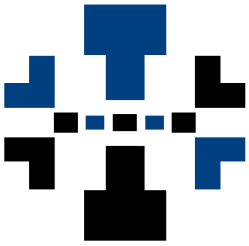


Srego CE ToolPack ActiveX Control

Version 1.0.0.60

User's Manual

Srego, Inc.
www.srego.com

Srego CE ToolPack ActiveX Control

Overview

Introduction

The **Srego CE ToolPack ActiveX** control has been designed to implement the functionality found in the **Srego CE ToolPack** command-line tools. The **Srego CE ToolPack** provides command-line utilities to manipulate a Windows® CE device (including Pocket PC, Handheld PC 2000, and Windows Mobile™ 2003) from the desktop to perform tasks like coping and moving files, creating and deleting directories, and transferring files between the desktop and the device. The command-line utilities are very useful tools, but they were not meant to be incorporated into a custom application since they are all stand-alone executables. However, since it is sometimes desirable to enable an application on the desktop with some of the functionality found in the command-line utilities, the **Srego CE ToolPack ActiveX** control was developed.

The **Srego CE ToolPack ActiveX** control is used on the desktop to perform tasks on or get information from a connected Windows® CE device. The **Srego CE ToolPack ActiveX** control can be used with any development tool that supports ActiveX controls. The **Srego CE ToolPack ActiveX** control uses RAPI (Remote Application Programming Interface) to communicate with the connected device. The device must be connected to the desktop via Microsoft® ActiveSync® before the ActiveX control can be successfully used. Because the **Srego CE ToolPack ActiveX** control using RAPI to communicate with the device, no additional software is required on the device.

The **Srego CE ToolPack ActiveX** control provides a wide range of functionality. Just like the command-line utilities in the **Srego CE ToolPack**, the ActiveX control provides methods to perform the following tasks:

- Move or Copy files on the device from one location to another.
- Delete files on the device.
- Create or Remove directories on the device.
- Get file lists and file information from the device.
- Transfer files between the desktop and the device.
- Synchronize files on the desktop with files on the device and vice versa.
- Update file times (touch files) on the device.

The **Srego CE ToolPack ActiveX** control also provides functionality not found in the command-line tools.

- Registry manipulation on the device.
- CE Database manipulation on the device.
- Device info retrieval such as processor type, battery information, and AC Status.

Requirements

The **Srego CE ToolPack ActiveX** control has the following requirements:

- Windows® 98, Windows® ME, Windows® NT, Windows® 2000, and Windows® XP
- Development Environment supporting ActiveX controls, such as Visual Basic® or Visual C++®
- Windows CE Device running Window CE 2.11 or higher
- Microsoft® ActiveSync® 3.0 or higher

Examples

Two examples are provided in the product directory. These examples demonstrate various aspects of the ActiveX control. The examples are found in the Product directory (such as c:\Program Files\Srego\CE ToolPack ActiveX\Examples)

Installation

The **Srego CE ToolPack ActiveX** control is installed on the desktop. The installation delivers the ActiveX control and documentation. The documentation and the readme.txt file are accessible under the Windows Start button under **Programs/Srego CE ToolPack ActiveX**. The readme.txt file will list all updates, fixes, and changes to the current version of the control.

The default installation directory is:

```
c:\program files\Srego\CE ToolPack ActiveX
```

Support

For product support and product updates, visit:

http://www.srego.com/support

or email:

support@srego.com

For product updates, check the support page on http://www.srego.com/support.

Disclaimer

While **Srego, Inc.** tests its software on many different Windows CE devices, it is not possible to test on every different device. **Srego, Inc.** will try to support all devices using the platforms supported by the application; however, it may not be possible to support all devices.

The **Srego CE ToolPack ActiveX** control can manipulate files, databases, and the registry on a Windows CE device. It is the developer's responsibility to backup all files before using the ActiveX control and to use the ActiveX control in a responsible manner.

The entire risk as to the results and performance of the Software is with the user. Although **Srego, Inc.** has used considerable efforts in preparing the Software, **Srego, Inc.** does not warrant the accuracy or completeness of the Software or Documentation. In no event will **Srego, Inc.** be liable for damages, including loss of profits or consequential damages, arising out of the use of the Software.

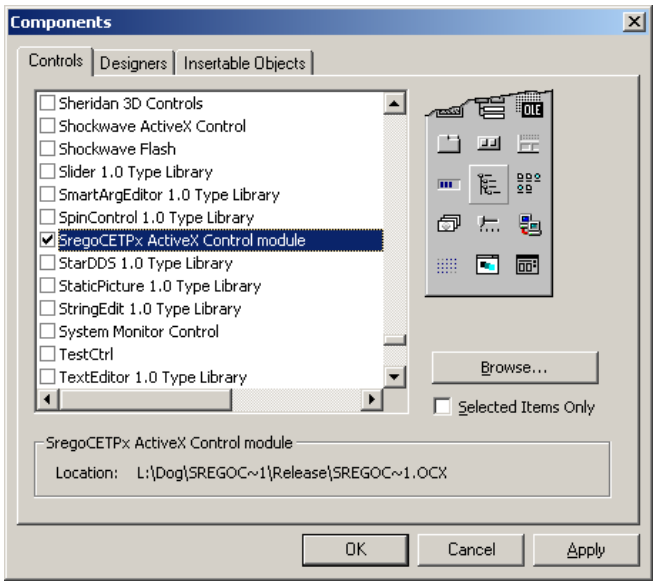
Trademark Notices

Microsoft, Windows Mobile 2003, Pocket PC, Handheld PC 2000, Visual Basic, Visual C++, ActiveX, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Using the Srego CE ToolPack ActiveX Control

While the **Srego CE ToolPack ActiveX** should be usable with any Development Environment supporting ActiveX controls, this document will provide all examples using Visual Basic® 6.0.

In Visual Basic®, you must select the **Srego CE ToolPack ActiveX** control on the Components dialog before the control can be placed on a form. Under Project/Components, find and select **the Srego CE ToolPack ActiveX** control:

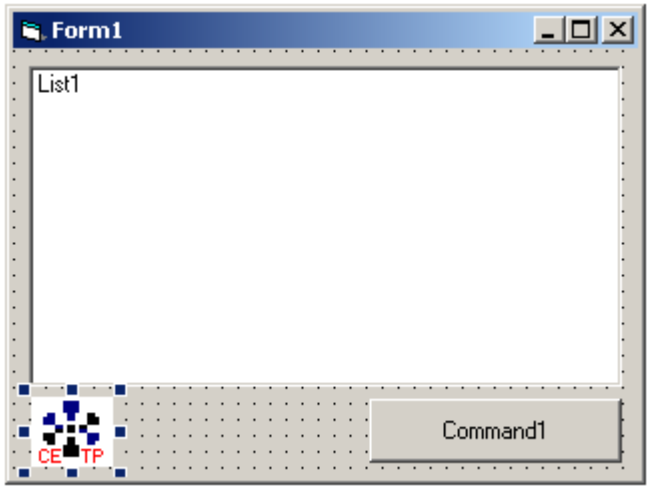


The **Srego CE ToolPack ActiveX** control will appear on the Visual Basic® Toolbox when selected. The control is shown at the bottom right in the toolbox below:

Sample Application

The following example demonstrates how to implement a simple application using the **Srego CE ToolPack ActiveX** control.

