: 32
 - Height: 32
 - ☐ Enable ROI

At the bottom of the dialog are three buttons: OK, Cancel, and Apply.

Display Tab

The Display tab allows you to specify the scaling mode, zoom factors, keying color, and the data bits to display.



SapAcquisition

SapAcquisition Member List

Properties	Description
Valid	Current state of object (read only)
SapControlType	Type of Sapera LT control = acquisition (read only)
Online	Availability of image acquisition (runtime)
NumServers	Number of available acquisition servers (read only)
AllServers	Names of all acquisition servers (read only, runtime)
Server	Acquisition server name
NumDevices	Number of available devices on server (read only)
AllDevices	Names of all devices on server (read only, runtime)
Device	Acquisition device name (relative to server)
SerialNumber	Acquisition device serial number
ConfigFile	Acquisition configuration file name (CCF)
Label	Descriptive text label (read only)
CamSel	Camera selector
AcqEventType	Acquisition event type
XferEventType	Transfer event type
AutoDisplay	Automatically displays acquired images
AutoEmpty	Automatic emptying of buffer state after acquisition
HorzFlip	Horizontal flip enable
VertFlip	Vertical flip enable
Planar	Planar acquisition mask
SignalStatus	Current signal status (runtime)
SignalNotify	Enables signal status events
SerialName	Serial port name on current server (read only, runtime)
FlatField	Flat-field correction enable
FlatFieldMode	Flat-field operation mode: hardware or software (read only)
OnBoardBuffers	Number of internal buffers on frame grabbers (used when acquiring images)
Bayer	Bayer conversion enable: off, white balance + convert, and convert
BayerLut	Lookup table for Bayer conversion
BayerLutEnable	Enable/disable Bayer lookup table
BayerAlign	Bayer alignment mode

BayerMethod	Bayer pixel value calculation method
BayerMode	Bayer conversion mode: hardware or software
BayerWBGain	Bayer white balance gain coefficients
BayerWBOffset	Bayer white balance offset coefficients
LutEnable	Enable/disable acquisition lookup table
Grabbing	Grab active flag (runtime)
HasTrash	Presence of a trash buffer
Cycle	Cycle mode for destination buffer object
ErrorMode	Error reporting mode
ResetTimeout	Timeout for device reset
LastErrorValue	Numeric value of most recent error (read only, runtime)
LastErrorDesc	Description of most recent error (read only, runtime)
AllowPropertyDlg	Allow display of property dialog
Buffer	Buffer component
Display	Display component
Lut	Lookup table component
BayerBuffer	Bayer conversion result buffer
FlatFieldGain	Flat-field correction gain buffer
FlatFieldOffset	Flat-field correction offset buffer

Methods	Description
---------	-------------

ShowProperties	Shows property dialog for acquisition component and sub-components
LoadConfigDlg	Interactively chooses the acquisition server, device, and configuration file
CompositeDlg	Interactively performs composite video adjustments (contrast, brightness, ...)
BayerDlg	Interactively adjusts Bayer conversion settings
FlatFieldDlg	Interactively calibrates flat-field correction settings
SaveConfig	Saves current acquisition parameters to configuration (CCF) file
LoadFlatField	Loads the flat-field gain and offset buffers from disk
SaveFlatField	Saves the flat-field gain and offset buffers to disk
ProgramLut	Reprograms acquisition lookup table from internal LUT object
SoftTrigger	Software trigger for acquisition
GetCap	Gets value of any low-level acquisition or transfer capability (4 bytes)
GetCapEx	Gets value of any low-level acquisition or transfer capability (more than 4 bytes)
GetParam	Gets value of any low-level acquisition or transfer parameter (4 bytes)
GetParamEx	Gets value of any low-level acquisition or transfer parameter (more than 4 bytes)
SetParam	Sets value of any low-level acquisition or transfer parameter (4 bytes) with possibility of delayed update

SetParamEx	Sets value of any low-level acquisition or transfer parameter (more than 4 bytes) with possibility of delayed update
Snap	Acquires one or more images
Grab	Starts continuous grab
Freeze	Stops continuous grab
Abort	Aborts continuous grab
Wait	Waits for end of grab (after Snap or Freeze), with an optional abort dialog
Message	Reports a message using the current reporting mode
Reset	Resets the current acquisition server
Events	Description
Acq	Acquisition events
Xfer	Transfers events (including trash buffer)
Signal	Notification of signal status changes
EndReset	End of server reset
SapError	Error event

SapAcquisition Member Description

Abort Method

Aborts continuous grab

object.**Abort()** As Boolean

Description

Stops acquisition immediately without waiting for the current frame to be completely transferred.

You should call Abort only for emergencies. For example, calling Wait after the Snap or Grab methods may fail because of a timeout condition (usually hardware related). In this case, using Abort is often the only way to correct the situation.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result as Boolean
Result = SapAcquisition1.Abort
```

Acq Event

Acquisition events

Object **_Acq**(*eventType* As enumAcqEvent, *eventCount* As Long)

Parameters

eventType Event type for which the acquisition event is triggered.

eventCount Current count of acquisition events. The initial value is 1 and increments after every event.

Description

Acquisition events are generated each time one or more of the conditions registered using the AcqEventType property are encountered. If two or more of these conditions happen simultaneously, then they are reported as such, with the *eventType* argument containing a binary OR combination of their values. Setting the AcqEventType property to the special value AcqEventNone disables all acquisition events completely.

Note that acquisition events are only available on certain boards and are usually not required. They must not be confused with the much more common transfer event mechanism. See the Xfer event for further information.

See the AcqEventType property for a list of possible values.

Example

```
SapAcquisition1.AcqEventType = AcqEventFrameLost
```

(other code)

```
Private Sub SapAcquisition1_Acq(ByVal eventType As SapAcquisitionLIB.enumAcqEvent,
ByVal eventCount As Long)
    EventText.Caption = "Acquisition Event(" & eventType & ", " & eventCount & ")"
End Sub
```

AcqEventType Property

AcqAcquisition event type

object.**AcqEventType** As enumAcqEvent

Description

Specifies which types are registered for acquisition events. To register more than one type, combine together the values using a bitwise OR operation. The following values are supported:

AcqEventNone	Disables acquisition events
AcqEventStartOffFrame	Start of frame
AcqEventEndOffFrame	End of frame
AcqEventVirtualFrame	Equivalent to AcqEventStartOffFrame for linescan cameras
AcqEventExternalTrigger	Received an external trigger that will then acquire at least one image. The maximum callback rate cannot be greater than the acquisition video frame rate.
AcqEventVerticalSync	Vertical sync detected, even if not acquiring.
AcqEventNoPixelClk	No pixel clock detected. Generated only once, unless the Snap or Grab method is called or the pixel clock is detected again and then lost.

AcqEventPixelClk	Pixel clock detected. Generated only one time, unless a new Snap or Grab method is called or the pixel clock is lost again and then detected.
AcqEventFrameLost	Lost a frame during live acquisition. This usually occurs if there is not enough bandwidth to transfer images to host memory.
AcqEventDataOverflow	Data overflow occurred during live acquisition. This usually occurs if the acquisition device cannot sustain the data rate of incoming images.
AcqEventExternalTriggerIgnored	Dropped an external trigger event. This usually occurs when the external trigger rate is faster then the acquisition frame rate.
AcqEventExternalTriggerTooSlow	The detected external trigger rate is too slow for the hardware to process. This can usually occur when using the shaft encoder multiplier.
AcqEventHsyncLock	Detected a horizontal sync unlock to lock condition.
AcqEventHsyncUnlock	Detected a horizontal sync lock to unlock condition.

The initial value for this property is AcqEventNone, unless you specify another value at design-time.

This property is read only during live grab, that is, when the Grabbing property is True.

Example

```
SapAcquisition1.AcqEventType = AcqEventFrameLost

(other code)

Private Sub SapAcquisition1_Acq(ByVal eventType As SapAcquisitionLIB.enumAcqEvent,
ByVal eventCount As Long)
    EventText.Caption = "Acquisition Event(" & eventType & "," & eventCount & ")"
End Sub
```

AllDevices Property

Names of all devices on server (read only)

object.**AllDevices** As Variant

Description

Returns the names of all available acquisition devices for the server currently selected using the Server property, as a list of text strings, one for each device. For example:

- Standard Composite Video & Y/C
- Monochrome Interface
- RGB Interface

This property is only available at runtime.

Example

```
Dim DeviceName
Dim Index As Long

Index = 0

If SapAcquisition1.NumDevices > 0 Then
    For Each DeviceName In SapAcquisition1.AllDevices
        DevicesText(Index).Caption = DeviceName
        Index = Index + 1
    Next
End If
```

AllowPropertyDlg Property

Allow display of property dialog

object.**AllowPropertyDlg** As Boolean

Description

Specifies whether the property dialog can be invoked at run-time, either by calling the ShowProperties method, or by right-clicking the SapAcquisition control display window. When this property is set to False, then ShowProperties returns an error, and the mouse right-click behavior is disabled.

The initial value for this property is True, unless you specify another value at design-time.

Example

```
SapAcquisition1.AllowPropertyDlg = False
```

AllServers Property

Names of all acquisition servers (read only)

object.**AllServers** As Variant

Description

Returns the names of all available acquisition servers, as a list of text strings, one for each server. For example:

```
-Bandit_II_1  
-Viper_Quad_1
```

This property is only available at runtime.

Example

```
Dim ServerName  
Dim Index As Long  
  
Index = 0  
  
If SapAcquisition1.NumServers > 0 Then  
    For Each ServerName In SapAcquisition1.AllServers  
        ServersText(Index).Caption = ServerName  
        Index = Index + 1  
    Next  
End If
```

AutoDisplay Property

Automatically displays acquired images

object.**AutoDisplay** As Boolean

Description

Specifies whether acquired images are shown automatically using the display mechanism built into the ActiveX control. When set to False, you need to explicitly call the Show method of the SapDisplay component to display an acquired or processed image.

The initial value for this property is True, unless you specify another value at design-time.

Example

```
SapAcquisition1.AutoDisplay = False
```

AutoEmpty Property

Automatic empty of buffer state after acquisition

object.**AutoEmpty** As Boolean

Description

Manages the 'auto empty' mechanism, used for synchronizing the transfer with the processing task in the application program.

By default, buffers are automatically set to the empty state after an image has been acquired into them. This means that a new image could be acquired in the same buffer before the processing task can even use it.

In this case, you should set this property to 'False' to disable this behavior. You must then set the State property of the SapBuffer component at the end of the processing task.

The initial value for this property is True, unless you specify another value at design-time.

Example

```
SapAcquisition1.AutoEmpty = False  
  
(Processing task)  
SapAcquisition1.Buffer.State(BufferIndex) = BufferStateEmpty
```

Bayer Property

Bayer conversion enable

object.**Bayer** As enumBayer

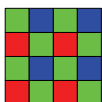
Description

Specifies whether Bayer conversion is enabled or disabled during acquisition, using one of the following values:

BayerOff	Bayer conversion disabled
BayerConvertOnly	Bayer conversion enabled

The Bayer format assigns each pixel in a monochrome image the value of one color channel. RGB images are created by using neighboring pixel values to get the two missing color channels for each pixel.

Pixels in one row of a Bayer image alternate between the green channel value and either the red or the blue channel value. The default scheme is shown below.



The missing color channel values are determined using neighboring pixel values for the color channel in question. This is done by various methods, some of which are more computationally expensive, but give better image quality when the input image contains many strong edges. See the BayerMethod property.

Use the BayerMode property to determine if hardware correction is available in the acquisition device.

The initial value for this property is BayerOff, unless you specify another value at design-time.

This property is read only during live grab, that is, when the Grabbing property is True.

Example

```
SapAcquisition1.Bayer = BayerConvertOnly
```





BayerAlign Property

Bayer alignment mode

Object **BayerAlign** As enumBayerAlign

Description

Specifies the current Bayer alignment mode, which must correspond to the upper left 2x2 square of the Bayer scheme of your camera. The following values are supported:

BayerAlignGBRG	
BayerAlignBGGR	
BayerAlignRGGB	
BayerAlignGRBG	

To enable or disable Bayer conversion, see the Bayer property.

The initial value for this property is BayerAlignGRBG, unless you specify another value at design-time.

This property is read only during live grab, that is, when the Grab property is True.

Example

```
BayerAlignText.Caption = SapAcquisition1.BayerAlign
```

BayerBuffer Property

Bayer conversion result buffer

object.BayerBuffer As SapBuffer

Description

Gives direct access to the RGB buffer object used as the destination for software Bayer conversion.

When Bayer conversion is performed in hardware (through the Bayer and BayerMode properties), this property refers to the same object as the Buffer property.

Example

```
Dim Result As Boolean  
Result = SapAcquisition1.BayerBuffer.SaveWithDlg(hWnd, 0, False)
```

BayerDlg Method

Interactively adjusts Bayer conversion settings

Object.**BayerDlg**(*parentWnd* As OLE_HANDLE) As Boolean

Parameters

parentWnd Parent window for the dialog box (0 if no parent window)

Description

Displays a dialog box that allows you to dynamically adjust the following settings related to Bayer conversion: alignment, pixel value calculation method, Gamma factor for Bayer lookup table, and white balance gain coefficients (manually or automatically).

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result as Boolean
Result = SapAcquisition1.BayerDlg(hWnd)
```

BayerLut Property

LUT for Bayer conversion

object.**BayerLut** As SapLut

Description

Gives direct access to the Bayer conversion LUT that is automatically created inside the control using the data format and pixel depth of the current acquisition device.

To enable or disable this LUT, use the BayerLutEnable property.

This property is read only during live grab, that is, when the Grabbing property is True.

Example

```
Dim LutFormat as enumSapFormat
LutFormat = SapAcquisition1.BayerLut.Format
```

BayerLutEnable Property

Enable/disable Bayer LUT

object.**BayerLutEnable** As Boolean

Description

Specifies whether the Bayer conversion LUT is enabled or disabled during acquisition. You can use it to apply color correction after the filtering (for example, Gamma correction).

To directly access the LUT, use the BayerLut property.

The initial value for this property is False, unless you specify another value at design-time.

This property is read only during live grab, that is, when the Grabbing property is True.

Example

```
SapAcquisition1.BayerLutEnable = True
```

BayerMethod Property

Bayer pixel value calculation method

object.**BayerMethod** As enumBayerMethod

Description

Specifies the current Bayer pixel value calculation method using one of the following values:

BayerMethod1	Technique based on bilinear interpolation. Fast, but tends to smooth the edges of the image.
BayerMethod2	Advanced technique, better for preserving the edges of the image. However, it works well only when the image has a strong content in green; otherwise, little amounts of noise may be visible within objects.
BayerMethod3	Advanced technique, almost as good as Method2 for preserving the edges, but independent of the image content in green. Small color artefacts of 1 pixel may be visible at the edges.

To enable or disable Bayer conversion, see the Bayer property.

The initial value for this property is BayerMethod1, unless you specify another value at design-time.

This property is read only during live grab, that is, when the Grabbing property is True.

Example

```
SapAcquisition1.BayerMethod = BayerMethodGoldenEagle
```

BayerMode Property

Bayer conversion mode (read only)

object.**BayerMode** As enumBayerMode

Description

Specifies the current Bayer conversion mode using one of the following values:

BayerHardware	Bayer conversion is performed in hardware
BayerSoftware	Bayer conversion is performed in software

The value of this property is set automatically by the control, depending on the availability of hardware Bayer conversion.

To enable or disable Bayer conversion, see the Bayer property.

Example

```
BayerText.Caption = SapAcquisition1.BayerMode
```

BayerWBGain Property

Bayer white balance gain coefficients

object.**BayerWBGain** As SapDataFRGB

Description

Specifies the Bayer white balance gain coefficients. These can be set manually or calculated automatically when the Bayer property is set to BayerWbAndConvert.

The white balance gain coefficients are the red, green, and blue gains applied to the input image before filtering. These are used to balance the three color components so that a pure white at the input gives a pure white at the output. Set all gains to 1.0 if no white balance gain is required.

The initial value for this property is 1.0 for each color component, unless you specify another value at design-time.

This property is read only during live grab, that is, when the Grabbing property is True.

Example

```
Dim NewWBGain As New SapDataRGB

NewWBGain.Red = 0.9
NewWBGain.Green = 1.0
NewWBGain.Blue = 1.1

SapAcquisition1.BayerWBGain = NewWBGain
SapAcquisition1.BayerMode = BayerWbAndConvert
```

BayerWBOffset Property

Bayer white balance offset coefficients

object.**BayerWBOffset** As SapDataFRGB

Description

Specifies the Bayer white balance offset coefficients. These apply only for hardware conversion, that is, when the value of the Bayer property is BayerConvertOnly.

The white balance offset coefficients are the red, green, and blue offsets applied to the input image before filtering. These are used to balance the three color components so that a pure white at the input gives a pure white at the output. Set all offsets to 0.0 if no white balance offset is required.

The initial value for this property is 0.0 for each color component, unless you specify another value at design-time.

This property is read only during live grab, that is, when the Grabbing property is True.

Example

```
Dim NewWBOffset As New SapDataRGB

NewWBOffset.Red = 0.05
NewWBOffset.Green = 0.1
NewWBOffset.Blue = 0.25

SapAcquisition1.BayerWBOffset = NewWBOffset
SapAcquisition1.BayerMode = BayerConvertOnly
```

Buffer Property

Buffer sub-component

object.**Buffer** As SapBuffer

Description

Gives direct access to the internal buffer sub-component. Image buffers are automatically created inside the control to match the data format and size of images grabbed from the current acquisition device.

When Bayer conversion is enabled and performed in hardware (through the Bayer and BayerMode properties), then this property represents the RGB buffer with the result of the conversion.

Example

```
Dim NumBuffers as Long
NumBuffers = SapAcquisition1.Buffer.Count
```

CamSel Property

Camera selector

object.**CamSel** As Long

Description

Specifies the zero-based index of the camera input from which the acquisition device grabs images. The maximum value allowed depends on the acquisition hardware and the current data format.

The initial value for this property is 0, unless you specify another value at design-time.

This property is read only during live grab, that is, when the Grabbing property is True.

Example

```
SapAcquisition1.CamSel = 1
```

CompositeDlg Method

Interactively performs composite video adjustments

Object.**CompositeDlg**(*parentWnd* As OLE_HANDLE) As Boolean

Parameters

parentWnd Parent window for the dialog box (0 if no parent window)

Description

Displays a dialog box that allows you to dynamically adjust the following acquisition settings related to composite video input signals: brightness/contrast, hue/saturation, and sharpness.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result as Boolean  
Result = SapAcquisition1.CompositeDlg(hWnd)
```

ConfigFile Property

Acquisition configuration file name

object.**ConfigFile** As String

Description

Name of the acquisition configuration file (CCF) that describes all camera and frame grabber-related acquisition parameters. Use one of the standard CCF files provided with Sapera LT or create one using the CamExpert utility.

The initial value for this property is an empty string, unless you specify another value at design-time.

This property is read only when the Online property is True.

Example

```
ConfigFileText.Caption = SapAcquisition1.ConfigFile
```

Cycle Property

Cycle mode for destination buffer object

object.**Cycle** As enumCycleMode

Description

Specifies the destination buffer cycling mode for image acquisition.

The available modes differ by the way in which they specify which image buffer gets the next data transfer.

The empty state refers to the scenario where buffer data has been completely processed and can be overwritten. It is set by application code as soon as it has finished processing buffer data.

The full state refers to the scenario where buffer data has not been processed since its latest data transfer. It is set by the transfer device as soon as a data transfer has completed.

The current buffer is the one in which the latest data transfer occurred.

The next buffer is the one immediately after the current buffer, with wraparound to the first buffer at the end of the list.

The trash buffer is defined as the last buffer in the list for the WithTrash modes only. Its state is always considered to be empty by the transfer device.

The Cycle property can have one of the following values:

CycleUnknown	Unknown cycle mode.
CycleAsynchronous	Always transfer to the next buffer, regardless of its state.
CycleSynchronous	If next buffer is empty, then transfer to next buffer; otherwise transfer to current buffer.
CycleWithTrash	If next buffer is empty, then transfer to the next buffer; otherwise transfer to the trash buffer. Repeat transferring to the trash buffer as long as the next buffer is full.
CycleOff	Always transfer to the current buffer.
CycleNextEmpty	If next buffer is empty, then transfer to next buffer; otherwise transfer to next empty buffer in the list. If all buffers are full, then

	transfer to current buffer.
CycleNextWithTrash	If next buffer is empty, then transfer to next buffer; otherwise transfer to next empty buffer in the list. If all buffers are full, then transfer to trash buffer. Repeat transferring to the trash buffer as long as there is no empty buffer in the list.

The initial value for this property is CycleUnknown, unless you specify another value at design-time.

If its value is CycleUnknown and the HasTrash property is True, then it is automatically changed to CycleWithTrash when the Online property is True; otherwise, it is set to CycleAsynchronous.

This property is read only during live grab, that is, when the Grabbing property is True.

Example

```
CycleText.Caption = SapAcquisition1.Cycle
```

Device Property

Acquisition device name

object.**Device** As String

Description

Specifies the name of the current acquisition device for the server currently selected using the Server property.

You can also use the LoadConfigDlg method to interactively chose the server, device, and configuration file (CCF) for the acquisition.

The initial value for this property is am empty string, unless you specify another value at design-time.

This property is read only when the Online property is True.

Example

```
SapAcquisition1.Server = "Bandit_II_1"
SapAcquisition1.Device = "Monochrome Interface"
```

Display Property

Display sub-component

object.**Display** As SapDisplay

Description

Gives direct access to the internal display sub-component, which is automatically created inside the control.

Example

```
Dim ScalingMode as enumViewScaling
ScalingMode = SapAcquisition1.Display.Scaling
```

EndReset Event

End of server reset

Object **EndReset**(*server* As String)

Parameters

server Name of affected server

Description

After calling the Reset method with the *notify* argument set to True, this event will be fired as soon as the reset operation is complete.

Example

```
Dim Result As Boolean
Result = SapAcquisition1.Reset(True)

(other code)

Private Sub SapAcquisition1_EndReset(Server As String)
    MsgBox("Received end of reset event for " & Server)
End Sub
```

ErrorMode Property

Error reporting mode

Object **ErrorMode** As enumErrorMode

Description

Specifies the global reporting mode for messages and errors, as one of the following values:

ErrorModeNotify	Sends messages to a popup window
ErrorModeLog	Sends messages to the DALSA Log Viewer
ErrorModeDebug	Sends messages to the active debugger, if any
ErrorModeEvent	Messages are reported through an event
ErrorModeCustom	Messages are not sent anywhere; they are just stored internally

This mode is used by the built-in error reporting mechanism. It is also used automatically when calling the Message method.

For the event reporting mode, a SapError event is fired every time an error occurs.

For custom reporting mode, the only way to retrieve messages is by reading the LastErrorValue or LastErrorDesc properties.

The initial value for this property is ErrorModeNotify, unless you specify another value at design-time.

Example

```
Dim Result As Boolean

SapAcquisition1.ErrorMode = ErrorModeEvent
Result = SapAcquisition1.Message("This triggers the SapError event")
```

```
(other code)
```

```
Private Sub SapAcquisition1_SapError(ByVal Value As Long, description As String)
    MsgBox("Sapera error event, description = " & Chr(13) & description)
End Sub
```

FlatField Property

Flat-field correction enabled

object.**FlatField** As Boolean

Description

Specifies whether flat-field correction is enabled or disabled during acquisition.

Flat-field correction compensates for uneven lighting conditions in acquired images by modifying pixel data using specific gain and offset values at each location. The FlatFieldDlg method interactively initializes buffers containing these values. It is also possible to call the LoadFlatField method to load them from disk files instead.

The FlatFieldMode property specifies whether flat-field correction is performed in hardware or software.

Buffers with the gain and offset values are available through the FlatFieldGain and FlatFieldOffset properties.

The initial value for this property is False, unless you specify another value at design-time.

This property is read only during live grab, that is, when the Grabbing property is True.

Example

```
SapAcquisition1.FlatField = True
```

FlatFieldDlg Method

Interactively calibrates flat-field correction settings

Object.**FlatFieldDlg**(*parentWnd* As OLE_HANDLE) As Boolean

Parameters

parentWnd Parent window for the dialog box (0 if no parent window)

Description

Displays a dialog box that allows you to interactively calibrate flat-field correction settings. You are asked to acquire both a black and a white reference image, following which the gain and offset values for each pixel are automatically calculated and stored in their respective buffers.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result as Boolean
Result = SapAcquisition1.FlatFieldDlg(hWnd)
```

FlatFieldGain Property

Flat-field correction gain buffer

object.**FlatFieldGain** As SapBuffer

Description

Gives direct access to a buffer containing the gain parameters for flat-field correction. This buffer is automatically created inside the control to match the data format and size of images grabbed from the current acquisition device.

You can use the FlatFieldDlg method to interactively initialize the buffer values. It is also possible to call the LoadFlatField method to load them from a disk file instead.

Example

```
Dim GainWidth as Long  
GainWidth = SapAcquisition1.FlatFieldGain.Width
```

FlatFieldMode Property

Flat-field conversion mode (read only)

object.**FlatFieldMode** As enumFlatFieldMode

Description

Specifies the current flat-field correction mode, using one of the following values:

FlatFieldHardware	Flat-field correction is performed in hardware
FlatFieldSoftware	Flat-field correction is performed in software

The value of this property is set automatically by the control, depending on the availability of hardware flat-field correction.

To enable or disable flat-field correction, see the FlatField property.

Example

```
FlatFieldText.Caption = SapAcquisition1.FlatFieldMode
```

FlatFieldOffset Property

Flat-field correction offset buffer

object.**FlatFieldOffset** As SapBuffer

Description

Gives direct access to a buffer containing the offset parameters for flat-field correction. This buffer is automatically created inside the control to match the data format and size of images grabbed from the current acquisition device.

You can use the FlatFieldDlg method to interactively initialize the buffer values. It is also possible to call the LoadFlatField method to load them from a disk file instead.

Example

```
Dim OffsetWidth as Long  
OffsetWidth = SapAcquisition1.FlatFieldOffset.Width
```

Freeze Method

Stops continuous grab

object.**Freeze()** As Boolean

Description

Issues a stop request for the current acquisition (started by the Grab method). The actual data transfer will end only after the current frame is completely transferred, so you should call the Wait method immediately after Freeze to ensure correct synchronization.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result as Boolean  
Result = SapAcquisition1.Freeze  
If Result = True Then  
    Result = SapAcquisition1.Wait(5000, False)  
End If
```

GetCap Method

Gets value of any low-level acquisition or transfer capability (4 bytes)

Object.**GetCap**(*cap* As Long, *value* As Long) As Boolean

Parameters

cap Low-level Sopera capability to read
value Contains the capability value when the method returns

Description

This method allows direct read access to low-level Sopera capabilities for the acquisition or transfer modules. When the capability value requires more than 4 bytes of storage, use the GetCapEx method instead.

Note that GetCap is rarely required. The acquisition control already uses important capabilities internally for self-configuration and validation.

See the *Sopera Acquisition Parameters Reference Manual* and the *Sopera Basic Modules Reference Manual* for a description of all capabilities and their possible values.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Value As Long  
Dim Result As Boolean  
Result = SapAcquisition1.GetCap(CORACQ_CAP_SIGNAL_STATUS, Value)
```

GetCapEx Method

Gets value of any low-level acquisition or transfer capability

Object.**GetCapEx**(*cap* As Long, *value* As Variant) As Boolean

Parameters

cap Low-level Sopera capability to read
value Contains the capability value when the method returns

Description

This method allows direct read access to low-level Sopera capabilities for the acquisition or transfer modules. If the capability value requires 4 bytes of storage, use the GetCap method instead.

Note that GetCapEx is rarely required. The acquisition control already uses important capabilities internally for self-configuration and validation.

See the *Sopera Acquisition Parameters Reference Manual* and the *Sopera Basic Modules Reference Manual* for a description of all capabilities and their possible values.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Value
Dim FormatItem
Dim Result As Boolean

Result = SapAcquisition1.GetCapEx(CORACQ_CAP_OUTPUT_FORMAT, Value)

CapText.Caption = "Capability value = "
For Each FormatItem In Value
    CapText.Caption = CapText.Caption & FormatItem & " "
Next
```

GetParam Method

Gets value of any low-level acquisition or transfer parameter (4 bytes)

Object.**GetParam**(*param* As Long, *value* As Long) As Boolean

Parameters

param Low-level Sopera parameter to read
value Contains the parameter value when the method returns

Description

This method allows direct read access to low-level Sopera parameters for the acquisition or transfer modules. When the parameter value requires more than 4 bytes of storage, use the GetParamEx method instead.

Note that GetParam is rarely needed. The acquisition control already uses important parameters internally for self-configuration and validation.

See the *Sopera Acquisition Parameters Reference Manual* and the *Sopera Basic Modules Reference Manual* for a description of all parameters and their possible values.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Value As Long
Dim Result As Boolean
Result = SapAcquisition1.GetParam(CORACQ_PRM_SIGNAL_STATUS, Value)
```

GetParamEx Method

Gets value of any low-level acquisition or transfer parameter

Object.**GetParamEx**(*param* As Long, *value* As Variant) As Boolean

Parameters

param Low-level Sopera parameter to read
value Contains the parameter value when the method returns

Description

This method allows direct read access to low-level Sopera parameters for the acquisition or transfer modules. If the parameter value requires 4 bytes of storage, use the GetParam method instead.

Note that GetParamEx is rarely required. The acquisition control already uses important parameters internally for self-configuration and validation.

See the *Sapera Acquisition Parameters Reference Manual* and the *Sapera Basic Modules Reference Manual* for a description of all parameters and their possible values.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Value
Dim Result As Boolean

Result = SapAcquisition1.GetParamEx(CORACQ_PRM_LABEL, Value)
ParamText.Caption = Value
```

Grab Method

Starts continuous grab

object.**Grab**() As Boolean

Description

Starts continuous acquisition.

Note that this is always done asynchronously, that is, no explicit checking is performed to verify if acquisition is already in progress. If you want to perform this check, you must first call the Wait method.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result as Boolean
Result = SapAcquisition1.Grab
```

Grabbing Property

Grab active flag (read only)

object.**Grabbing** As Boolean

Description

Indicates continuous acquisition is active. Use the Grab method to initiate continuous acquisition.

This property is only available at runtime.

Example

```
Dim IsGrabbing as Long
IsGrabbing = SapAcquisition1.Grabbing
```

HasTrash Property

Presence of a trash buffer

object.**HasTrash** As Boolean

Description

Indicates whether the acquisition buffers include an additional trash buffer. The latter is used automatically whenever acquisition cannot be performed in the other buffers.

Changing the value of HasTrash can automatically change the Cycle property inside the acquisition control.

The initial value for this property is True, unless you specify another value at design-time.

This property is read only when the Online property is True.

Example

```
SapAcquisition1.HasTrash = True
```

HorzFlip Property

Horizontal flip enable

object.**HorzFlip** As Boolean

Description

Indicates whether horizontal flipping (mirroring) mode for acquired images is active.