- In Visual Basic, create a new Standard EXE project.
- Select the **Srego CE ToolPack ActiveX** control under Project/Components so that it will appear in the ToolBox.
- On a form, place a **Srego CE ToolPack ActiveX** control. The control will not display when the application runs, so it does not matter where it is placed on the form.
- Place a ListBox control and a CommandButton Control on the form as shown below.



- Add the code on the following page.
- Run the application and press the command button.
- The GetFileList method is used to retrieve a list of directories and files from the device's \My Documents directory. The information will be added to the ListBox.
- If the device is not properly connected, an error message will be displayed in the ListBox.

Sample Code:

```
Option Explicit

Private Sub Command1_Click()

    Dim fileList As Variant
    Dim typeList As Variant
    Dim count As Variant
    Dim i As Long

    List1.Clear

    If SregoCETPx1.GetFileList("\my documents\*.*", _
                               fileList, typeList, count) Then

       ' Show all Directories first

        For i = 0 To count - 1

            If typeList(i) = 0 Then  ' directory

                List1.AddItem "[" + fileList(i) + "]"

            End If

        Next

        List1.AddItem ""

       ' Show all Files after directories

        For i = 0 To count - 1

            If typeList(i) = 1 Then ' regular file

                List1.AddItem fileList(i)

            End If

        Next

    Else

        List1.AddItem "Unable to Get File List."
        List1.AddItem "Make sure device is properly connected."

    End If

End Sub
```

Constants

Many of the methods used by the **Srego CE ToolPack ActiveX** control use constant values defined by RAPI. The following list of constants can be used in a module in Visual Basic to simply and clarify coding. This list is also provided in a file called **SregoCETPx.bas** delivered with the ActiveX control's documentation.

```
Const CEVT_I2 = 2
Const CEVT_UI2 = 18
Const CEVT_I4 = 3
Const CEVT_UI4 = 19
Const CEVT_FILETIME = 64
Const CEVT_LPWSTR = 31
Const CEVT_BLOB = 65
Const CEVT_BOOL = 11
Const CEVT_R8 = 5

Const CEDB_NO_AUTOINCREMENT = 0
Const CEDB_AUTOINCREMENT = 1

Const OPEN_EXISTING = 3

Const CEDB_NOCOMPRESS = 65536

Const CEDB_PROPNOTFOUND = 64
Const CEDB_PROPDELETE = 512

Const ERROR_INVALID_PARAMETER = 87
Const ERROR_NO_DATA = 232
Const ERROR_INSUFFICIENT_BUFFER = 122
Const ERROR_KEY_DELETED = 1018
Const ERROR_NO_MORE_ITEMS = 259

Const CEDB_SORT_DEFAULT = 0
Const CEDB_SORT_DESCENDING = 1
Const CEDB_SORT_CASEINSENSITIVE = 2
Const CEDB_SORT_UNKNOWNFIRST = 4
Const CEDB_SORT_GENERICORDER = 8

Const CEDB_SEEK_CEOID = 1
Const CEDB_SEEK_BEGINNING = 2
Const CEDB_SEEK_END = 4
Const CEDB_SEEK_CURRENT = 8
Const CEDB_SEEK_VALUESMALLER = 16
Const CEDB_SEEK_VALUEFIRSTEQUAL = 32
Const CEDB_SEEK_VALUEGREATER = 64
Const CEDB_SEEK_VALUENEXTEQUAL = 128

Const ERROR_DISK_FULL = 112
Const ERROR_DUP_NAME = 52

Const HKEY_CLASSES_ROOT = 0
Const HKEY_CURRENT_USER = 1
Const HKEY_LOCAL_MACHINE = 2
Const HKEY_USERS = 3

Const PROCESSOR_INTEL_386 = 386
Const PROCESSOR_INTEL_486 = 486
Const PROCESSOR_INTEL_PENTIUM = 586
Const PROCESSOR_INTEL_860 = 860
Const PROCESSOR_MIPS_R2000 = 2000
```

```
Const PROCESSOR_MIPS_R3000 = 3000
Const PROCESSOR_MIPS_R4000 = 4000
Const PROCESSOR_HITACHI_SH3 = 10003
Const PROCESSOR_HITACHI_SH4 = 10005
Const PROCESSOR_ALPHA_21064 = 21064
Const PROCESSOR_PPC_601 = 601
Const PROCESSOR_PPC_603 = 603
Const PROCESSOR_PPC_604 = 604
Const PROCESSOR_PPC_620 = 620
Const PROCESSOR_PPC_821 = 821
Const PROCESSOR_SHx_SH3 = 103
Const PROCESSOR_SHx_SH4 = 104
Const PROCESSOR_STRONGARM = 2577
Const PROCESSOR_ARM720 = 1824
Const PROCESSOR_ARM820 = 2080
Const PROCESSOR_ARM920 = 2336

Const BATTERY_CHG_HIGH = 1
Const BATTERY_CHG_LOW = 2
Const BATTERY_CHG_CRITICAL = 4
Const BATTERY_CHG_CHARGING = 8
Const BATTERY_CHG_NO_SYSTEM_BATTERY = 128
Const BATTERY_CHG_UNKNOWN_STATUS = 255

Const AC_LINE_DISCONNECTED = 0
Const AC_LINE_CONNECTED = 1
Const AC_LINE_STATUS_UNKNOWN = 255
```

Distribution

The **Srego CE ToolPack ActiveX** control can be distributed with your application royalty-free as long as it is delivered in its runtime form. The developer serial number must **NOT** be distributed with your application.

It is recommended that the **SregoCETPx.ocx** and **SregoCETPx.lic** files both be delivered to the Windows® System 32 directory (c:\windows\system32, c:\winnt\system32, etc.). The **SregoCETPx.ocx** must be registered. It is possible to delivery the .ocx to any directory; however, if you have multiple applications using the same .ocx, the last one registered will be used which could cause problem with the other applications. If the .ocx is always delivery to the Windows System 32 directory and only delivered if the .ocx is newer than the existing one (if any), there should be no conflicts.