The initial value for this property is False, unless you specify another value at design-time.

Example

```
SapAcquisition1.HorzFlip = True
```

Label Property

Descriptive text label (read only)

object.**Label** As String

Description

Returns a complete text description of the current acquisition device. The string is composed of a combination of the Server and Device properties, as follows:

Server [Device]

The value of this property is only relevant when the Online property is True.

Example

```
LabelText.Caption = SapAcquisition1.Label
```

LastErrorDesc Property

Description of most recent error (read only)

object.**LastErrorDesc** As String

Description

Gets a text description of the latest acquisition control or low-level Sapera error.

See the **ErrorMode** property for a description of the available error modes.

See the *Sapera Basic Modules Reference Manual* for details on low-level Sapera functionality.

This property is only available at runtime.

Example

```
Dim Result as Boolean
Result = SapAcquisition1.Grab

If Result = False Then
    ErrorText.Caption = SapAcquisition1.LastErrorDesc
End If
```

LastErrorValue Property

Numeric value of most recent error (read only)

object.**LastErrorValue** As Long

Description

Gets the numeric value of the latest acquisition control or low-level Sapera error.

See the **ErrorMode** property for a description of the available error modes.

See the *Sapera Basic Modules Reference Manual* for details on low-level Sapera functionality.

This property is only available at runtime.

Example

```
Dim Result as Boolean
Result = SapAcquisition1.Grab

If Result = False Then
    ErrorText.Caption = SapAcquisition1.LastErrorValue
End If
```

LoadConfigDlg Method

Interactively configures the acquisition device

Object.**LoadConfigDlg**(*parentWnd* As OLE_HANDLE) As Boolean

Parameters

parentWnd Parent window for the dialog box (0 if no parent window)

Description

Displays a dialog box that allows you to interactively choose the server, device, and configuration file (CCF)

for the acquisition. After successfully using this dialog, the Server, Device, and ConfigFile properties are correctly configured for acquisition. The Online property is also automatically set to True, allowing you to call the Snap or Grab methods to acquire images.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result as Boolean
Result = SapAcquisition1.LoadConfigDlg(hWnd)
```

LoadFlatField Method

Loads the flat-field gain and offset buffers from disk

Object.**LoadFlatField**(*fileName* As String) As Boolean

Parameters

fileName Name of file containing the gain and offset buffer values

Description

Loads flat-field correction gain and offset buffers from disk. The specified file is in TIFF format, and contains the data for both buffers, which are then available through the FlatFieldGain and FlatFieldOffset properties.

Use the FlatFieldDlg method to interactively initialize the buffer values instead.

To enable flat-field correction, you must then set the FlatField property to True.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result as Boolean
Result = SapAcquisition1.LoadFlatField("FlatFieldBuffers.tif")

If Result = True Then
    SapAcquisition1.FlatField = True
End If
```

Lut Property

LUT sub-component

object.**Lut** As SapLut

Description

Gives direct access to the internal LUT sub-component, which is automatically created inside the control.

Changes to the LUT only take effect after calling the ProgramLut method.

Example

```
Dim NumEntries as Long
NumEntries = SapAcquisition1.Lut.Entries
```

LutEnable Property

Enable/disable acquisition LUT

object.**LutEnable** As Boolean

Description

Specifies whether the acquisition LUT is enabled or disabled.

To directly access the LUT, use the Lut property. Changes to the LUT only take effect after calling the ProgramLut method.

The initial value for this property is False, unless you specify another value at design-time.

This property is read only during live grab, that is, when the Grabbing property is True.

Example

```
Dim Result as Boolean
Result = SapAcquisition1.ProgramLut()

If Result = True Then
    SapAcquisition1.LutEnable = True
End If
```

Message Method

Reports a message using the current error mode

Object.**Message**(*text* As String) As Boolean

Parameters

text Message to report

Description

Reports a custom message using the current error reporting mode.

See the ErrorMode property for a description of the available modes.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result as Boolean
Result = SapAcquisition1.Message("Processing task failed")
```

NumDevices Property

Number of available devices (read only)

object.**NumDevices** As Long

Description

Gets the number of available acquisition devices for the server currently selected using the Server property. You can then use the AllDevices property to get a text description of each.

This property is only available at runtime.

Example

```
DevicesText = "Number of devices = " & SapAcquisition1.NumDevices
```

NumServers Property

Number of available acquisition servers (read only)

object.**NumServers** As Long

Description

Gets the number of servers that have at least one acquisition device. You may then use the AllServers property to get a text description of each.

Example

```
ServersText = "Number of acquisition servers = " & SapAcquisition1.NumServers
```

OnBoardBuffers Property

Number of internal buffers on the acquisition device

object.**OnBoardBuffers** As Long

Description

Specifies the number of internal hardware buffers to be used between the acquisition and the image buffers.

If you change the value for this property you should always read it back. If the new value is equal to 0 it means that the acquisition hardware has no internal buffers, or that the value cannot be changed. If this new value is less than the one originally written it means that there is not enough internal memory for all the buffers.

The initial value for this property is 0, unless you specify another value at design-time.

This property is read only during live grab, that is, when the Grabbing property is True.

Example

```
Dim NumInternal as Long
NumInternal = 3

SapAcquisition1.OnBoardBuffers = NumInternal

If SapAcquisition1.OnBoardBuffers < NumInternal Then
    MsgBox("Unable to use " & NumInternal & " internal buffers")
End If
```

Online Property

Availability of image acquisition

object.**Online** As Boolean

Description

Specifies whether image acquisition is available and correctly configured. By default, the acquisition control is offline immediately after its initialization.

Do not confuse this property with the Valid property. The latter only indicates if the control is in a working state; acquisition may still be not available. For example, it is still possible to load images from disk in this case.

The initial value for this property is False.

This property is only available at runtime. It is also read only during live grab, that is, when the Grabbing property is True.

Example

```
SapAcquisition1.Server = "Bandit_II_1"  
SapAcquisition1.Device = "Standard Composite Video & Y/C"  
SapAcquisition1.ConfigFile = "c:\CameraFiles\MyCamera.ccf"  
  
SapAcquisition1.Online = True
```

Planar Property

Planar acquisition mask

object.**Planar** As Long

Description

Specifies the current configuration for synchronous acquisition into vertical planar buffers, where all cameras are synchronized together.

Individual bits for this property are set to 1 if the corresponding camera is enabled for planar acquisition, otherwise, they are set to 0. The entry at bit 0 corresponds to the first camera, the entry at bit 1 corresponds to the second camera, and so on.

When this property is set to 0, planar acquisition is disabled. In this case, the camera identified by the CamSel property is used for acquisition.

The initial value for this property is 0, unless you specify another value at design-time.

This property is read only during live grab, that is, when the Grabbing property is True.

Example

```
` Use the first two cameras  
SapAcquisition1.Planar = 3
```

ProgramLut Method

Reprograms acquisition LUT

Object.**ProgramLut**() As Boolean

Description

Reprograms the acquisition LUT.

After modifying the current LUT through the Lut property, use this method to apply the changes. You must set the LutEnable property to True in order to affect acquired images.

Note that some acquisition devices do not support enabling or disabling the LUT.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Result = SapAcquisition1.Lut.Reverse

If Result = True Then
    Result = SapAcquisition1.Lut.ProgramLut
End If
```

Reset Method

Reset the current acquisition server

Object.**Reset**(*notify* As Boolean) As Boolean

Parameters

notify Use end of server reset event

Description

Resets the hardware device associated with the server identified by the Server property.

If *notify* is False, then this method returns as soon as the reset operation is complete, or if the time interval specified by the ResetTimeout property has expired. In the latter case, the return value is False.

If *notify* is True, then the method returns immediately. The end of reset event EndReset is then fired as soon as the reset operation is complete. No event is fired if the time interval specified by the ResetTimeout property expires first.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Result = SapAcquisition1.Reset(True)

(other code)

Private Sub SapAcquisition1_EndReset(Server As String)
    MsgBox("Received end of reset event for " & Server)
End Sub
```

ResetTimeout Property

Timeout value for device reset

object.**Planar** As Long

Description

Specifies the timeout value (in milliseconds) used when resetting a server using the Reset method.

The initial value for this property is 20000 (20 seconds), unless you specify another value at design-time.

Example

```
Dim Result As Boolean

SapAcquisition1.ResetTimeout = 5000
Result = SapAcquisition1.Reset(False)

If Result = False Then
    MsgBox("Server reset failed because the timeout has expired")
End If
```

SapControlType Property

Type of Sopera LT control (read only)

object.**SapControlType** As enumSapControlType

Description

Specifies the type of the ActiveX control. For the SapAcquisition control, this value is always SapControlTypeAcquisition

Example

```
CtrlTypeText.Caption = "Control type = " & SapAcquisition1.SapControlType
```

SapError Event

Error event

Object **SapError**(*value* As Long, *description* As String)

Parameters

value Numeric value of the error
description Text description of the error

Description

When the value of the `ErrorMode` property is `ErrorModeEvent`, this event will be fired as soon as an error occurs internally in the control or when the `Message` method is called.

Example

```
Dim Result As Boolean

SapAcquisition1.ErrorMode = ErrorModeEvent
Result = SapAcquisition1.Message("This triggers the SapError event")

(other code)

Private Sub SapAcquisition1_SapError(ByVal Value As Long, description As String)
    MsgBox("Sapera error event, description = " & Chr(13) & description)
End Sub
```

SaveConfig Method

Saves the current acquisition parameters

Object **SaveConfig**(*fileName* As String) As Boolean

Parameters

fileName Name of the acquisition configuration file

Description

Saves the current values of the camera and frame grabber related acquisition parameters to the specified CCF file.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result as Boolean
Result = SapAcquisition1.SaveConfig("Default.ccf")
```

SaveFlatField Method

Saves the flat-field gain and offset buffers to disk

Object.**SaveFlatField**(*fileName* As String) As Boolean

Parameters

fileName Name of file for the gain and offset buffer values

Description

Saves flat-field correction gain and offset buffers to disk. The specified file is always written in TIFF format, no matter which file extension you specify.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result as Boolean
Result = SapAcquisition1.SaveFlatField("FlatFieldBuffers.tif")
```

SerialName Property

Serial port name on current device (read only)

object.**SerialName** As String

Description

Returns the name of the serial port attached to the acquisition device specified by the current values of the Server and Device properties.

The string is empty if there is no associated serial port.

Example

```
NameText.Caption = SapAcquisition1.SerialName
```

SerialNumber Property

Acquisition device serial number (read only)

object.**SerialNumber** As String

Description

Returns a text representation of the serial number corresponding to the hardware device identified by the current value of the Server property. It consists of either the letter ‘S’ or ‘H’ followed by seven digits, for example, “S1234567”.

The value of this property is only relevant when the Online property is True.

Example

```
SerialNumberText.Caption = SapAcquisition1.SerialNumber
```

Server Property

Acquisition server name

object.**Server** As String

Description

Specifies the name of the current acquisition server.

You can also use the LoadConfigDlg method to interactively choose the server, device, and configuration file (CCF) for the acquisition.

The initial value for this property is an empty string, unless you specify another value at design-time.

This property is read only when the Online property is True.

Example

```
SapAcquisition1.Server = "X64CL_1"
```

SetParam Method

Sets value of any low-level acquisition or transfer parameter (4 bytes)

Object.**SetParam**(*param* As Long, *value* As Long, *updateNow* as Boolean) As Boolean

Parameters

<i>param</i>	Low-level Spera parameter to write
<i>value</i>	New parameter value
<i>updateNow</i>	Allows delayed updating of acquisition parameters

Description

This method allows direct write access to low-level Spera parameters for the acquisition or transfer modules. When the parameter value requires more than 4 bytes of storage, use the SetParamEx method instead.

If *updateNow* is True, calling SetParam will program the acquisition hardware with the new value immediately. However, some parameters should not be set individually, as this may result in inconsistencies and error conditions in the acquisition resource.

If *updateNow* is False, new parameter values are accumulated internally. The next time *SetParam* is called with *updateNow* set to True, all new values will be sent in one operation to the acquisition hardware, thus avoiding the problems just described.

Delayed updating applies to acquisition parameters only.

Note that *SetParam* is rarely required. The acquisition control already uses important parameters internally for self-configuration and validation.

See the *Sapera Acquisition Parameters Reference Manual* and the *Sapera Basic Modules Reference Manual* for a full description of all the parameters and their possible values.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Left As Long
Dim Top As Long
Dim Width As Long
Dim Height As Long
Dim Result As Boolean

Result = SapAcquisition1.SetParam(CORACQ_PRM_CAMSEL, 1, False)

Left = 40
Top = 80
Width = 160
Height = 80

Result = SapAcquisition1.SetParam(CORACQ_PRM_CROP_TOP, Top, False)
Result = SapAcquisition1.SetParam(CORACQ_PRM_CROP_LEFT, Left, False)
Result = SapAcquisition1.SetParam(CORACQ_PRM_CROP_WIDTH, Width, False)
Result = SapAcquisition1.SetParam(CORACQ_PRM_CROP_HEIGHT, Height, True)
```

SetParamEx Method

Sets value of any low-level acquisition or transfer parameter

Object.**SetParamEx**(*param* As Long, *value* As Variant, *updateNow* as Boolean) As Boolean

Parameters

<i>param</i>	Low-level Sapera parameter to write
<i>value</i>	New parameter value
<i>updateNow</i>	Allows delayed updating of acquisition parameters

Description

This method allows direct write access to low-level Sapera parameters for the Acquisition or Transfer modules. If the parameter value requires 4 bytes of storage, the *SetParam* method can be used instead.

If *updateNow* is True, calling *SetParamEx* will program the acquisition hardware with the new value immediately. However, some parameters should not be set individually, as this may result in inconsistencies and error conditions in the acquisition resource.

If *updateNow* is False, new parameter values are accumulated internally. The next time *SetParamEx* is called with *updateNow* set to True, all the new values will be sent in one operation to the acquisition hardware, thus avoiding the problems just described.

Delayed updating applies to acquisition parameters only.

Note that SetParamEx is rarely required. The acquisition control already uses important parameters internally for self-configuration and validation.

See the *Sapera Acquisition Parameters Reference Manual* and the *Sapera Basic Modules Reference Manual* for a full description of all the parameters and their possible values.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Result = SapAcquisition1.SetParamEx(CORACQ_PRM_CAM_NAME, "New Name")
```

ShowProperties Method

Shows property dialog for acquisition control

Object.**ShowProperties()** As Boolean

Description

This method displays the property dialog for the control. Not all properties are included in this dialog. However, the most common ones, both for the acquisition component and the various sub-components, are included.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Result = SapAcquisition1.ShowProperties
```

Signal Event

Notification of signal status changes

Object_**Signal**(*value* As enumSignalStatus)

Parameters

value Numeric value of the error

Description

When the value of the SignalNotify property is True, this event is then fired whenever the status of input signals connected to the acquisition device changes.

See the SignalStatus property for a list of possible values.

Example

```
SapAcquisition1.SignalNotify = True
(other code)
```

```
Private Sub SapAcquisition1_Signal(ByVal Value As enumSignalStatus)
    MsgBox("Signal status changed, new value = " & Value)
End Sub
```

SignalNotify Property

Enables signal status events

object.**SignalNotify** As Boolean

Description

Specifies whether the Signal event should be fired whenever the status of input signals connected to the acquisition device changes.

The initial value for this property is False, unless you specify another value at design-time.