The **SregoCETPx.ocx** also requires the **rapi.dll** and **ceutil.dll**. These .dlls are delivered with Microsoft® ActiveSync® which is a requirement to use the **Srego CE ToolPack ActiveX** control.

General Methods

The methods in this section deal with the general operation of the ActiveX control such as establishing and terminating a connection between the desktop and the device and retrieving error information.

SetSerialNumber

With version 1.0.0.51 and after, the Srego CE ToolPack ActiveX control uses a method in the code to set the serial number rather than the .lic file or the registry. This change allows more development platforms to support the ActiveX control since some development platforms do not support the design-time and runtime modes of ActiveX controls.

The method must be called with a valid serial number or the control will run in Evaluation mode only. This method needs to only be called once.

Syntax:

```
boolean SetSerialNumber(BSTR serialNumber)
```

Return Value:

Returns TRUE for Success and FALSE otherwise.

Example:

```
SregoCETPx1.SetSerialNumber "1234567890"
```

InitializeConnection

This method can be used to initialize a RAPI connection to the connected device. All methods requiring a RAPI connection will automatically initialize the connection if one has not already been established, so it is not necessary to explicitly initialize the connection to the device.

This method can be used to test if a connection can be made to a device without actually performing any task. The connection with the device will be disconnected automatically when the ActiveX control is terminated; however, you can use the **TerminateConnection** method to terminate the connection at any time.

If multiple **Srego CE TookPack ActiveX** controls are used in the same application, they will all share the same RAPI connection to the device.

The use of the **InitializeConnection** and **TerminateConnection** methods are typically not necessary unless there are very specific needs to test for a connection without performing a task or to explicitly terminate a connection. For typical use, the management of the RAPI connection is handled automatically by the ActiveX control greatly simplifying the use of the methods.

Syntax:

```
boolean InitializeConnection()
```

Return Value:

The return value is True if the connection is successfully made and False if not. If the return value is False, you can use **GetErrorMessage** and **GetLastCeErrorMessage** to get more details on why the connection failed.

TerminateConnection

This method can be used to terminate a RAPI connection with the device. It is not necessary to call this method since the RAPI connection will be terminated automatically when the ActiveX control is destroyed.

If multiple Srego CE TookPack ActiveX controls are used in the same application, they will all share the same RAPI connection to the device. The **TerminateConnection** method will terminate the shared connection; however, each method requiring a RAPI connection will automatically create a connection if one does not exists.

The use of the **InitializeConnection** and **TerminateConnection** methods are typically not necessary unless there are very specific needs to test for a connection without performing a task or to explicitly terminate a connection. For typical use, the management of the RAPI connection is handled automatically by the ActiveX control greatly simplifying the use of the methods.

Syntax:

```
boolean TerminateConnection()
```

Return Value:

The return value is True if successful and False otherwise.

GetVersion

The **GetVersion** method will return the version of the **Srego CE ToolPack ActiveX** control.

Syntax:

```
BSTR GetVersion()
```

Return Value:

The method returns a string containing the version of the ActiveX control.

GetErrorMessage

This method returns current error message in the error buffer. This method can be called repeatedly to get detailed error information. The **ResetErrorMessage** method can be used to clear the error buffer; however, each new method that can populate the error buffer will automatically clear the buffer when the method is called.

Syntax:

```
BSTR GetErrorMessage()
```

Return Value:

The method returns the oldest error message in the buffer. When the error buffer is empty, the method returns an empty string.

ResetErrorMessages

This method clears the error message buffer.

Syntax:

```
boolean ResetErrorMessages()
```

Return Value:

The method returns True for success and False otherwise.

GetLastCeErrorMessage

This method returns detailed error information from the device after an error has occurred. Since many of the ActiveX control's methods are wrappers around RAPI calls, it is often useful to get the detailed error message provided by the device. This method should be called immediately after an error has occurred or the results may not be accurate.

Syntax:

```
BSTR GetLastCeErrorMessage()
```

Return Value:

The method returns the last error message reported by the Device.

GetLastCeErrorNum

This method returns the error code from the device after an error has occurred. This method is similar to **GetLastCeErrorMessage** only it returns the error number instead of the error message.

Syntax:

```
long GetLastCeErrorNum()
```

Return Value:

The method returns the last error number reported by the Device.

SetLocale

This method explicitly sets the locale for the ActiveX. This method may sometimes be necessary for converting strings between Unicode and Multibyte.

Syntax:

```
BSTR SetLocale( long category, BSTR locale )
```

Category must be one of the following:

LC_TYPE = 2

Return Value:

Returns the locale string that was set.

Example:

```
' sets to default
List1.AddItem SregoCETPx1.SetLocale(2, "")

' Sets to a specific locale
List1.AddItem SregoCETPx1.SetLocale(2, "English_United States.1252")
```

Connection Event

ConnectionEvent

This event can be used to monitor the connection between the desktop and the device. The **ConnectionEvent** event is fired every time the status of the device's connection changes. When the **Srego CE ToolPack ActiveX** control is first initialized, it will fire an event to indicate the current connection status. If the connection events are tracked, the status of the device's connection can always be known.

Syntax:

```
void ConnectionEvent(long eventType, BSTR message)
```

The **eventType** parameter can be one of the following values:

- **CONNECTION_EVENT_ACTIVE** (0) – indicates that a connection exists between the desktop and the device. This event is fired when the ActiveX control is initialized, if a connection already exists. Capturing this event first when an application is run will indication that a connection already exists; if no connection exists, the CONNECTION_EVENT_LISTEN event will be fired instead. Capturing this event any other time will indicate that a connection was established between the desktop and the device. The CONNECTION_EVENT_IP_ADDRESS event is sent after this event to indicate the device's IP address.

- **CONNECTION_EVENT_ANSWERED** (1) – indicates that the Windows® CE communications manager has detected a communications interface.

- **CONNECTION_EVENT_DISCONNECTION** (2) – indicates that the connection between the desktop and the device has been terminated. Capturing this event will indicate that the connection between the desktop and the device has been terminated.

- **CONNECTION_EVENT_ERROR** (3) – indicates that the communication between the desktop and the device failed to start.

- **CONNECTION_EVENT_INACTIVE** (4) – indicates that there has been a disconnection or that there is a disconnected state between the desktop and the device.

- **CONNECTION_EVENT_IP_ADDRESS** (5) – indicates that a connection between the desktop and the device was established and that an IP address has been assigned to the device. The message parameter will contain the IP address. If the device is connected via a USB or Serial cable, the IP address will be

127.0.0.1 (local hosts). If the device is connected via a network connection, the IP address will be the device's assigned IP address. This event is fired immediately after the CONNECTION_EVENT_ACTIVE event.

- **CONNECTION_EVENT_LISTEN** (6) – indicates that a connection does not exist between the desktop and device and that the connection monitor is listening for a status change. Capturing this event when the ActiveX control is initialized will indicate that there is no connection between the desktop and the device.

- **CONNECTION_EVENT_TERMINATED** (7) – indicates that the Windows® CE connection manager has been terminated.

The **message** parameter is only used by the CONNECTION_EVENT_IP_ADDRESS event where it will contain a string holding the IP address. All other events will pass an empty string for this parameter.

Example:

```
Private Sub SregoCETPx1_ConnectionEvent(ByVal eventType As Long, ByVal message As String)

    Dim eventMessage As String

    eventMessage = ""

    Select Case eventType

        Case CONNECTION_EVENT_ACTIVE:  eventMessage = "Active"

        Case CONNECTION_EVENT_ANSWERED: eventMessage = "Answered"

        Case CONNECTION_EVENT_DISCONNECTION: eventMessage = "Disconnection"

        Case CONNECTION_EVENT_ERROR: eventMessage = "Error"

        Case CONNECTION_EVENT_INACTIVE: eventMessage = "Inactive"

        Case CONNECTION_EVENT_IP_ADDRESS: eventMessage = "IP Address: " + message

        Case CONNECTION_EVENT_LISTEN: eventMessage = "Listen"

        Case CONNECTION_EVENT_TERMINATED: eventMessage = "Terminated"

    End Select

    List1.AddItem "Connection Event: " + eventMessage

End Sub
```

ConnectionEventAlt

The **ConnectionEventAlt** event is identical to the **ConnectionEvent** only the **message** parameter has been renamed to **eventMsg** to prevent a conflict in PowerBuilder.

Syntax:

```
void ConnectionEventAlt(long eventType, BSTR eventMsg)
```

Device Info Methods

The methods in this section will retrieve information about the device.

GetDeviceProcessorType

The **GetDeviceProcessorType** method returns the process type of the connected device.

Syntax:

```
long GetDeviceProcessorType()
```

Return Value:

This method returns 0 if the call is unsuccessful. If success, the value will be one of the following:

PROCESSOR_INTEL_386 (386)
PROCESSOR_INTEL_486 (486)
PROCESSOR_INTEL_PENTIUM (586)
PROCESSOR_INTEL_860 (860)
PROCESSOR_MIPS_R2000 (2000)
PROCESSOR_MIPS_R3000 (3000)
PROCESSOR_MIPS_R4000 (4000)
PROCESSOR_HITACHI_SH3 (10003)
PROCESSOR_HITACHI_SH4 (10005)
PROCESSOR_ALPHA_21064 (21064)
PROCESSOR_PPC_601 (601)
PROCESSOR_PPC_603 (603)
PROCESSOR_PPC_604 (604)
PROCESSOR_PPC_620 (620)
PROCESSOR_PPC_821 (821)
PROCESSOR_SHx_SH3 (103)
PROCESSOR_SHx_SH4 (104)
PROCESSOR_STRONGARM (2577)
PROCESSOR_ARM720 (1824)
PROCESSOR_ARM820 (2080)
PROCESSOR_ARM920 (2336)

Example:

```
Dim procType As Long
procType = SregoCETPx1.GetDeviceProcessorType

If procType = 0 Then

    MsgBox "Unable to get device's processor type."

Else
```

```
    MsgBox "Device Processor Type: " + Str(procType)

End If
```

GetBatteryInfo

The **GetBatteryInfo** method returns information about the device's battery status.

Syntax:

```
boolean GetBatteryInfo(VARIANT* batteryFlag,
                       VARIANT* batteryLifePercent,
                       VARIANT* batteryLifeTime,
                       VARIANT* batteryFullLifeTime)
```

- **batteryFlag** – returns a variant containing the battery charge status. The value can be a combination of the following values:

  BATTERY_CHG_HIGH (1)
  BATTERY_CHG_LOW (2)
  BATTERY_CHG_CRITICAL (4)
  BATTERY_CHG_CHARGING (8)
  BATTERY_CHG_NO_SYSTEM_BATTERY (128)
  BATTERY_CHG_UNKNOWN_STATUS (255)

- **batteryLifePercent** – returns a variant containing the percentage of battery charge remaining. The value will be from 0 to 100 or 255 if the status is unknown.

- **batteryLifeTime** – returns a variant containing the time in seconds of battery life remaining. Some devices may not support this value and will return 0 or -1.

- **batteryFullLifeTime** – returns a variant containing the time in seconds of battery life with a full charge. Some devices may not support this value and will return 0 or -1.

Return Value:

The method returns True for success and False otherwise.

Example:

```
Dim batteryFlag As Variant
Dim batteryLifePercent As Variant
Dim batteryLifeTime As Variant
Dim batteryLifeFullTime As Variant

If SregoCETPx1.GetBatteryInfo(batteryFlag, batteryLifePercent, _
             batteryLifeTime, batteryLifeFullTime) Then

    List1.AddItem "Battery Flag: " + Str(batteryFlag)
    List1.AddItem "Battery Life Percent: " + _
                    Str(batteryLifePercent)


End If
```

GetACLineInfo

The **GetACLineInfo** method returns information about the devices AC Line status. The AC Line is either connected or disconnected.

Syntax:

```
long GetACLineInfo()
```

Return Value:

The method returns one of the following values:

-1 – Unable to connect to device.
-2 – Unable to get AC Line Status

AC_LINE_DISCONNECTED (0)
AC_LINE_CONNECTED (1)
AC_LINE_STATUS_UNKNOWN (255)

Example:

```
Dim acLineStatus As Long

  acLineStatus = SregoCETPx1.GetACLineInfo

  Dim buffer As String

  Select Case acLineStatus

      Case AC_LINE_CONNECTED:
          buffer = "Connected"

      Case AC_LINE_DISCONNECTED:
          buffer = "Disconnected"

      Case AC_LINE_STATUS_UNKNOWN:
          buffer = "Unknown"

      Case -1:

          buffer = "Unable to connect to device"

      Case -2:

          buffer = "Unable to get AC Line status"

  End Select

  List1.AddItem "AC Line Status: " + buffer
```

GetMemoryInfo

The **GetMemoryInfo** method returns the device's current physical and available memory. The memory sizes are returned in bytes

Syntax:

```
boolean GetMemoryInfo(VARIANT* total,
                      VARIANT* available)
```

- **total** – a variant that returns the total size of the physical memory in bytes.
- **available** – a variant that returns the amount of mememory available in bytes.

Return Value:

The method returns TRUE if successful and FALSE otherwise.