Example

```
SapAcquisition1.SignalNotify = True

(other code)

Private Sub SapAcquisition1_Signal(ByVal Value As enumSignalStatus)
    MsgBox("Signal status changed, new value = " & Value)
End Sub
```

SignalStatus Property

Current input signal status (read only)

object.**SignalStatus** As enumSignalStatus

Description

Specifies the status of input signals connected to the acquisition device. Since many signals may be detected at the same time, values are usually combined together using a bitwise OR operation. The following values are supported:

SignalNone	No signal
SignalHSyncPresent	Horizontal sync signal (analog video) or line valid (digital video)
SignalVSyncPresent	Vertical sync signal (analog video) or frame valid (digital video)
SignalPixelClkPresent	Pixel clock signal
SignalChromaPresent	Color burst signal (valid for NTSC and PAL)
SignalHSyncLock	Successful lock to a horizontal sync signal (analog video) or a line valid (digital video)
SignalVSyncLock	Successful lock to a vertical sync signal (analog video) or a frame valid (digital video)

This property is only available at runtime.

Example

```
If SapAcquisition1.SignalStatus And SignalHSyncPresent Then
    HsyncText.Caption = "Horizontal sync / line valid is present"
Else
    HsyncText.Caption = "Horizontal sync / line valid is NOT present"
End If
```

Snap Method

Acquires one or more images

object.**Snap**(*count* As Long) As Boolean

Parameters

count Number of images to acquire

Description

Acquires a finite number of images (usually 1).

This method returns as soon as acquisition has begun. You may, therefore, need to call the Wait method immediately after Snap to ensure that all images have been acquired before proceeding.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result as Boolean
Result = SapAcquisition1.Snap(1)
```

SoftTrigger Method

Software trigger for acquisition

object.**SoftTrigger**(*trigType* As enumSoftwareTrigger) As Boolean

Parameters

trigType Trigger type can be one of the following values:

SoftwareTriggerExt	External trigger
SoftwareTriggerExtFrame	External frame trigger
SoftwareTriggerExtLine	External line trigger

Description

Simulates a trigger to the acquisition device. Use this method for testing purposes when the actual hardware trigger is not available. This feature may not be implemented on the current acquisition device.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result as Boolean
Result = SapAcquisition1.SoftTrigger(SoftwareTriggerExt)
```

Valid Property

Current state of object (read only)

object.**Valid** As Boolean

Description

Specifies whether the acquisition control is correctly initialized.

Do not confuse this property with the Online property, as it only indicates if the control is in a working state, but acquisition may still be not available. For example, it is still possible to load images from disk in this case.

Example

```
If SapAcquisition1.Valid = True Then
    ValidText.Caption = "Acquisition control is correctly initialized"
Else
    ValidText.Caption = "Acquisition control is NOT correctly initialized"
End If
```

VertFlip Property

Vertical flip enable

object.**VertFlip** As Boolean

Description

Indicates whether vertical flipping (mirroring) mode for acquired images is active.

The initial value for this property is False, unless you specify another value at design-time.

This property is read only during live grab, that is, when the Grabbing property is True.

Example

```
SapAcquisition1.VertFlip = True
```

Wait Method

Wait for end of acquisition

object.**Wait**(*timeOut* As Long, *abortDlg* as Boolean) As Boolean

Parameters

timeOut Maximum amount of time to wait, in milliseconds (ms).

abortDlg Use an abort dialog box after the timeout interval has expired.

Description

Waits for the complete termination of image acquisition. You may want to call Wait after the Snap method to make certain that the required number of images have been acquired before proceeding. You should definitely call Wait after calling the Freeze method.

If *abortDlg* is False and the time-out expires when the acquisition is still not completed, this method then returns False. A common reason for this error is some manner of hardware failure. In this case, call the Abort method to unconditionally terminate the transfer.

You may also get an error if the time-out is too small and does not give the acquisition enough time to complete. You should, therefore, always specify a value large enough to allow one full image to be acquired. You can specify a much larger value (like a few seconds) if your application allows it.

Setting *abortDlg* to True yields a more evolved version of this method. In addition to the specified time-out, it allows you to wait an indefinite amount of time while giving you the option to abort at any time. This is useful when the time for one frame is variable (for example, external trigger).

If *abortDlg* is True, and the acquisition has already finished, the dialog then will not be shown and Wait returns True immediately. The same happens if the acquisition ends before the specified time-out.

If the acquisition is still active after the time-out period, then the dialog appears. If the acquisition ends before you click the **Abort** button, the dialog then automatically closes and Wait returns True. Wait returns False if **Abort** is clicked.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result as Boolean
Result = SapAcquisition1.Freeze
If Result = True Then
    Result = SapAcquisition1.Wait(5000, False)
End If
```

Xfer Event

Transfer events

Object **Xfer**(*eventType* As enumXferEvent, *eventCount* As Long, *bufIndex* As Long, *isTrash* As Boolean, *pairIndex* As Long)

Parameters

<i>eventType</i>	Event type for which the transfer event is triggered.
<i>eventCount</i>	Current count of transfer events. The initial value is 1 and increments after every event.
<i>bufIndex</i>	Index of the buffer resource associated with the transfer event, starting at 0.
<i>isTrash</i>	Specifies if the transfer event is associated with a trash buffer.
<i>pairIndex</i>	Reserved for future use.

Description

Transfer events are fired each time one or more of the conditions registered using the XferEventType property are encountered. If two or more of these conditions happen simultaneously, they are then reported as such, with the *eventType* argument containing a binary OR combination of their values. Setting the XferEventType property to the special value XferEventNone disables transfer events completely.

See the XferEventType property for a list of possible values.

Note that a separate event count is associated with each buffer resource. This means that if you have a buffer count equal to 3 (Count Property of SapBuffer), you will get 3 Xfer events with *eventCount* equal to 1, one for each of the 3 possible values for *bufIndex*.

Example

```
SapAcquisition1.XferEventType = XferEventEndOfFrame

(other code)

Private Sub SapAcquisition1.Xfer(ByVal eventType As enumXferEvent, ByVal eventCount As Long, ByVal bufIndex As Long, ByVal isTrash As Boolean, ByVal pairIndex As Long)
    EventText.Caption = "Transfer Event (" & eventType & ", " & eventCount & ")"
End Sub
```

XferEventType Property

Transfer event type

object.**XferEventType** As enumXferEvent

Description

Specifies which types are registered for transfer events. To register more than one type, combine the values together using a bitwise OR operation. The following values are supported:

XferEventNone	Disable transfer events
XferEventStartOfField	Start of field (odd or even)
XferEventStartOfOdd	Start of odd field
XferEventStartOfEven	Start of even field
XferEventStartOfFrame	Start of frame
XferEventEndOfField	End of field (odd or even)
XferEventEndOfOdd	End of odd field
XferEventEndOfEven	End of even field
XferEventEndOfFrame	End of frame
XferEventEndOfLine	After a specific line number <i>eventType = XferEventEndOfLine lineNum</i>
XferEventEndOfNLines	After a specific line number (linescan cameras only) <i>eventType = XferEventEndOfNLines numLines</i>
XferEventEndOfTransfer	End of transfer, that is, after all frames have been transferred following calls to the Snap or Grab/Freeze methods.
XferEventLineUnderrun	The number of active pixels per line received from a video source is less than it should be.
XferEventFieldUnderrun	The number of active lines per field received from a video source is less than it should be.

Setting this property to the special value XferEventNone disables transfer events completely.

The initial value for this property is XferEventNone, unless you specify another value at design-time.

This property is read only during live grab, that is, when the Grabbing property is True.

Example

```
SapAcquisition1.XferEventType = XferEventEndOfFrame

(other code)

Private Sub SapAcquisition1.Xfer(ByVal eventType As enumXferEvent, ByVal eventCount
As Long, ByVal bufIndex As Long, ByVal isTrash As Boolean, ByVal pairIndex As Long)
    EventText.Caption = "Transfer Event(" & eventType & ", " & eventCount & ")"
End Sub
```

SapBuffer

SapBuffer Member List

Properties	Description
Valid	Current state of object (read only)
Count	Number of buffers
Width	Buffer width
Height	Buffer height
Format	Buffer data format
BitsPerPixel	Pixel depth in bits (read only)
BytesPerPixel	Pixel depth in bytes (read only)
Pitch	Number of bytes between the starting address of two consecutive lines (read only)
Type	Buffer type (scatter-gather, overlay, ...)
Index	Current buffer index
State(Index)	State of buffer: empty/full (runtime)
CounterStamp(Index)	Buffer counter stamp value (read only, runtime)
DataAddress (Index)	Address of buffer data (read only, runtime)
SpaceUsed(Index)	Number of data bytes actually stored in buffer (read only, runtime)
ROI	Current region of Interest
UseROI	Enable region of Interest
Methods	Description
ClearBlack	Clears one (or all) buffer to black
Clear	Clears one (or all) buffer to any value (as a SapDataXxx object)
Load	Loads buffers from a file
LoadWithDlg	Interactively loads one buffer or a sequence from a file
Save	Saves buffers to a file
SaveWithDlg	Interactively saves one buffer or a sequence to a file

GetDataXY	Gets pixel value (as a SapDataXxx object) at a specific location
SetDataXY	Sets pixel value (as a SapDataXxx object) at a specific location
GetRectDataArray	Gets pixel values in a rectangular area
SetRectDataArray	Sets pixel values in a rectangular area
GetDataArray	Gets pixel values for the whole buffer
SetDataArray	Sets pixel values for the whole buffer

SapBuffer Member Description

BitsPerPixel Property

Pixel depth in bits (read only)

object.**BitsPerPixel** As Long

Description

Gets the number of significant bits for the image buffers.

The value for this property depends on the current value of the Format property.

Example

```
BitsPerPixelText.Caption = SapAcquisition1.Buffer.BitsPerPixel
```

BytesPerPixel Property

Pixel depth in bytes (read only)

object.**BytesPerPixel** As Long

Description

Gets the number of bytes required to store one pixel for the image buffers.

The value for this property depends on the current value of the Format property.

Example

```
BytesPerPixelText.Caption = SapAcquisition1.Buffer.BytesPerPixel
```

Clear Method

Clears one or all buffers to any value

object.**Clear**(*index* As Long, *value* as Variant) As Boolean

Parameters

index Index of the buffer to clear, starting at 0.

value New value for buffer pixels.

Description

Clears the content of the image buffer at *index* to the specified value. The maximum index is equal to the value

of the Count property, minus 1. Set *index* to -1 to clear all image buffers.

The *value* argument must be of type SapDataMono, SapDataRGB, SapDataYUV, SapDataHSI, or SapDataHSV, depending on the value of the Format property.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Dim Data As New SapDataMono

Data.Mono = 128
Result = SapAcquisition1.Buffer.Clear(-1, Data)
```

ClearBlack Method

Clears one or all buffers to black

object.**ClearBlack**(*index* As Long) As Boolean

Parameters

index Index of the buffer to clear, starting at 0.

Description

Clears the content of the image buffer at *index* to black. The maximum index is equal to the value of the Count property, minus 1. Set *index* to -1 to clear all image buffers.

Note that black does not always correspond to the value 0. A good example of this is YUV data formats.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Result = SapAcquisition1.Buffer.ClearBlack(-1)
```

Count Property

Number of image buffers

object.**Count** As Long

Description

Specifies the number image buffers (excluding the trash buffer, if any).

The initial value for this property is 2, unless you specify another value at design-time.

This property is read only if the current object is a sub-component of a SapAcquisition component, and if the Online property of the latter is True.

Example

```
SapAcquisition1.Buffer.Count = 10
```

CounterStamp Property

Buffer counter stamp value (read only)

object.**CounterStamp**(*index* As Long) As Long

Parameters

index Buffer index, starting at 0.

Description

Gets a unique value associated with an image buffer. This value is expressed in microseconds. It has no meaning by itself. However, subtracting timestamp values for two image buffers gives the amount of time elapsed between the end of their respective data transfers.

Note that some acquisition devices do not support this feature.

This property is only available at runtime.

Example

```
SapAcquisition1.XferEventType = XferEventEndOfFrame

(other code)

Private Sub SapAcquisition1_Xfer(ByVal eventType As enumXferEvent, ByVal eventCount
As Long, ByVal bufIndex As Long, ByVal isTrash As Boolean, ByVal pairIndex As Long)
    CounterText(bufIndex).Caption = SapAcquisition1.Buffer.CounterStamp(bufIndex)
End Sub
```

DataAddress Property

Address of buffer data (read only)

object.**DataAddress** (*index* As Long) As Variant

Parameters

index Buffer index, starting at 0.

Description

Gets the virtual address where buffer data is stored. Read this property when you need to process buffers in the application itself. Since the GetDataArray and SetDataArray methods are too slow for this purpose, you need direct access through a pointer. In order to correctly interpret the raw data, you also need to know the values of some or all of the following properties: Width, Height, Format, BitsPerPixel, BytesPerPixel, and Pitch.

The address will rarely be needed by the application code itself. Instead, you will typically pass it to a processing function in another ActiveX control or an external DLL.

This property is only available at runtime.

Example

```
Dim I As Long

For I = 0 To SapAcquisition1.Buffer.Count - 1
    AddressText(I).Caption = SapAcquisition1.Buffer.DataAddress(I)
Next
```

Format Property

Buffer data format

object.**Format** As enumSapFormat

Description

Specifies the data format for the image buffers, as one of the following values:

Monochrome (unsigned)

SapFormatMono1	1-bit
SapFormatMono8	8-bit
SapFormatMono16	16-bit
SapFormatMono32	32-bit

Monochrome (signed)

SapFormatInt8	8-bit
SapFormatInt16	16-bit
SapFormatInt32	32-bit

RGB Color

SapFormatRGB5551	16-bit (5 for each of red/green/blue, 1 for alpha)
SapFormatRGB565	16-bit (5 for red, 6 for green, 5 for blue)
SapFormatRGB888	24-bit (8 for red, 8 for green, 8 for blue)
SapFormatRGB8888	32-bit (8 for each of red/green/blue, 8 for alpha)
SapFormatRGB101010	32-bit (10 for each of red/green/blue, 2 unused)
SapFormatRGB161616	48-bit (16 for each of red/green/blue)
SapFormatRGBP8	8-bit planar
SapFormatRGBP16	16-bit planar

YUV Color

SapFormatUYVY	16-bit, 4:2:2 subsampled
SapFormatYUY2	16-bit, 4:2:2 subsampled
SapFormatYVYU	16-bit, 4:2:2 subsampled
SapFormatYUYV	16-bit, 4:2:2 subsampled
SapFormatY411	12-bit, 4:1:1 subsampled
SapFormatY211	8-bit, 4:2:2 subsampled
SapFormatYUV	32-bit (8 for each of Y/U/V, 8 for alpha)

Other Formats

SapFormatHSV	32-bit HSV (8 for each component, 8 unused)
SapFormatHSI	32-bit HSI (8 for each component, 8 unused)
SapFormatHSIP8	8-bit HSI planar
SapFormatFloat	32-bit signed floating point
SapFormatPoint	64-bit (32-bit signed integer for both X and Y components)
SapFormatFPoint	64-bit (32-bit signed floating-point for both X and Y components)

This property is read only if the current object is a sub-component of a SapAcquisition component, in which case its value depends on the current acquisition data format. Otherwise, its initial value is SapFormatMono8.

Example

```
FormatText.Caption = SapAcquisition1.Buffer.Format
```

GetDataArray Method

Gets pixel values for a whole buffer

Object.**GetDataArray**(*index* As Long, *data* As Variant) As Boolean

Parameters

index Buffer index, starting at 0.

data Data area for pixel values

Description

Reads a consecutive series of elements (pixels) from an image buffer, ignoring line boundaries. When this method returns, all the pixels are copied to an array returned in the *data* argument. The type of the pixel elements in the array consists of the best match between the buffer format and the ActiveX compatible types.

If *index* is equal to -1, the current internal buffer index is then assumed.

Note that reading elements from video memory buffers may be very slow.

Return Value

Returns True if successful, False otherwise

Example

```
Dim BufData
Dim Result As Boolean

Result = SapAcquisition1.Buffer.GetDataArray(-1, BufData)

(modify pixel data)

Result = SapAcquisition1.Buffer.SetDataArray(-1, BufData)
```

GetDataXY Method

Gets pixel value at a specific location

Object.**GetDataXY**(*index* As Long, *x* As Long, *y* As Long, *data* As Variant) As Boolean

Parameters

index Buffer index, starting at 0.

x Horizontal position

y Vertical position

data Data area for pixel value

Description

Reads a single element (pixel) from an image buffer. When this method returns, pixel data is of type SapDataMono, SapDataRGB, SapDataYUV, SapDataHSI, or SapDataHSV, depending on the value of the Format property.

If *index* is equal to -1, the current internal buffer index is then assumed.

Note that reading elements from video memory buffers may be very slow.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Pixel As Long
Dim Result As Boolean
Dim BufData(100)

' This assumes that the buffer data is 8-bit monochrome, so that BufData(Pixel) is a
' SapDataMono object after the call to GetDataXY, and it has a Mono property
For Pixel = 1 To 100
    Result = SapAcquisition1.Buffer.GetDataXY(-1, Pixel, Pixel, BufData(Pixel))
    BufData(Pixel).Mono = BufData(Pixel).Mono + 64
    Result = SapAcquisition1.Buffer.SetDataXY(-1, Pixel, Pixel, BufData(Pixel))
Next Pixel
```

GetRectDataArray Method

Gets pixel values in a rectangular area

Object.**GetRectDataArray**(*index* As Long, , *x* As Long, *y* As Long, , *width* As Long, *height* As Long, *data* As Variant) As Boolean

Parameters

<i>index</i>	Buffer index, starting at 0.
<i>x</i>	Left coordinate of rectangle origin
<i>y</i>	Top coordinate of rectangle origin
<i>width</i>	Rectangle width
<i>height</i>	Rectangle height
<i>data</i>	Data area for pixel values

Description

Reads a rectangular region of elements (pixels) from an image buffer. When this method returns, all the pixels (within the rectangle) are copied to an array returned in the *data* argument. The type of the pixel elements in the array consists of the best match between the buffer format and the ActiveX compatible types.