Example:

```
Dim total As Variant
Dim available As Variant

If SregoCETPx1.GetMemoryInfo(total, available) Then

   List1.AddItem "Total Device Memory: " + Str(total) + " bytes"
   List1.AddItem "Available Device Memory: " + _
                  Str(available) + " bytes"

Else

   MsgBox "Unable to get memory info from device."

End If
```

GetStorageInfo

The **GetStorageInfo** method returns the device's current storage size and available storage. The sizes are returned in bytes.

Syntax:

```
boolean GetStorageInfo(VARIANT* storageSize,
                       VARIANT* freeSpace)
```

- **storageSize** – a variant that returns the size of the storage space in bytes.
- **freeSpace** – a variant that returns the size of the storage space that is free in bytes.

The freeSpace values is always 24K bytes less than what is really available because this memory is reserved for system operations.

Return Value:

The method returns TRUE if successful and FALSE otherwise.

Example:

```
Dim storageSpace As Variant
Dim freeSpace As Variant
```

```
If SregoCETPx1.GetStorageInfo(storageSpace, freeSpace) Then

    Debug.Print storageSpace
    Debug.Print freeSpace


End If
```

GetSystemMetric

The **GetSystemMetric** method returns one of the device's system metrics specified by the index parameter. While there are several metrics this method can return, the most useful are the screen width and height.

Syntax:

```
long GetSystemMetric(long index)
```

The index parameter can be any of the valid system metric indexes. Two of the useful ones are shown below:

SYSMET_CXSCREEN (0) – screen width in pixels
SYSMET_CYSCREEN (1) – screen height in pixels

Return Value:

The method returns -1 if the device is not connected, 0 if there was an error, or the requested system metric.

Example:

```
Dim width As Long
Dim height As Long

width = SregoCETPx1.GetSystemMetric(SYSMET_CXSCREEN)
height = SregoCETPx1.GetSystemMetric(SYSMET_CYSCREEN)
```

GetStorageCardList

The **GetStorageCardList** method returns a list of the device's storage card directories.

Syntax:

```
boolean GetStorageCardList(VARIANT* storageCardList,
                           VARIANT* count);
```

The **storageCardList** parameter will return a variant array of string values containing the names of the storage card directories.

The **count** parameter will return a variant long containing the number of items in the array.

Return Value:

The method returns False if the device is not connected, and true otherwise.

Example:

```
Dim storageCardList As Variant
Dim count As Variant

Dim i As Long

List1.Clear

If SregoCETPx1.GetStorageCardList(storageCardList, count) Then

    For i = 0 To count - 1

        List1.AddItem storageCardList(i)

    Next

End If
```

GetStorageCardListToBuffer

The **GetStorageCardListToBuffer** method store a list of device's storage card directories in the buffer. This method is identical in functionality to the **GetStorageCardList** method with the exception that the data is stored in the buffer instead of being returned in variant arrays. The buffer methods, **GetBufferCount** and **GetBufferValue**, are used to retrieve the buffer information. This method can be used to more easily return complex data with development platforms where it is difficult to deal with variant arrays.

Syntax:

```
long GetStorageCardListToBuffer();
```

Return Value:

The method returns -1 if the device is not connected or the number of items stored in the buffer.

Example:

```
Dim storageCardList As Variant
Dim count As Variant

Dim i As Long

count = SregoCETPx1.GetStorageCardListToBuffer

List1.Clear

For i = 0 To count - 1

    List1.AddItem SregoCETPx1.GetBufferValue(0, i)

Next
```

GetOperatingSystemVersion

The **GetOperatingSystemVersion** method returns the specific version number of the device's operating system. The Major Version, Minor Version, and Build (or Revision) are all returned as longs. The Build value is somewhat misleading in that it is the third values in a Windows CE operating system version: <Major>.<Minor>.<Build>. For example, Windows Mobile 5.0 may give: 5.1.70. If you look in the System/About for the Device, you will usually see the 3 digit version following by a Build number in parentheses. This Build number is different from the Build (or revision) returned by this method; they are just both confusingly called Build.

Syntax:

```
boolean GetOperatingSystemVersion(long* major,
                                  long* minor,
                                  long* build)
```

Return Value:

The method returns TRUE is successful and FALSE otherwise.

Example:

```
Sub GetOperatingSystem()

    Dim major As Long
    Dim minor As Long
    Dim build As Long

    Dim buffer As String

    buffer = ""

    If SregoCETPx1.GetOperatingSystemVersion(major, minor, build) Then

        buffer = Trim(Str(major)) + "." + Trim(Str(minor)) + _
                 " (Build " + Trim(Str(build)) + ")"
    End If

    OSLabel.Text = buffer

End Sub
```

Buffer Methods

The methods in this section will manipulate and retrieve data from the buffer. To help simplify developing with platforms that do not readily support the Variant data type (especially Variant arrays), the buffer methods can be used to return a scalar String value after another methods stores lists in the buffer. Think of the buffer as a 2 dimensional array of string values (rows and columns). Depending on the method used to populate the buffer, the columns (identified with the buffer id) will be used to store different kinds of data and the rows will contains the different items for each type.

GetBufferCount

The **GetBufferCount** method returns the number of rows in the buffer.

Syntax:

```
long GetBufferCount()
```

Return Value:

The method returns the number of rows in the buffer.

GetBufferValue

The **GetBufferValue** method returns a single string value from the buffer.

Syntax:

```
BSTR GetBufferValue(long id, long pos)
```

- **id** – specified the buffer id or column of data to return from. The type of data stored in each buffer column is dependent on the method used to populate the buffer..
- **pos** – specifies the row of the buffer column to return a value for.

Return Value:

The method returns the specified buffer value if the id and pos are valid and an empty string otherwise.number of rows in the buffer.

Example:

```
Dim fileList As Variant
Dim typeList As Variant
Dim count As Variant
Dim i As Long

If SregoCETPx1.GetFileListEx("\my documents\*.*", _
                             fileList, typeList, count, "/S /NS") Then

    For i = 0 To count - 1

        If typeList(i) = 0 Then ' directory

            List1.AddItem Str(i) + " - Dir: " + fileList(i)

        ElseIf typeList(i) = 1 Then ' file

            List1.AddItem Str(i) + " - File: " + fileList(i)

        End If

    Next

Else

    MsgBox "Unable to get filelist."

End If
```

ResetBuffer

The **ResetBuffer** method clears the contents of the buffer. It is not necessary to clear the buffer unless memory needs to be returned to the system immediately. All methods that populate the buffer will automatically reset the buffer contents when they are called.

Syntax:

```
void ResetBuffer()
```

ByteBufferClear

This method clears an internal buffer of byte values. The ByteBufferAdd method is used to add values to the array. Other methods, like RegSetBinaryValueFromByteBuffer will use the Byte Buffer.