If *index* is equal to -1, the current internal buffer index is then assumed.

Note that reading elements from video memory buffers may be very slow.

Return Value

Returns True if successful, False otherwise

Example

```
Dim BufData
Dim Result As Boolean

Result = SapAcquisition1.Buffer.GetRectDataArray(-1, 50, 50, 100, 100, BufData)

(modify pixel data)

Result = SapAcquisition1.Buffer.SetRectDataArray(-1, 200, 200, 100, 100, BufData)
```

Height Property

Buffer height

object.**Height** As Long

Description

Specifies the height (in lines) for the image buffers.

This property is read only if the current object is a sub-component of a SapAcquisition component, in which case its value depends on the current acquisition height. Otherwise, its initial value property is 480.

Example

```
HeightText.Caption = SapAcquisition1.Buffer.Height
```

Index Property

Current buffer index

object.**Index** As Long

Description

Specifies the index of the current buffer. You usually do not need to change its value, since it is automatically updated when acquiring images.

This property is read only if the current object is a sub-component of a SapAcquisition component.

Example

```
IndexText.Caption = SapAcquisition1.Buffer.Index
```

Load Method

Loads one buffer or a sequence from a file

Object.**Load**(*fileName* As String, *bufIndex* As Long, *numBuffers* As Long, *frameIndex* As Long, *options* As String) As Boolean

Parameters

<i>fileName</i>	Name of the image file to load.	
<i>bufIndex</i>	Index of the buffer (or first buffer, for sequences) in which to load, starting at 0. Specify -1 to use the current index.	
<i>numBuffers</i>	Maximum number of image buffers to load when the file contains a sequence, where a value of 0 is equivalent to the number of buffers in the current object.	
<i>frameIndex</i>	Index of first image frame to load when the file contains a sequence.	
<i>options</i>	String containing the loading options. The following are supported:	
	"-format bmp"	Window bitmap format
	"-format tiff"	TIFF format
	"-format jpeg"	JPEG format

"-format jpeg_2000-component [value] "	JPEG 2000 format. When loading into a monochrome buffer, specify which color component to load (0 for red, 1 for green, 2 for blue); otherwise, this argument is ignored.
"-format crc"	DALSA proprietary format
"-format raw -width [value]-height [value] -o [offset] "	Raw data format. You must specify the image width and height, as well as the offset of image data from the beginning of the file.
"-format avi"	AVI image sequence format
"-format auto"	Automatic format detection

Description

Loads an image file into the current image buffer. If no *options* are specified, the format is automatically detected.

If the format is AVI, you can use *frameIndex* to specify the first frame to load from the file. If *numBuffers* is 0, the number of frames loaded will not exceed the buffer count.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Dim FileName As String

FileName = Environ("SAPERADIR") & "\Images\Display\mono8.bmp"
Result = SapAcquisition1.Buffer.Load(FileName, 0, 1, 0, "-format auto")
```

LoadWithDlg Method

Interactively loads one buffer or a sequence from a file

Object.**LoadWithDlg**(*parentWnd* As OLE_HANDLE, *index* As Long, *sequence* As Boolean) As Boolean

Parameters

<i>parentWnd</i>	Parent window for the dialog box (0 if no parent window)
<i>index</i>	Index of the buffer (or first buffer, for sequences) in which to load, starting at 0. Specify -1 to use the current index.
<i>sequence</i>	Specifies whether the list of files to load should include sequence file (AVI) or not.

Description

Interactively loads an image file or a sequence starting into the image buffer specified by *index*.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Result = SapAcquisition1.Buffer.LoadWithDlg(hWnd, 0, False)
```

Pitch Property

Gets the offset between the start of two consecutive lines in memory (read only)

object.**Pitch** As Long

Description

Gets the number of bytes between two consecutive lines of an image buffer. This is usually equal to the number of bytes per line, with possible exceptions for buffers located in video memory.

Example

```
PitchText.Caption = SapAcquisition1.Buffer.Pitch
```

ROI Property

Current region of interest

object.**ROI** As SapRectangle

Description

Specifies the rectangular area to use within the image buffers for acquiring images. To enable the ROI, set the UseROI property to True.

You should not rely on any initial value for this property, therefore, you should specify it either at design-time or through application code.

This property is read only when the UseROI property is True.

Example

```
Dim NewROI As New SapRectangle

NewROI.Left = 120
NewROI.Top = 80
NewROI.Width = 240
NewROI.Height = 160

SapAcquisition1.Buffer.ROI = NewROI
SapAcquisition1.Buffer.UseROI = True
```

Save Method

Saves one buffer or a sequence to a file

Object.**Save**(*fileName* As String, *options* As String, *bufIndex* As Long, *numBuffers* As Long) As Boolean

Parameters

<i>fileName</i>	Name of the image file to save.	
<i>options</i>	String containing the saving options. The following are supported:	
	"-format bmp"	Window bitmap format
	"-format tiff"	TIFF format. Compression can be set to none, run-length
	-compression [none/rle/lzw/jpeg]	encoding, Lempel-Ziv-Welch, or JPEG. For the latter,
	-quality [value]"	you can also set a quality level.

<code>"-format jpeg</code> <code>-quality [value]"</code>	JPEG format. The quality level can vary between 1 and 100.
<code>"-format jpeg_2000</code> <code>-quality [value]"</code>	JPEG 2000 format. The quality level can vary between 1 and 100, where the latter specifies lossless compression.
<code>"-format crc"</code>	DALSA proprietary format
<code>"-format raw"</code>	Raw data format
<code>"-format avi"</code>	AVI image sequence format
<i>bufIndex</i>	Index of the first image buffer to save when the file contains a sequence, where a value of -1 is equivalent to the first buffer. If the file contains only one image, this then is the index of the buffer to save and -1 is equivalent to the current index.
<i>numBuffers</i>	Maximum number of image buffers to save when the file contains a sequence, where a value of 0 is equivalent to the number of buffers in the current object.

Description

Saves one or more image buffers to a file.

If the format is AVI, use the *bufIndex* and *numBuffers* arguments to specify the first buffer and the number of buffers to save. When saving to a file with any other format, *numBuffers* is ignored.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Dim FileName As String

FileName = Environ("TEMP") & "\MyImage.bmp"
Result = SapAcquisition1.Buffer.Save(FileName, "-format bmp", 0, 1))
```

SaveWithDlg Method

Interactively saves one buffer or a sequence to a file

Object.**SaveWithDlg**(*parentWnd* As OLE_HANDLE, *index* As Long, *sequence* As Boolean) As Boolean

Parameters

<i>parentWnd</i>	Parent window for the dialog box (0 if no parent window)
<i>index</i>	Index of the buffer (or first buffer, for sequences) to save, starting at 0. Specify -1 to use the current index.
<i>sequence</i>	Specifies whether the list of files to save should include a sequence file (AVI) or not.

Description

Interactively saves an image file or a sequence starting from the image buffer specified by *index*.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Result = SapAcquisition1.Buffer.SaveWithDlg(hWnd, 0, False)
```

SetDataArray Method

Sets pixel values for a whole buffer

Object.**SetDataArray**(*index* As Long, *data* As Variant) As Boolean

Parameters

index Buffer index, starting at 0.
data Data area containing pixel values

Description

Writes a consecutive series of elements (pixels) to an image buffer, ignoring line boundaries.

If *index* is equal to -1, the current internal buffer index is then assumed.

Note that writing elements to video memory buffers may be very slow.

Return Value

Returns True if successful, False otherwise

Example

```
Dim BufData
Dim Result As Boolean

Result = SapAcquisition1.Buffer.GetDataArray(-1, BufData)

(modify pixel data)

Result = SapAcquisition1.Buffer.SetDataArray(-1, BufData)
```

SetDataXY Method

Sets pixel value at a specific location

Object.**SetDataXY**(*index* As Long, *x* As Long, *y* As Long, *data* As Variant) As Boolean

Parameters

index Buffer index, starting at 0.
x Horizontal position
y Vertical position
data Data area containing pixel value

Description

Writes a single element (pixel) to an image buffer. Pixel data must be of type SapDataMono, SapDataRGB, SapDataYUV, SapDataHSI, or SapDataHSV, depending on the value of the Format property.

If *index* is equal to -1, the current internal buffer index is then assumed.

Note that writing elements to video memory buffers may be very slow.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Pixel As Long
Dim Result As Boolean
Dim BufData(100)

' This assumes that buffer data is 8-bit monochrome, so that BufData(Pixel) is a
' SapDataMono object after the call to GetDataXY, and it has a Mono property
For Pixel = 1 To 100
    Result = SapAcquisition1.Buffer.GetDataXY(-1, Pixel, Pixel, BufData(Pixel))
    BufData(Pixel).Mono = BufData(Pixel).Mono + 64
    Result = SapAcquisition1.Buffer.SetDataXY(-1, Pixel, Pixel, BufData(Pixel))
Next Pixel
```

SetRectDataArray Method

Sets pixel values in a rectangular area

Object.**SetRectDataArray**(*index* As Long, *x* As Long, *y* As Long, *width* As Long, *height* As Long, *data* As Variant) As Boolean

Parameters

<i>index</i>	Buffer index, starting at 0.
<i>x</i>	Left coordinate of rectangle origin
<i>y</i>	Top coordinate of rectangle origin
<i>width</i>	Rectangle width
<i>height</i>	Rectangle height
<i>data</i>	Data area containing pixel values

Description

Writes a rectangular region of elements (pixels) to an image buffer.

If *index* is equal to -1, the current internal buffer index is then assumed.

Note that writing elements to video memory buffers may be very slow.

Return Value

Returns True if successful, False otherwise

Example

```
Dim BufData
Dim Result As Boolean

Result = SapAcquisition1.Buffer.GetRectDataArray(-1, 50, 50, 100, 100, BufData)

(modify pixel data)

Result = SapAcquisition1.Buffer.SetRectDataArray(-1, 200, 200, 100, 100, BufData)
```

SpaceUsed Property

Number of data bytes actually stored in buffer (read only)

object.SpaceUsed(*index* As Long) As Long

Parameters

index Buffer index, starting at 0.

Description

Gets the actual number of data bytes stored in a buffer after acquiring an image. This is normally the same value that you would get by multiplying the values of the Width, Height, and BytesPerPixel properties. However, when acquiring variable-length images, this value can be smaller.

This property is only available at runtime.

Example

```
SapAcquisition1.XferEventType = XferEventEndOfFrame

(other code)

Private Sub SapAcquisition1.Xfer(ByVal eventType As enumXferEvent, ByVal eventCount
As Long, ByVal bufIndex As Long, ByVal isTrash As Boolean, ByVal pairIndex As Long)
SpaceText(bufIndex).Caption = SapAcquisition1.Buffer.SpaceUsed(bufIndex)End Sub
```

State Property

Current state of image buffer

object.State(*index* As Long) As enumBufferState

Parameters

index Buffer index, starting at 0.

Description

Specifies whether the specified buffer is ready to accept a new image, or currently contains unprocessed data. The following values are supported:

BufferStateEmpty	The buffer is ready to receive new data.
BufferStateFull	The buffer contains unprocessed data.
BufferStateOverflow	The buffer contains incorrect data due to insufficient hardware bandwidth. This state only occurs together when BufferStateEmpty is active (the two values are combined using bitwise OR).

Sapera LT automatically manages the buffer state by default if the current object is a sub-component of a SapAcquisition object, so that you rarely have to set this property directly. If you wish to perform this management yourself, you must first set the AutoEmpty property of the SapAcquisition object to False.

The initial value for this property is BufferStateEmpty.

Example

```
Dim I As Long
For I = 0 To SapAcquisition1.Buffer.Count - 1
    StateText(I).Caption = SapAcquisition1.Buffer.State(I)
Next
```

Type Property

Buffer type

object.**Type** As enumBufferType

Description

Specifies the image buffer type. The following values are supported:

BufferTypeContiguous

Buffers are allocated in Sapera LT contiguous memory (one large chunk of non-pageable and non-moveable memory reserved by Sapera LT at boot time). Buffer data is therefore contained in a single memory block (not segmented). These buffers can be used as the destination for image acquisition.

BufferTypeScatterGather

Buffers are allocated in noncontiguous memory (paged pool). Pages are locked in physical memory so that a scatter-gather list can be built. This allows allocation of very large buffers that can be used as the destination for image acquisition. The maximum amount of memory that can be allocated depends on available memory, the operating system, and the application(s) used. If the amount of system memory exceeds 4 GBytes, Sapera LT automatically uses BufferTypeScatterGatherPhysical instead.

BufferTypeVirtual

Similar to TypeScatterGather except that memory pages are not locked. This allows allocation of very large buffers, but they cannot be used as the destination for image acquisition.

BufferTypeOffscreen

Buffers are allocated in system memory. Displaying these buffers can use display adapter hardware to copy from the buffer to video memory. System memory offscreen buffers can be created using any pixel format, but calling the Show method of the Display sub-component will take longer to execute if the display hardware does not efficiently support its pixel format.

BufferTypeOffscreenVideo

Buffers are allocated in offscreen video memory. Displaying these buffers uses display adapter hardware to perform a fast copy in video memory. These buffers are typically used when a graphical element is reused for several consecutive frames without modification. In this case, it is more efficient to keep this element in video memory and use display hardware capabilities.

BufferTypeOverlay

Buffers are allocated in video memory. Once you call the Show method of the Display sub-component once, the display adapter overlay hardware will keep updating the display with the buffer contents with no additional calls. The pixel format of overlay buffers must be supported by the display hardware. Typically, overlay buffers support more pixel formats (like YUV) than offscreen buffers. Also, color keying is supported for overlays. The display sub-component determines the behavior of the overlay regarding key colors.

BufferTypeDummy

Dummy buffers do not have any data memory. They can be used as placeholders by data transfers when there is no physical data transfer.

The initial value for this property is BufferTypeScatterGather, unless you specify another value at design-time.

This property is read only if the current object is a sub-component of a SapAcquisition component and if the Online property of the latter is True.

Example

```
TypeText.Caption = SapAcquisition1.Buffer.Type
```

UseROI Property

Enable region of interest

object.**UseROI** As Boolean

Description

Specifies whether the current region of interest (ROI property) is enabled or disabled. The ROI specifies the rectangular area to use within the image buffers for acquiring images.

The initial value for this property is False, unless you specify another value at design-time.

This property is read only if the current object is a sub-component of a SapAcquisition component, and if the Online property of the latter is True.

Example

```
Dim NewROI As New SapRectangle

NewROI.Left = 120
NewROI.Top = 80
NewROI.Width = 240
NewROI.Height = 160

SapAcquisition1.Buffer.ROI = NewROI
SapAcquisition1.Buffer.UseROI = True
```

Valid Property

Current state of object

object.**Valid** As Boolean

Description

Specifies whether the buffer sub-component is correctly initialized.

Example

```
If SapAcquisition1.Buffer.Valid = True Then
    ValidText.Caption = "Buffer sub-component is correctly initialized"
Else
    ValidText.Caption = "Buffer sub-component is NOT correctly initialized"
End If
```

Width Property

Buffer width

object.**Width** As Long

Description

Specifies the width (in columns) for the image buffers.

This property is read only if the current object is a sub-component of a SapAcquisition component, in which case its value depends on the current acquisition width. Otherwise, its initial value is 640.

Example

```
WidthText.Caption = SapAcquisition1.Buffer.Width
```

SapDisplay

SapDisplay Member List

Properties

Valid	Current state of object (read only)
Window	Display window: none / ActiveX control
KeyColor	Current keying color
Scaling	View scaling mode
ZoomHorz	Horizontal zoom factor (if scaling = zoom)
ZoomVert	Vertical zoom factor (if scaling = zoom)
SourceROI	Source region of interest from buffer (read only)
DestROI	Destination region of interest on display (read only)
DataBits	Number of bits to display from image buffer

Methods

Show	Show an image in the view window
Hide	Hide a displayed image
GetDC	Get GDI device context for the view area
ReleaseDC	Free GDI device context for the view area

SapDisplay Member Description

DataBits Property

Number of bits to display from image buffers

object.**DataBits** As Long

Description

Specifies the number of significant bits to display for the image buffers. This is useful when this number does not correspond to the number of bytes per pixel, for example, images acquired from 10-bit cameras in 16-bit buffers.

The initial value for this property depends on the current buffer data format and number of bits per pixel, unless you specify another value at design-time.

Example

```
SapAcquisition1.Display.DataBits = SapAcquisition1.Buffer.BitsPerPixel
```

DestROI Property

Destination region of interest on display (read only)

object.**DestROI** As SapRectangle

Description

Specifies the rectangular region of the viewing area that will show the buffer area defined by the SourceROI property.

The value of this property is automatically calculated by Sapera according to the current values of the Scaling, ZoomHorz, and ZoomVert properties.

Example

```
Dim DestROI As Object
Set DestROI = SapAcquisition1.Display.DestROI

DestLeftText.Caption = DestROI.Left
DestTopText.Caption = DestROI.Top
DestWidthText.Caption = DestROI.Width
DestHeightText.Caption = DestROI.Height
```

GetDC Method

Get GDI device context for the view area

Object.**GetDC**(*dc* As Long) As Boolean

Parameters

dc Returns display context value.

Description

Gets the Windows Device Context corresponding to the current view area.

Make certain that you call the ReleaseDC method after you are finished using the display context value, to avoid possible resource contention problems with Windows.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Dim DC As Long

Result = SapAcquisition1.Display.GetDC(DC)

(use DC for graphics)

Result = SapAcquisition1.Display.ReleaseDC()
```

Hide Method

Hide a displayed image

Object.**Hide()** As Boolean

Description

Hides the currently displayed buffer. This is only relevant when dealing with buffers of overlay type (BufferTypeOverlay).

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Result = SapAcquisition1.Display.Hide
```

KeyColor Property

Keying color for overlay display

object.**KeyColor** As SapDataRGB

Description

Specifies the keying color when dealing with buffers of overlay type (BufferTypeOverlay).

The initial value for this property is 0 for each color component, unless you specify other values at design-time.

For an 8-bit display mode, only the red color component is then relevant.

Example

```
Dim NewColor As New SapDataRGB

NewColor.Red = 10
NewColor.Green = 20
NewColor.Blue = 30

SapAcquisition1.Display.KeyColor = NewColor
```

ReleaseDC Method

Free GDI device context for the view area

Object.**ReleaseDC()** As Boolean

Description

Releases the Windows Device Context corresponding to the current view area. Call this method after you are finished using a display context value obtained from the GetDC method.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Dim DC As Long

Result = SapAcquisition1.Display.GetDC(DC)

(use DC for graphics)

Result = SapAcquisition1.Display.ReleaseDC()
```

Scaling Property

View scaling mode

object.**Scaling** As enumViewScaling

Description

Specifies the mode specifying how buffer content is scaled to the viewing area, as one of the following values:

ScalingNone	There is a one-to-one correspondence between image buffer data and pixels shown in the view area. This is the default mode.
ScalingFitToWindow	Displayed image buffer contents are scaled so that they are shown completely in the view area, while keeping the aspect ratio.
ScalingZoom	Image buffer contents are scaled by independent horizontal and vertical factors before being shown in the view area.

The ScalingZoom mode uses the current values of the ZoomHorz and ZoomVert properties as scaling factors.

Note that these mode apply to displayed images only, they do not affect buffer data.

The initial value for this property is ScalingNone, unless you specify another value at design-time.

Example

```
SapAcquisition1.Display.ZoomHorz = 1.5
SapAcquisition1.Display.ZoomVert = 2.0
SapAcquisition1.Display.Scaling = ScalingZoom
```

Show Method

Show an image in the view window

Object.**Show()** As Boolean

Description

Shows the current image buffer, that is, the one at the current index in the SapBuffer object.

You usually do not need to call this method explicitly if the AutoDisplay Property of the SapAcquisition component is True, as acquired images are shown automatically using the display mechanism built into the ActiveX control. However, if you set AutoDisplay to False, then you can use this method to display processed images instead.

Calling this method has no effect if the value of the Window Property is set to 0.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Result = SapAcquisition1.Display.Show
```

SourceROI Property

Source region of interest in image buffer (read only)

object.**SourceROI** As SapRectangle

Description

Specifies the rectangular region of the image buffers that will be displayed in the current viewing area defined by the DestROI property.

The value of this property is automatically calculated by Sapera according to the current values of the Scaling, ZoomHorz, and ZoomVert properties.

Example

```
Dim SourceROI As Object
Set SourceROI = SapAcquisition1.Display.SourceROI

SourceLeftText.Caption = SourceROI.Left
SourceTopText.Caption = SourceROI.Top
SourceWidthText.Caption = SourceROI.Width
SourceHeightText.Caption = SourceROI.Height
```

Valid Property

Current state of object

object.**Valid** As Boolean

Description

Specifies whether the display sub-component is correctly initialized.

Example

```
If SapAcquisition1.Display.Valid = True Then
    ValidText.Caption = "Display sub-component is correctly initialized"
Else
    ValidText.Caption = "Display sub-component is NOT correctly initialized"
End If
```

Window Property

Current display window

object.**Window** As Long

Description

Specifies the current display window.

Set this property to 1 to use the display mechanism built into the ActiveX control. Acquired images are shown automatically if the AutoDisplay Property of the SapAcquisition component is True. Alternatively, you can also set AutoDisplay to False, and call the Show Method explicitly whenever needed.

Set this property to 0 to disable built-in image display. In this case, the application is fully responsible for taking over image display responsibilities.

The initial value for this property is 1, unless you specify another value at design-time.

Example

```
SapAcquisition1.Display.Window = 0
SapAcquisition1.Visible = False
```

ZoomHorz Property

Horizontal zoom factor

object.**ZoomHorz** As Float

Description

Specifies the horizontal zoom factor to apply to displayed images when the Scaling property is set to ScalingZoom.

The initial value for this property is 1.0, unless you specify another value at design-time.

Example

```
SapAcquisition1.Display.ZoomHorz = 1.5
SapAcquisition1.Display.ZoomVert = 2.0
SapAcquisition1.Display.Scaling = ScalingZoom
```

ZoomVert Property

Vertical zoom factor

object.**ZoomVert** As Float

Description

Specifies the vertical zoom factor to apply to displayed images when the Scaling property is set to ScalingZoom.

The initial value for this property is 1.0, unless you specify another value at design-time.

Example

```
SapAcquisition1.Display.ZoomHorz = 1.5  
SapAcquisition1.Display.ZoomVert = 2.0  
SapAcquisition1.Display.Scaling = ScalingZoom
```

SapLut

SapLut Member List

Properties

Valid	Current state of object (read only)
Entries	Number of entries
Format	Data format
EntrySize	Number of bytes for each entry (read only)
Pages	Number of pages (read only)
Signed	Signed/unsigned flag (read only)
TotalSize	Total number of bytes for LUT data storage (read only)

Methods

GetEntry	Get specific LUT entry
SetEntry	Set specific LUT entry (using a SapDataXxx object)
GetDataArray	Get all LUT data entries
SetDataArray	Set all LUT data entries
Arithmetic	Apply an arithmetic operation to all entries
BinaryPattern	Modify some entries based on a binary pattern
Boolean	Apply an Boolean operation to all entries
Gamma	Apply Gamma correction to all entries
Normal	Apply a linear mapping with a positive slope to all entries
Reverse	Apply a linear mapping with a negative slope to all entries
Roll	Relocate all entries upwards or downwards as one block

Shift	Apply a logical shift to all entries
Slope	Modify a range of entries using a linear mapping
Threshold	Apply a threshold operation to some or all entries
Load	Load entries from a file
Save	Save entries to a file

SapLut Member Description

Arithmetic Method

Applies an arithmetic operation to all entries

object.**Arithmetic**(*operation* As enumArithmeticOp, *value* as Variant) As Boolean

Parameters

<i>operation</i>	Specifies how to modify LUT data elements. The following operations are available:
ArithmeticAdd	Addition with saturation: $\text{Entry}(I) = \text{Min}(\text{maxValue}, \text{Entry}(I) + \text{value})$
ArithmeticAsub	Absolute subtraction: $\text{Entry}(I) = \text{Abs}(\text{Entry}(I) - \text{value})$
ArithmeticMax	Maximum value: $\text{Entry}(I) = \text{Max}(\text{Entry}(I), \text{value})$
ArithmeticMin	Minimum value: $\text{Entry}(I) = \text{Min}(\text{Entry}(I), \text{value})$
ArithmeticScale	Scale to smaller maximum value: $\text{Entry}(I) = (\text{Entry}(I) * \text{value}) / \text{maxValue}$
ArithmeticSub	Subtraction with saturation: $\text{Entry}(I) = \text{Max}(\text{minValue}, \text{Entry}(I) - \text{value})$
<i>value</i>	Value used to modify entries.

Description

Modifies all LUT entries using an arithmetic operation.

The *value* argument must be of type SapDataMono or SapDataRGB, depending on the value of the Format property.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Data As New SapDataMono
Dim Result As Boolean
Dim Lut As Object

Set Lut = SapAcquisition1.Lut

If Lut.Format = SapFormatMono8 Or Lut.Format = SapFormatMono10 Then
    Data.Mono = 64
    Result = Lut.Arithmetic(ArithmeticAdd, Data)
End If
```

BinaryPattern Method

Modifies some entries based on a binary pattern

object.**BinaryPattern**(*bit* As Long, *value* as Variant) As Boolean

Parameters

bit Bit number that identifies the indices of the LUT data elements to modify.

value Value used to modify entries.

Description

Modifies some LUT entries based on a binary pattern. Only the entries with indices that have the specified *bit* set are modified using *value*. Each entry is calculated as follows:

```
If I And 2^bit <> 0 Then
    Entry(I) = value
End If
```

The *value* argument must be of type SapDataMono or SapDataRGB, depending on the value of the Format property.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Data As New SapDataMono
Dim Result As Boolean
Dim Lut As Object

Set Lut = SapAcquisition1.Lut

If Lut.Format = SapFormatMono8 Or Lut.Format = SapFormatMono10 Then
    Data.Mono = 0
    Result = Lut.Arithmetic(BinaryPattern(6, Data))
End If
```

Boolean Method

Applies an Boolean operation to all entries

object.**Boolean**(*operation* As enumBooleanOp, *value* as Variant) As Boolean

Parameters

operation Specifies how to modify LUT data elements. The following operations are available:

BooleanAnd Entry(I) = Entry(I) And *value*

BooleanOr Entry(I) = Entry(I) Or *value*

BooleanXor Entry(I) = Entry(I) Xor *value*

Value Value used to modify entries.

Description

Modifies all LUT entries using a bitwise Boolean operation.

The *value* argument must be of type SapDataMono or SapDataRGB, depending on the value of the Format property.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Data As New SapDataMono
Dim Result As Boolean
Dim Lut As Object

Set Lut = SapAcquisition1.Lut

If Lut.Format = SapFormatMono8 Or Lut.Format = SapFormatMono10 Then
    Data.Mono = -1
    Result = Lut.Arithmetic(BooleanXor, Data)
End If
```

Entries Property

Number of LUT entries

object.Entries As Long

Description

Specifies the number of LUT entries.

This property is read only if the current object is a sub-component of a SapAcquisition component, in which case its value depends on the current acquisition data format. Otherwise, its initial value is 256.

Example

```
EntriesText.Caption = SapAcquisition1.Lut.Entries
```

EntrySize Property

Number of bytes for each LUT entry (read only)

object.EntrySize As Long

Description

Specifies the number of bytes required to store a single LUT entry.

If the current object is a sub-component of a SapAcquisition component, then the initial value for this property depends on the current acquisition data format. Otherwise, its initial value is 1.

Example

```
EntrySizeText.Caption = SapAcquisition1.Lut.EntrySize
```

Format Property

LUT data format

object.**Format** As enumSapFormat

Description

Specifies the data format for LUT entries, as one of the following values:

Monochrome (unsigned)

SapFormatMono8	8-bit
SapFormatMono9	9-bit
...	...
SapFormatMono15	15-bit
SapFormatMono16	16-bit

Monochrome (signed)

SapFormatInt8	8-bit
SapFormatInt9	9-bit
...	...
SapFormatInt15	15-bit
SapFormatInt16	16-bit

Color (non-interlaced)

SapFormatColorNI8	8-bit
SapFormatColorNI9	9-bit
...	...
SapFormatColorNI15	15-bit
SapFormatColorNI16	16-bit

Color (interlaced)

SapFormatColorI8	8-bit
SapFormatColorI9	9-bit
...	...
SapFormatColorI15	15-bit
SapFormatColorI16	16-bit

For non-interlaced color formats, the red/green/blue components for one LUT element are stored separately:

RRR ... RRR	Red components of all elements
GGG ... GGG	Green components of all elements
BBB ... BBB	Blue components of all elements

For interlaced color formats, the red/green/blue components for one LUT element are stored together:

RGBRGBRGB	First three elements
...	...
RGBRGBRGB	Last three elements

This property is read only if the current object is a sub-component of a SapAcquisition component, in which case its value depends on the current acquisition data format. Otherwise, its initial value is SapFormatMono8.