Syntax:

```
void ByteBufferClear()
```

ByteBufferAddValue

This method will add a value to the byte buffer stored in memory. The byte buffer can be cleared with ByteBufferClear. Methods like RegSetBinaryValueFromByteBuffer will use the internal buffer. In some development platforms, it is difficult to pass variant arrays, so this approach eliminates this problem even if it is not the most elegant solution. The value parameters should be a value from 0 to 255.

Syntax:

```
void ByteArrayAddValue( long value )
```

File Methods

The methods in this section will manipulate files on the device, transfer files between the device, and return information about files on the device.

GetFileList

The **GetFileList** method returns a list of files matching the specified path. The path can contain wildcards (? and/or *). This method is similar to the command-prompt DIR command.

Syntax:

```
boolean GetFileList(   BSTR path,
                       VARIANT* fileList,
                       VARIANT* typeList,
                       VARIANT* count )
```

- **Path** – the search path to return the files for. The path can contain wildcards (* and/or ?).
- **fileList** – a variant that returns an array of string values containing the names of the files matching the search path.
- **typeList** – a variant that returns an array of long values containing the type of the files matching the search path. The type value is 0 for a directory and 1 for a file.
- **count** – a variant that returns a long value specifying how many files are in the arrays.

Return Value:

The method returns True for success and False otherwise.

Example:

```
Dim fileList As Variant
Dim typeList As Variant
Dim count As Variant
Dim i As Long

If SregoCETPx1.GetFileList("\my documents\*.*", _
                           fileList, typeList, count) Then

    For i = 0 To count - 1

        If typeList(i) = 0 Then ' directory

            List1.AddItem Str(i) + " - Dir: " + fileList(i)

        ElseIf typeList(i) = 1 Then ' file

            List1.AddItem Str(i) + " - File: " + fileList(i)
```

```
        End If

    Next

Else

    MsgBox "Unable to get filelist."

End If
```

GetFileListEx

The **GetFileListEx** method returns a list of files matching the specified path. The path can contain wildcards (? and/or *). This method is similar to the command-prompt DIR command. The **GetFileListEx** is an extension of the **GetFileList** method; the options parameter has been added to allow more features (specifically to recursively search the subdirectories).

Syntax:

```
boolean GetFileListEx(BSTR path,
                      VARIANT* fileList,
                      VARIANT* typeList,
                      VARIANT* count,
                      BSTR options)
```

- **Path** – the search path to return the files for. The path can contain wildcards (* and/or ?).
- **fileList** – a variant that returns an array of string values containing the names of the files matching the search path.
- **typeList** – a variant that returns an array of long values containing the type of the files matching the search path. The type value is 0 for a directory and 1 for a file.
- **count** – a variant that returns a long value specifying how many files are in the arrays.
- **Options** – a string that specifies the options. If set to blank (“”), the method is equivalent to the **GetFileList** method. The option can be one of the following values separated by a space:

  /S - recursively search all subdirectories with the search criteria
  /NS – do not add subdirectory name to the return value lists
  /NF – do not add file names to the return value lists

  Using the /NS and /NF options together will never return any items in the lists.

Return Value:

The method returns True for success and False otherwise.

Example:

```
Dim fileList As Variant
Dim typeList As Variant
Dim count As Variant
Dim i As Long

If SregoCETPx1.GetFileListEx("\my documents\*.*", _
                             fileList, typeList, count, "/S /NS") Then

    For i = 0 To count - 1

        If typeList(i) = 0 Then ' directory

            List1.AddItem Str(i) + " - Dir: " + fileList(i)

        ElseIf typeList(i) = 1 Then ' file

            List1.AddItem Str(i) + " - File: " + fileList(i)

        End If

    Next

Else

    MsgBox "Unable to get filelist."

End If
```

GetFileListToBuffer

The **GetFileListToBuffer** method stores a list of files matching the specified path to the control's buffer. This method is identical in functionality to the **GetFileListEx** method with the exception that the data is stored in the buffer instead of being returned in variant arrays. The buffer methods, **GetBufferCount** and **GetBufferValue**, are used to retrieve the buffer information. This method can be used to more easily return complex data with development platforms where it is difficult to deal with variant arrays.

The file path list will be stored in Buffer 0 and the type list will be stored in Buffer 1. All buffer values are stored as strings, so the type list is a "0" for directories and "1" for files.

Syntax:

```
long GetFileListToBuffer( BSTR path,
                          BSTR options)
```

- **Path** – the search path to return the files for. The path can contain wildcards (* and/or ?).

- **Options** – a string that specifies the options. If set to blank (“”), the method is equivalent to the **GetFileList** method. The option can be one of the following values separated by a space:

  /S - recursively search all subdirectories with the search criteria
  /NS – do not add subdirectory name to the return value lists
  /NF – do not add file names to the return value lists

  Using the /NS and /NF options together will never return any items in the lists.

Return Value:

The method returns the number of items stored in the buffer or -1 if a problem was encountered.

Example:

```
Dim filePath As String
Dim fileType As String
Dim count As Long


count = SregoCETPx1.GetFileListToBuffer("\my documents\*.*", "/s")

Dim i As Long

List1.Clear

For i = 0 To count - 1

    filePath = SregoCETPx1.GetBufferValue(0, i) ' buffer 0 for path
    fileType = SregoCETPx1.GetBufferValue(1, i) ' buffer 1 for type

    If fileType = "0" Then ' directory
        List1.AddItem "Dir: " + filePath
    ElseIf fileType = "1" Then ' file
        List1.AddItem "       File: " + filePath
    End If

Next
```

CompareDirectores

The **CompareDirectories** method is a powerful method for determining differences between the files in a directory on the desktop and those in a directory on the device. This method can simplify any file synchronization tasks when combined with the **PushFile**, **PullFile**, and **DeleteFile** methods.

Syntax:

```
boolean CompareDirectories(BSTR desktopPath,
                           BSTR devicePath,
                           BSTR filter,
                           BSTR options,
                           VARIANT* fileList,
                           VARIANT* statusList,
                           VARIANT* count)
```

- desktopPath – this parameter must be a valid directory name on the desktop. The path cannot contain wildcards or a filename.
- devicePath – this parameter must be a valid directory on the device. The path cannot contain wildcards or a filename.
- filter – this parameter specifies the search filter to use in the comparison. The filter must be specified. If all files need to be compared, use "*.*". The filter can use the * and ? wildcard or be a specific filename.
- options – the options parameter can contain a space delimited list of option flags. The options string can be blank ("") or one or more of the following:

  /S – recursively search all subdirectories
  /0 – do not return any files that are the same (status 0)
  /1 – do not return any files that are only on the desktop (status 1)
  /2 – do not return any files that are only on the device (status 2)
  /3 – do not return any files that are newer on the desktop (status 3)
  /4 – do not return any files that are newer on the device (status 4)

  By default, all files will be return. The /# flags can be used to filter the return list.