Example

```
FormatText.Caption = SapAcquisition1.Lut.Format
```

Gamma Method

Applies Gamma correction to all entries

object.**Gamma**(*factor* As Float) As Boolean

Parameters

factor Gamma correction factor to apply.

Description

Modifies all LUT entries using inverse gamma correction with the specified *factor*. This is used to correct the light response of the camera, which is often a power function (referred to as the gamma function). A *factor* of 1 means no correction is applied and a normal LUT is computed instead.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Result = SapAcquisition1.Lut.Gamma(1.5)
```

GetDataArray Method

Gets all LUT data entries

Object.**GetDataArray**(*data* As Variant) As Boolean

Parameters

data Data area for LUT data values

Description

Directly reads raw LUT data.

When this method returns, all the LUT elements are copied to an array returned in the *data* argument. The type of elements in the array consists of the best match between the LUT format and the ActiveX compatible types.

Return Value

Returns True if successful, False otherwise

Example

```
Dim LutData
Dim Result As Boolean

Result = SapAcquisition1.Lut.GetDataArray(LutData)

(modify LUT data)

Result = SapAcquisition1.Lut.SetDataArray(LutData)
```

GetEntry Method

Gets a specific LUT entry

Object.**GetEntry**(*pos* As Long, *data* As Variant) As Boolean

Parameters

pos Position of LUT element to read, starting at 0.

data Data area for LUT value.

Description

Gets the value of a single LUT element. When this method returns, *data* is of type SapDataMono or SapDataRGB, depending on the value of the Format property.

Return Value

Returns True if successful, False otherwise

Example

```
Dim LutEntry
Dim Result As Boolean
Dim Text as String

Result = SapAcquisition1.Lut.GetEntry(0, LutEntry)

If Lut.Format = SapFormatMono8
    Text = LutEntry.Mono
End If
```

Load Method

Loads LUT entries from a file

Object.**Load**(*fileName* As String) As Boolean

Parameters

fileName Name of the LUT file to load.

Description

Loads LUT entries from a file. The Entries and Format properties of the LUT are updated to reflect the file contents.

Return Value

Returns True if successful, False otherwise

Example

```
Dim FileName As String
Dim Result As Boolean

FileName = Environ("SAPERADIR") & "\Lut\MyLut.lut"
Result = SapAcquisition1.Lut.Load(FileName)

(modify LUT entries)

Result = SapAcquisition1.Lut.Save(FileName)
```

Normal Method

Applies a linear mapping to all entries

object.**Normal()** As Boolean

Description

Modifies all LUT entries using a linear mapping with a positive slope, as follows:

Entry(0) = MinValue
Linear mapping from Entry(0) to Entry(MaxPos)
Entry(MaxPos) = MaxValue

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Result = SapAcquisition1.Lut.Normal
```

Pages Property

Number of LUT pages (read only)

object.**Pages** As Long

Description

Gets the number of color planes in the LUT. This value is usually 1 if the value of the Format property is monochrome, and 3 if it is color.

Example

```
PageText.Caption = SapAcquisition1.Lut.Pages
```

Reverse Method

Applies a reverse mapping to all entries

object.**Reverse()** As Boolean

Description

Modifies all LUT entries using a linear mapping with a negative slope, as follows:

Entry(0) = MaxValue
Linear mapping from Entry(0) to Entry(MaxPos)
Entry(MaxPos) = MinValue

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Result = SapAcquisition1.Lut.Reverse
```

Roll Method

Relocates all LUT entries upwards or downwards

object.**Roll**(*entries* As Long) As Boolean

Parameters

entries Number of entries by which LUT data should be shifted.

Description

Relocates LUT entries upwards or downwards as one block. The actual data elements are not modified, and their position relative to one another remains the same. If *entries* is positive, then a downward shift occurs. If it is negative, an upward shift occurs. This behavior is expressed as follows:

```
If entries > 0 Then
    Entry((I + entries) % MaxIndex] = Entry(I)
Else
    Entry(I) = Entry((I - entries) % MaxIndex]
End If
```

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Result = SapAcquisition1.Lut.Roll(64)
```

Save Method

Saves LUT entries to a file

Object.**Save**(*fileName* As String) As Boolean

Parameters

fileName Name of the LUT file to save.

Description

Saves LUT entries from a file.

Return Value

Returns True if successful, False otherwise

Example

```
Dim FileName As String
Dim Result As Boolean

FileName = Environ("SAPERADIR") & "\Lut\MyLut.lut"
Result = SapAcquisition1.Lut.Load(FileName)

(modify LUT entries)

Result = SapAcquisition1.Lut.Save(FileName)
```

SetDataArray Method

Sets all LUT data entries

Object.**SetDataArray**(*data* As Variant) As Boolean

Parameters

data New LUT entry values.

Description

Directly write new entry values to a LUT.

Return Value

Returns True if successful, False otherwise

Example

```
Dim LutData
Dim Result As Boolean

Result = SapAcquisition1.Lut.GetDataArray(LutData)

(modify LUT data)

Result = SapAcquisition1.Lut.SetDataArray(LutData)
```

SetEntry Method

Sets a specific LUT entry

Object.**SetEntry**(*pos* As Long, *data* As Variant) As Boolean

Parameters

pos Position of LUT entry to write, starting at 0.

data New value for LUT entry

Description

Sets the value of a single LUT entry. The *pos* argument must be of type SapDataMono or SapDataRGB, depending on the value of the Format property.

Return Value

Returns True if successful, False otherwise

Example

```
Dim LutEntry as New SapDataMono
Dim LutIndex As Long
Dim Result As Boolean

For LutIndex = 0 to 255
    LutEntry.Mono = 255 - LutIndex
    Result = SapAcquisition1.Lut.SetEntry(LutIndex, LutEntry)
Next LutIndex

Result = SapAcquisition1.ProgramLut
```

Shift Method

Apply a logical shift to all LUT entries

object.**Shift**(*bits* As Long) As Boolean

Parameters

bits Number of bits by which LUT entries should be shifted.

Description

Modifies all LUT entries using a logical shift. If *bits* is positive a left shift occurs and the least significant bits are filled with 0's. If *bits* is negative a right shift occurs and the most significant bits are filled with 0's. This behavior is expressed as follows:

```
If bits > 0 Then
    Entry(I) = LeftShift(Entry(I), bits)
Else
    Entry(I) = RighthShift(Entry(I), -bits)
End If
```

Return Value

Returns True if successful, False otherwise

Example

```
Dim Result As Boolean
Result = SapAcquisition1.Lut.Shift(1)
```

Signed Property

Signed/unsigned flag for LUT entries (read only)

object.**Signed** As Long

Description

Specifies if the current value of the Format property represents signed or unsigned data.

If the current object is a sub-component of a SapAcquisition component, then the initial value for this property depends on the current acquisition data format. Otherwise, its initial value is False.

Example

```
SignedText.Caption = SapAcquisition1.Lut.Signed
```

Slope Method

Modify a range of LUT entries with a linear mapping

object.**Slope**(*start* As Long, *end* As Long, *minVal* As Variant, *maxVal* As Variant, *clip* As Boolean) As Boolean

Parameters

start Starting LUT position for linear mapping.

end Ending LUT position for linear mapping.

minVal LUT entry value at starting position.

maxVal LUT entry value at ending position.

clip Specifies whether LUT entries outside the mapping range should also be modified.

Description

Modifies part of a LUT with a linear mapping. Entries from *start* to *end* are remapped from *minVal* to *maxVal*. If *clip* is True, then elements outside the range are unaffected. If False, then elements below *start* are set to *minVal* and elements above *end* are set to *maxVal*. This behavior is expressed as follows:

```
If clip = True Then
    Entry(0) to Entry(start - 1) = minVal
End If

Entry(start) = minVal
(Linear mapping from Entry(start) to Entry(end))
Entry(end) = maxVal

If clip = True Then
    Entry(end + 1) to Entry(maxIndex - 1) = maxVal
End If
```

Both *minVal* and *maxVal* must be of type SapDataMono or SapDataRGB, depending on the value of the Format property.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Start As Long
Dim End As Long
Dim MinVal As New SapDataMono
Dim MaxVal As New SapDataMono
Dim Result As Boolean
Dim Lut As Object

Set Lut = SapAcquisition1.Lut

If Lut.Format = SapFormatMono8 Or Lut.Format = SapFormatMono10 Then
    Start = Lut.Entries / 8
    End = (Lut.Entries * 7) / 8
    MinVal.Mono = 0
    MaxVal.Mono = 255
    Result = Lut.Slope(Start, End, MinVal, MaxVal, True)
End If
```

Threshold Method

Applies a threshold operation to some or all LUT entries

object.**Threshold**(*lowVal* As Variant, *highVal* As Variant) As Boolean

Parameters

lowVal Reference value for single threshold OR lower reference value for double threshold.

highVal Upper reference value for double threshold.

Description

Modifies all LUT elements using a threshold operation.

Setting the value of *highVal* to the special Empty value for the variant data type specifies a single threshold.

Entries with a value lower than *lowVal* are set to the lowest possible value. Elements with a value higher than or equal to *lowVal* are set to the highest possible value. This behavior is expressed as follows:

```
If Entry(I) < lowVal Then
    Entry(I) = minValue
Else
    Entry(I) = maxValue
```

Setting the value of *highVal* to a value different from Empty for the variant data type implements double threshold. Elements with a value higher than or equal to *lowVal*, but lower than *highVal*, are set to the highest possible value. Elements outside that range are set to the lowest possible value. This behavior is expressed as follows:

```
If Entry(I) >= lowVal And Entry(I) <= highVal Then
    Entry(I) = maxValue
Else
    Entry(I) = minValue
```

Both *lowVal* and *highVal* must be of type SapDataMono or SapDataRGB, depending on the value of the Format property.

Return Value

Returns True if successful, False otherwise

Example

```
Dim ThreshValue As New SapDataMono
Dim MinValue As New SapDataMono
Dim MaxValue As New SapDataMono
Dim Lut As Object

Set Lut = SapAcquisition1.Lut

If Lut.Format = SapFormatMono8 Or Lut.Format = SapFormatMono10 Then
    ThreshValue.Mono = Lut.Entries / 2
    Result = Lut.Threshold(ThreshValue, Empty)

    MinValue.Mono = Lut.Entries / 4
    MaxValue.Mono = (Lut.Entries * 3) / 4
    Result = Lut.Threshold(MinValue, MaxValue)
End If
```

TotalSize Property

Total number of bytes for all LUT entries (read only)

object.**TotalSize** As Long

Description

Specifies the total number of bytes required to store all LUT entries.

If the current object is a sub-component of a SapAcquisition component, then the initial value for this property depends on the current acquisition data format. Otherwise, its initial value is 256.

Example

```
TotalSizeText.Caption = SapAcquisition1.Lut.TotalSize
```

Valid Property

Current state of object

object.**Valid** As Boolean

Description

Specifies whether the LUT sub-component is correctly initialized.

Example

```
If SapAcquisition1.Lut.Valid = True Then  
    ValidText.Caption = "LUT sub-component is correctly initialized"  
Else  
    ValidText.Caption = "LUT sub-component is NOT correctly initialized"  
End If
```

SapRectangle

SapRectangle Member List

Properties	Description
Top	Topmost coordinate
Left	Leftmost coordinate
Width	Width in pixels
Height	Height in lines

Methods

Init	Set all properties to their default values
------	--

SapRectangle Member Description

Height Property

Height in lines
object.**Height** As Long

Description

Specifies the height (in lines) for the rectangle.

Example

```
Dim Rect As New SapRectangle
Rect.Height = 128
```

Init Method

Sets all properties to their default values
Object.**Init()** As Boolean

Description

Sets the values of all properties to default values, specifying a rectangle of 32x32 at coordinate (0,0).

Return Value

Returns True if successful, False otherwise

Example

```
Dim Rect As New SapRectangle

Rect.Init
Rect.Left = 64
Rect.Top = 32
Rect.Width = 256
Rect.Height = 128
```

Left Property

Leftmost coordinate

object.**Left** As Long

Description

Specifies the leftmost coordinate (column number) for the rectangle.

Example

```
Dim Rect As New SapRectangle  
Rect.Left = 64
```

Top Property

Topmost coordinate

object.**Top** As Long

Description

Specifies the topmost coordinate (line number) for the rectangle.

Example

```
Dim Rect As New SapRectangle  
Rect.Top = 32
```

Width Property

Width in columns

object.**Width** As Long

Description

Specifies the width (in columns) for the rectangle.

Example

```
Dim Rect As New SapRectangle  
Rect.Width = 256
```

SapDataMono

SapDataMono Member List

Properties	Description
Mono	Monochrome data value

Methods

Clear	Set property values to black
-------	------------------------------

SapDataMono Member Description

Clear Method

Sets property values to black

Object.**Clear**() As Boolean

Description

Sets the properties to the monochrome equivalent of black (equal to 0).

Return Value

Returns True if successful, False otherwise

Example

```
Dim Data As New SapDataMono

Data.Clear
Data.Mono = 100
```

Mono Property

Monochrome data value

object.**Mono** As Long

Description

Specifies the monochrome data value.

Example

```
Dim Data As New SapDataMono
Data.Mono = 100
```

SapDataRGB

SapDataRGB Member List

Properties	Description
Red	Red component of data value
Green	Green component of data value
Blue	Blue component of data value

Methods

Clear	Set property values to black
-------	------------------------------

SapDataRGB Member Description

Blue Property

Blue component of data value

object.**Blue** As Long

Description

Specifies the blue component of the data value.

Example

```
Dim Data As New SapDataRGB
Data.Blue = 200
```

Clear Method

Sets property values to black

Object.**Clear()** As Boolean

Description

Sets the properties to the RGB equivalent of black (equal to 0).

Return Value

Returns True if successful, False otherwise

Example

```
Dim Data As New SapDataRGB

Data.Clear
Data.Red = 100
Data.Green = 150
Data.Blue = 200
```

Green Property

Green component of data value

object.**Green** As Long

Description

Specifies the green component of the data value.

Example

```
Dim Data As New SapDataRGB  
Data.Green = 150
```

Red Property

Red component of data value

object.**Red** As Long

Description

Specifies the red component of the data value.

Example

```
Dim Data As New SapDataRGB  
Data.Red = 100
```

SapDataYUV

SapDataYUV Member List

Properties	Description
Y	Y component of data value
U	U component of data value
V	V component of data value
Methods	
Clear	Set property values to black

SapDataYUV Member Description

Clear Method

Sets property values to black

Object.**Clear()** As Boolean

Description

Sets the properties to the YUV equivalent of black (only Y is equal to 0).

Return Value

Returns True if successful, False otherwise

Example

```
Dim Data As New SapDataYUV  
  
Data.Clear  
Data.Y = 100  
Data.U = 150  
Data.V = 200
```

U Property

U component of data value

object.**U** As Long

Description

Specifies the U component of the data value.

Example

```
Dim Data As New SapDataYUV  
Data.U = 150
```

V Property

V component of data value

object.**V** As Long

Description

Specifies the V component of the data value.

Example

```
Dim Data As New SapDataYUV  
Data.V = 200
```

Y Property

Y component of data value

object.Y As Long

Description

Specifies the Y component of the data value.

Example

```
Dim Data As New SapDataYUV
Data.Y = 100
```

SapDataHSI

SapDataHSI Member List

Properties	Description
H	H component of data value
S	S component of data value
I	I component of data value

Methods

Clear	Set property values to black
-------	------------------------------

SapDataHSI Member Description

Clear Method

Sets property values to black

Object.**Clear**() As Boolean

Description

Sets the properties to the HSI equivalent of black.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Data As New SapDataHSI

Data.Clear
Data.H = 100
Data.S = 150
Data.I = 200
```

H Property

H component of data value

object.H As Long

Description

Specifies the H component of the data value.