- **fileList** – a variant that returns an array of string values containing the full path name to the files compared (minus either the desktop search path and/or the device search path).
- **statusList –** a variant that returns an array of long values containing the status of each file returned in the fileList. The status can be one of the following values:

  0 – the files are the same (name and date)
  1 – the file is only on the desktop
  2 – the file is only on the device
  3 – the file is newer on the desktop
  4 – the file is newer on the device

- **count** – a variant that returns a long value containing the number of items in the lists.

The **CompareDirectories** method can take an unacceptable amount of time and may consume all available resources if too many files are compared (such as comparing the root directory on a large disk on the desktop to the root directory on the device).

If the **/S** option is not specified, only the file if the paths specified will be compared.

Example:

```
Dim fileList As Variant
Dim statusList As Variant
Dim count As Variant

If SregoCETPx1.CompareDirectories("c:\temp\test", "\my documents\test", _
                                  "*.*", "/s", fileList, statusList, count) Then


    Dim i As Long
    Dim buffer As String

    List1.Clear

    For i = 0 To count - 1

        buffer = "File: " + fileList(i) + "  Status: "

        Select Case statusList(i)

            Case 0: buffer = buffer + "Same"

            Case 1: buffer = buffer + "Only on Desktop"

            Case 2: buffer = buffer + "Only on Device"

            Case 3: buffer = buffer + "Newer on Desktop"

            Case 4: buffer = buffer + "Newer on Device"

        End Select


        List1.AddItem buffer
    Next

End If
```

The following example will only return files that are on the device:

```
If SregoCETPx1.CompareDirectories("c:\temp\test", "\my documents\test", _
                                  "*.*", "/s /0 /1 /3 /4", fileList, _
                                  statusList, count) Then
    .
    .
    .


End If
```

MoveFile

The **MoveFile** method will move one or more files from one location on the device to another location on the device. To move multiple files, the source file can contain wildcards (? and/or *). If moving a single file, the destination file can specify a different filename for the moved file; otherwise, the destination file must be a directory.

Syntax:

```
boolean MoveFile(BSTR sourceFile,
                 BSTR destinationFile,
                 boolean overwriteFlag)
```

- **sourceFile –** the path on the device of the file(s) to move. The path can contain wildcards (* and/or ?).
- **destinationFile** – the destination path on the device to place the moved files.
- **overwriteFlag** – if set to FALSE, the files will not be overwritten in the destination path if they already exists. If set to TRUE, the files will be overwritten in the destination path if they already exists.

Return Value:

The method returns True for success and False otherwise.

Events:

The **MoveFile** method fires several events while it is moving files to help monitor its progress:

- `MoveInfo`
- `PreFileMove`
- `PostFileMove`

Example:

```
SregoCETPx1.MoveFile "\my documents\test1\sample.txt", _
                     "\my documents\test2", True


SregoCETPx1.MoveFile "\my documents\test1\*.txt", _
                     "\my documents\test2", True
```

A more detailed example of moving files with the **MoveFile** method is shown below. This example utilizes the events fired by **MoveFile** to display the progress with ListBox and ProgressBar controls:

```
Private Sub MoveCommand_Click()

    OverallProgressBar.Visible = True

    List1.Clear

    If SregoCETPx1.MoveFile("\my documents\test1\*.dgn", _
                   "\my documents\test2", True) Then

        List1.AddItem "File Move Completed Successfully."

    Else

        List1.AddItem "Unable to Move files."

    End If

    OverallProgressBar.Visible = False

End Sub


Private Sub SregoCETPx1_MoveInfo(ByVal fileCount As Long)

    List1.AddItem "Files to Move: " + Str(fileCount)
    List1.AddItem ""

    OverallProgressBar.Value = 0
    OverallProgressBar.Min = 0

    If fileCount > 0 Then
        OverallProgressBar.Max = fileCount
    Else
        OverallProgressBar.Max = 1
    End If

    DoEvents

End Sub


Private Sub SregoCETPx1_PreFileMove(ByVal id As Long, _
                ByVal sourceFile As String, _
                ByVal destFile As String, ByVal size As Double)
```

```
    ' Event fired before each individual file is moved.

      List1.AddItem "File: " & Str(id)
      List1.AddItem "Source: " & sourceFile
      List1.AddItem "Destination: " & destFile
      List1.AddItem "Size: " & Str(size) & " bytes"

End Sub

Private Sub SregoCETPx1_PostFileMove(ByVal id As Long, _
                  ByVal sourceFile As String, _
                  ByVal destFile As String, _
                  ByVal size As Double, ByVal status As Long)

    ' Event Fired after each individual file is moved.

    If status = 1 Then  ' file successfully copied

        List1.AddItem "File Successfully Moved."

    ElseIf status = 2 Then  ' file exists and was not overwritten

        List1.AddItem "File exists and was not overwritten."

    Else

        List1.AddItem "File Move Failed. "

    End If

    List1.AddItem ""

    OverallProgressBar.Value = id + 1

    DoEvents

End Sub
```

CopyFile

The **CopyFile** method will copy one or more files from one location on the device to another location on the device. To copy multiple files, the source file can contain wildcards (? and/or *). If copying a single file, the destination file can specify a different filename to rename the copied file; otherwise, the destination file must be a directory.

Syntax:

```
boolean CopyFile(BSTR sourceFile,
                 BSTR destinationFile,
                 boolean overwriteFlag)
```

- **sourceFile –** the path on the device of the file(s) to copy. The path can contain wildcards (* and/or ?).
- **destinationFile** – the destination path on the device.

- **overwriteFlag** – if set to FALSE, the files will not be overwritten in the destination path if they already exists. If set to TRUE, the files will be overwritten in the destination path if they already exists.

Return Value:

The method returns True for success and False otherwise.