Example

```
Dim Data As New SapDataHSI  
Data.H = 150
```

I Property

I component of data value

object.I As Long

Description

Specifies the I component of the data value.

Example

```
Dim Data As New SapDataHSI  
Data.I = 200
```

S Property

S component of data value

object.S As Long

Description

Specifies the S component of the data value.

Example

```
Dim Data As New SapDataHSI  
Data.S = 100
```

SapDataHSV

SapDataHSV Member List

Properties	Description
H	H component of data value
S	S component of data value
V	V component of data value

Methods

Clear	Set property values to black
-------	------------------------------

SapDataHSV Member Description

Clear Method

Sets property values to black

Object.**Clear**() As Boolean

Description

Sets the properties to the HSV equivalent of black.

Return Value

Returns True if successful, False otherwise

Example

```
Dim Data As New SapDataHSV  
  
Data.Clear  
Data.H = 100  
Data.S = 150  
Data.V = 200
```

H Property

H component of data value

object.**H** As Long

Description

Specifies the H component of the data value.

Example

```
Dim Data As New SapDataHSV  
Data.H = 150
```

S Property

S component of data value

object.S As Long

Description

Specifies the S component of the data value.

Example

```
Dim Data As New SapDataHSV  
Data.S = 200
```

V Property

V component of data value

object.V As Long

Description

Specifies the V component of the data value.

Example

```
Dim Data As New SapDataHSV  
Data.V = 100
```

SapDataFRGB

SapDataFRGB Member List

Properties	Description
Red	Red component of data value
Green	Green component of data value
Blue	Blue component of data value

Methods

Clear	Set property values to black
-------	------------------------------

SapDataFRGB Member Description

Blue Property

Blue component of data value

object.**Blue** As Float

Description

Specifies the blue component of the data value.

Example

```
Dim Data As New SapDataFRGB  
Data.Blue = 200.0
```

Clear Method

Sets property values to black

Object.**Clear**() As Boolean

Description

Sets the properties to the FRGB equivalent of black (equal to 0.0).

Return Value

Returns True if successful, False otherwise

Example

```
Dim Data As New SapDataFRGB  
  
Data.Clear  
Data.Red = 100.0  
Data.Green = 150.0  
Data.Blue = 200.0
```

Green Property

Green component of data value

object.**Green** As Float

Description

Specifies the green component of the data value.

Example

```
Dim Data As New SapDataFRGB  
Data.Green = 150.0
```

Red Property

Red component of data value

object.**Red** As Float

Description

Specifies the red component of the data value.

Example

```
Dim Data As New SapDataFRGB  
Data.Red = 100.0
```


DALSA Contact Information

Sales Information

Visit our web site: <http://www.imaging.com/>

Email: <mailto:info@dalsa-coreco.com>

International/Canada

DALSA
7075 Place Robert-Joncas Suite #142
St. Laurent, Quebec
H4M 2Z2
Canada

Tel: (514) 333-1301

Fax: (514) 333-1388

USA

DALSA
Bldg. 8 2nd Floor
900 Middlesex Turnpike
Billerica, Ma. 01821

Tel: (978) 670-2000

Fax: (978) 670-2010

Technical Support

Any support question or request can be submitted via our web site:

Technical support form via our web page:

<http://www.imaging.com/support>

Support requests for imaging product installations,

Support requests for imaging applications

Camera support information

<http://www.imaging.com/camsearch>

Product literature and driver updates

<http://www.imaging.com/download>

Glossary of Terms

Bandwidth

Describes the measure of data transfer capacity. A computer system's PCI expansion bus is rated for a maximum peak data bandwidth of 132 MB/sec. PCI devices must share the maximum PCI bus bandwidth when transferring data to and from system memory or other devices.

BMP file

(**BitMaP** file) Also known as a "bump" file, it is the Windows native bitmap format. BMP files provide formats for 2, 16, 256 or 16 million colors (1-bit, 4-bit, 8-bit, and 24-bit color). Refer to the *Sapera Basic Modules Reference Manual* – Buffer File Formats.

CCF

Camera configuration file.

Composite Video

A video signal that is composed of the luminance and color information plus the synchronization signals together. Common composite video formats are NTSC and PAL.

Contiguous memory

A block of physical memory, occupying consecutive addresses.

Driver

Also called a device driver, a program routine that links a peripheral device to the operating system. Specific to the Bandit-II, its VGA driver is required for its display adapter functionality and a device driver is required for its frame grabber capabilities.

Frame

One complete image data set or its equivalent storage space.

Grab

Acquiring an image frame by means of a frame grabber.

Host

Refers to the computer system that supports the installed frame grabber.

Interlaced

Describing the standard television method of raster scanning in which the image is the product of two fields, each of which is made up of the image's alternate lines (that is, one field is comprised of lines 1, 3, 5, and so forth, and the other is comprised of lines 2, 4, 6, and so forth).

LSB

Least Significant Bit in a binary data word.

LUT

(Lookup Table) In image processing, the memory that stores the values for point processes. Input pixel values are those from the original image, while output values are those displayed on the monitor as altered by the chosen point process.

MSB

Most Significant Bit in a binary data word.

NTSC

National Television Systems Committee. Color TV standard used in North America and other countries. The interlaced video signal is composed of a total of 525 video lines at a frame rate of 30 Hz.

PAL

Phase Alteration by Line. Color TV standard used in most of Europe and other countries. The interlaced video signal is composed of a total of 625 video lines at a frame rate of 25 Hz.

Pixel

A contraction of "picture element". The number of pixels describes the number of digital samples taken of the analog video signal. The number of pixels per video line by the number of active video lines describes the acquisition image resolution. The binary size of each pixel (for example, 8-bits, 15-bits, 24-bits) defines the number of gray levels or colors possible for each pixel.

RAW

A Sapera data file format where there is no header information and which supports any Sapera buffer type. Refer to the *Sapera Basic Modules Reference Manual – Buffer File Formats*.

ROI - Region of Interest

A rectangular segment of an image to be isolated for saving or other functions.

RGB

A representation of color using the three primary colors (red, green, blue) as components. Video signals in RGB format are typically a non-composite video standard. A digital true color image can be represented by 8-bits per color (24-bits/pixel). Often image data is stored or transferred in 32-bits/pixel where the upper 8-bits of each pixel are unused.

Scatter Gather

Host system memory allocated for frame buffers is virtually contiguous but physically scattered throughout all available memory.

Y/C

Also known as S-video or S-VHS. A composite video standard where the signal is composed of separate luminance and chroma signals. A Y/C source will produce a signal that has more resolution than NTSC or PAL. The resulting video image will have more clarity and sharpness.

YUV 4:2:2 – YUV 4:2:0

A common color space used in composite video color systems. **Y** is the luminance component of the monochrome signal while **U** and **V** are the color difference components. **U** is red minus **Y** (**R-Y**), and **V** is blue minus **Y** (**B-Y**). In order to display YUV data on a computer screen, it must be converted into RGB through a process known as "color space conversion." YUV is used because it saves storage space and transmission bandwidth compared to RGB. YUV is not compressed RGB; rather, it is the mathematical equivalent of RGB.

Index

A

- Abort
 - SapAcquisition 33
- Acq
 - SapAcquisition 34
- AcqEventType
 - SapAcquisition 34
- ActiveX Container 3
- ActiveX Control 3, 4, 12, 15
- AllDevices
 - SapAcquisition 35
- AllowPropertyDlg
 - SapAcquisition 36
- AllServers
 - SapAcquisition 36
- Arithmetic
 - SapLut 95
- AutoDisplay
 - SapAcquisition 37
- AutoEmpty
 - SapAcquisition 37

B

- Bayer
 - SapAcquisition 38
- BayerAlign
 - SapAcquisition 39
- BayerDlg
 - SapAcquisition 40
- BayerLut
 - SapAcquisition 40
- BayerLutEnable
 - SapAcquisition 41
- BayerMethod
 - SapAcquisition 41
- BayerMode

- SapAcquisition 42
- BayerWBGain
 - SapAcquisition 42
- BayerWBOffset
 - SapAcquisition 43
- BinaryPattern
 - SapLut 96
- BitsPerPixel
 - SapBuffer 73
- Blue
 - SapDataFRGB 118
 - SapDataRGB 111
- Boolean 16
 - SapLut 96
- Borland Delphi 3, 4
- Buffer
 - SapAcquisition 43
- BytesPerPixel
 - SapBuffer 73

C

- CamExpert 18
- CamSel
 - SapAcquisition 44
- CCF 18
- Clear
 - SapBuffer 73
 - SapDataFRGB 118
 - SapDataHSI 114
 - SapDataHSV 116
 - SapDataMono 110
 - SapDataRGB 111
 - SapDataYUV 113
- ClearBlack
 - SapBuffer 74
- COM objects 9, 11, 15
- CompositeDlg
 - SapAcquisition 44
- ConfigFile
 - SapAcquisition 45
- Count
 - SapBuffer 74
- CounterStamp
 - SapBuffer 75

Cycle
 SapAcquisition 45

D

DALSA Log Viewer 16, 17
DataAddressPtr
 SapBuffer 75
DataBits
 SapDisplay 88
Delphi 7, 13, 14, 16
Demo Programs
 Acquisition Demo (C#) 8
 Acquisition Demo (Delphi 7) 8
 Acquisition Demo (VB.NET) 7
 Acquisition Demo (Visual Basic 6.0) 7
design-time 9, 10, 14
DestROI
 SapDisplay 89
Device
 SapAcquisition 46
Display
 SapAcquisition 46
DLL 3

E

EndReset
 SapAcquisition 47
Entries
 SapLut 97
EntrySize
 SapLut 97
error description 17
error reporting 16
ErrorMode
 SapAcquisition 47
Events
 ActiveX Control 3

F

FlatField
 SapAcquisition 48
FlatFieldDlg
 SapAcquisition 48

FlatFieldGain
 SapAcquisition 49
FlatFieldMode
 SapAcquisition 49
FlatFieldOffset
 SapAcquisition 50
Form Designer 12, 16
Format
 SapBuffer 76
 SapLut 98
Freeze
 SapAcquisition 50

G

Gamma
 SapLut 99
GetCap
 SapAcquisition 51
GetCapEx
 SapAcquisition 51
GetDataArray
 SapBuffer 77
 SapLut 99
GetDataXY
 SapBuffer 77
GetDC
 SapDisplay 89
GetEntry
 SapLut 100
GetParam
 SapAcquisition 52
GetParamEx
 SapAcquisition 52
GetRectDataArray
 SapBuffer 78
Grab
 SapAcquisition 53
Grabbing
 SapAcquisition 53
Green
 SapDataFRGB 118
 SapDataRGB 112

H

H

- SapDataHSI 115
- SapDataHSV 116
- handler 10, 12, 16
- HasTrash
 - SapAcquisition 54
- Height
 - SapBuffer 79
 - SapRectangle 108
- Hide
 - SapDisplay 90
- HorzFlip
 - SapAcquisition 54

I

I

- SapDataHSI 115
- Index
 - SapBuffer 79
- Init
 - SapRectangle 108
- International/Canada Sales Office 121

K

- KeyColor
 - SapDisplay 90

L

- Label
 - SapAcquisition 54
- LastErrorDesc
 - SapAcquisition 55
- LastErrorValue
 - SapAcquisition 55
- Left
 - SapRectangle 109
- Load
 - SapBuffer 79
 - SapLut 100
- LoadConfigDlg

- SapAcquisition 55
- LoadFlatField
 - SapAcquisition 56
- LoadWithDlg
 - SapBuffer 80
- lookup table (LUT) 19, 23
- Lut
 - SapAcquisition 56
- LutEnable
 - SapAcquisition 57

M

- Message
 - SapAcquisition 57
- Methods
 - ActiveX Control 3
- Microsoft Visual Basic 4
- Microsoft Visual Studio .NET 3, 4, 7, 10, 11, 12, 13
- Microsoft Windows 2000 4
- Microsoft Windows XP 4
- Mono
 - SapDataMono 110

N

- Normal
 - SapLut 101
- NumDevices
 - SapAcquisition 58
- NumServers
 - SapAcquisition 58

O

- OLE Container 3
- OLE Control 3
- OnBoardBuffers
 - SapAcquisition 58
- Online
 - SapAcquisition 59
- operation modes
 - board requirements 4
 - minimum system requirements 4

P

- Pages
 - SapLut 101
- Pascal 13
- Pentium 4
- Pitch
 - SapBuffer 81
- Planar
 - SapAcquisition 59
- ProgramLut
 - SapAcquisition 60
- Properties
 - ActiveX Control 3

R

- RAD 3
- Red
 - SapDataFRGB 119
 - SapDataRGB 112
- references folder 13
- region-of-interest (ROI) 21
- ReleaseDC
 - SapDisplay 91
- Reset
 - SapAcquisition 60
- ResetTimeout
 - SapAcquisition 61
- resource allocation 3
- Reverse
 - SapLut 101
- ROI
 - SapBuffer 81
- Roll
 - SapLut 102
- runtime 9, 11, 14, 15

S

- S
 - SapDataHSI 115
 - SapDataHSV 117
- SapControlType
 - SapAcquisition 61

- SapError
 - SapAcquisition 62
- Save
 - SapBuffer 81
 - SapLut 102
- SaveConfig
 - SapAcquisition 62
- SaveFlatField
 - SapAcquisition 63
- SaveWithDlg
 - SapBuffer 82
- Scaling
 - SapDisplay 91
- SerialName
 - SapAcquisition 63
- SerialNumber
 - SapAcquisition 64
- Server
 - SapAcquisition 64
- SetDataArray
 - SapBuffer 83
 - SapLut 103
- SetDataXY
 - SapBuffer 83
- SetEntry
 - SapLut 103
- SetParam
 - SapAcquisition 64
- SetParamEx
 - SapAcquisition 65
- SetRectDataArray
 - SapBuffer 84
- Shift
 - SapLut 104
- Show
 - SapDisplay 91
- ShowProperties
 - SapAcquisition 66
- Signal
 - SapAcquisition 66
- SignalNotify
 - SapAcquisition 67
- SignalStatus
 - SapAcquisition 67
- Signed

- SapLut 104
- Slope
 - SapLut 104
- Snap
 - SapAcquisition 68
- SoftTrigger
 - SapAcquisition 68
- Solution Explorer 10, 11
- SourceROI
 - SapDisplay 92
- SpaceUsed
 - SapBuffer 85
- State
 - SapBuffer 85

T

- Threshold
 - SapLut 105
- Top
 - SapRectangle 109
- TotalSize
 - SapLut 107
- Type
 - SapBuffer 86

U

- U
 - SapDataYUV 113
- US Sales Office 121
- UseROI
 - SapBuffer 87

V

- V
 - SapDataHSV 117
 - SapDataYUV 113
- Valid
 - SapAcquisition 69
 - SapBuffer 87
 - SapDisplay 93
 - SapLut 107
- VertFlip
 - SapAcquisition 69

- Visual Basic 3, 7, 8, 10, 11, 16

W

- Wait
 - SapAcquisition 69
- Width
 - SapBuffer 87
 - SapRectangle 109
- Window
 - SapDisplay 93
- wrapper 13

X

- Xfer
 - SapAcquisition 70
- XferEventType
 - SapAcquisition 71

Y

- Y
 - SapDataYUV 114

Z

- ZoomHorz
 - SapDisplay 93
- ZoomVert
 - SapDisplay 94