Events:

The **CopyFile** method fires several events while it is copying files to help monitor its progress:

- CopyInfo
- PreFileCopy
- PostFileCopy

Example:

```
SregoCETPx1.CopyFile "\my documents\test1\sample.txt", _
                     "\my documents\test2", True

SregoCETPx1.CopyFile "\my documents\test1\*.txt", _
                     "\my documents\test2", True
```

A more detailed example of copying files with the **CopyFile** method is shown below. This example utilizes the events fired by **CopyFile** to display the progress with ListBox and ProgressBar controls:

```
Private Sub CopyFileCommand_Click()

    List1.Clear
    OverallProgressBar.Visible = True

    If SregoCETPx1.CopyFile("\my documents\test1\*.txt", _
                            "\my documents\test2", True) Then

        List1.AddItem "File Copy Completed Successfully."

    Else

        List1.AddItem "Unable to Copy files."

    End If

    OverallProgressBar.Visible = False

End Sub

Private Sub SregoCETPx1_CopyInfo(ByVal fileCount As Long)

    ' Event fired when CopyFile starts so you will no how many files
```

```
    ' it plans on copying

    List1.AddItem "Files to Copy: " + Str(fileCount)
    List1.AddItem ""

End Sub


Private Sub SregoCETPx1_PreFileCopy(ByVal id As Long, _
                                    ByVal sourceFile As String, _
                                    ByVal destFile As String, _
                                    ByVal size As Double)

    ' Event fired before each individual file is copied

     List1.AddItem "File: " & Str(id)
     List1.AddItem "Source: " & sourceFile
     List1.AddItem "Destination: " & destFile
     List1.AddItem "Size: " & Str(size) & " bytes"

End Sub

Private Sub SregoCETPx1_PostFileCopy(ByVal id As Long, _
                                     ByVal sourceFile As String, _
                                     ByVal destFile As String, _
                                     ByVal size As Double, _
                                     ByVal status As Long)

    ' Event Fired after each individual file is copied.
    ' Also contains the status of file copy (0 - fail, 1 - success,
    ' 2 - not copied because overwrite protection)

    If status = 1 Then  ' file successfully copied

        List1.AddItem "File Successfully Copied."

    ElseIf status = 2 Then  ' file exists and was not overwritten

        List1.AddItem "File exists and was not overwritten."

    Else

        List1.AddItem "File Copy Failed."

    End If

    List1.AddItem ""

    OverallProgressBar.Value = id + 1

    DoEvents

End Sub
```

DeleteFile

The **DeleteFile** method will delete one or more files on the device. To delete multiple files, use wildcards in the filename (? and/or *).

Syntax:

```
boolean DeleteFile(BSTR filename)
```

- **filename –** the path on the device of the file(s) to delete. The path can contain wildcards (* and/or ?).

Return Value:

The method returns True for success and False otherwise.

Events:

The DeleteFile method fires several events while it is deleting files to help monitor its progress:

- DeleteInfo
- PreFileDelete
- PostFileDelete

Example:

```
SregoCETPx1.DeleteFile "\my documents\test1\test1.txt"

SregoCETPx1.DeleteFile "\my documents\test1\*.txt"
```

A more detailed example of deleting files with the **DeleteFile** method is shown below. This example utilizes the events fired by **DeleteFile** to display the progress with ListBox and ProgressBar controls:

```
Private Sub DeleteCommand_Click()

    OverallProgressBar.Visible = True
    List1.Clear

    If SregoCETPx1.DeleteFile("\my documents\test1\*.txt") Then

        List1.AddItem "File Deleted Successfully."

    Else

        List1.AddItem "Unable to Delete files."

    End If

    OverallProgressBar.Visible = False
```

```
End Sub

Private Sub SregoCETPx1_DeleteInfo(ByVal fileCount As Long)

    ' Event fired when DeleteFile starts so you will no how many files
    ' it plans on deleting.  This event is also used to initialize the
    ' progress indicator.


    List1.AddItem "Files to Delete: " + Str(fileCount)
    List1.AddItem ""

    OverallProgressBar.Value = 0
    OverallProgressBar.Min = 0
    If fileCount > 0 Then
        OverallProgressBar.Max = fileCount
    Else
        OverallProgressBar.Max = 1
    End If

    DoEvents

End Sub

Private Sub SregoCETPx1_PreFileDelete(ByVal id As Long, _
                                      ByVal filename As String, _
                                      ByVal size As Double)

   ' Event fired before each individual file is Deleted.

     List1.AddItem "File: " & Str(id)
     List1.AddItem "Filename: " & filename
     List1.AddItem "Size: " & Str(size) & " bytes"

End Sub

Private Sub SregoCETPx1_PostFileDelete(ByVal id As Long, _
                                       ByVal filename As String, _
                                       ByVal size As Double, _
                                       ByVal status As Long)

    ' Event Fired after each individual file is Deleted.
    ' Also contains the status of file copy (0 - fail, 1 - success)


    If status = 1 Then  ' file successfully pushed

        List1.AddItem "File Successfully Deleted."

    Else

        List1.AddItem "File Delete Failed. "

    End If

    List1.AddItem ""

    OverallProgressBar.Value = id + 1

    DoEvents

End Sub
```

DeleteFileEx

The **DeleteFileEx** method is an extended version of the DeleteFile method that will accept an option string as a parameter. The current option supported will recursively search subdirectories to delete the file specified by the search criteria.

Syntax:

```
boolean DeleteFile(BSTR filename, BSTR options)
```

- **filename –** the path on the device of the file(s) to delete. The path can contain wildcards (* and/or ?).
- **options –** the options string can either be empty ("") or one or more of the following flags (separated by a space if there is more than one option specified):

  /S – this flag will cause the method to recursively search all sub directories to find matching files to delete. This option could be dangerous if not used carefully.

  /AR – this flag will cause the method to delete Read Only files.

  /H – this flag will cause the method to delete Hidden files.

  If the option string is empty, the method will function the same as DeleteFile.

Return Value:

The method returns True for success and False otherwise.

Events:

The DeleteFileEx method fires the same events as the DeleteFile method, see the DeleteFile for detail examples of using the events:

- `DeleteInfo`
- `PreFileDelete`
- `PostFileDelete`

Example:

```
SregoCETPx1.DeleteFileEx "\my documents\*.txt", "/S"

SregoCETPx1.DeleteFileEx "\my documents\*.txt", "/S /AR"
```

MakeDirectory

The **MakeDirectory** method creates a new directory on the device.

Syntax:

```
boolean MakeDirectory(BSTR directoryName)
```

- **directoryName** – the name of the directory to create on the device.

Return Value:

The method returns True for success and False otherwise.

Example:

```
If SregoCETPx1.MakeDirectory("\my documents\test") Then
   ' Success
Else
   ' Failure
End If
```

RemoveDirectory

The **RemoveDirectory** method removes a directory on the device. The directory must be empty before it can be removed.

Syntax:

```
boolean RemoveDirectory(BSTR directoryName)
```

- **directoryName** – the name of the directory to remove on the device.

Return Value:

The method returns True for success and False otherwise.

Example:

```
If SregoCETPx1.RemoveDirectory("\my documents\test") Then
   ' Success
Else
   ' Failure
End If
```

PullFile

The **PullFile** method will copy one or more files from the device to desktop. To pull multiple files, the device path can contain wildcards (? and/or *). If pulling a single file, the destination path can specify a different filename to rename the copied file; otherwise, the destination path must be a directory. The **PullFileEx** method is similar to the **PullFile** method only it provides more options.

Syntax:

```
boolean PullFile(BSTR devicePath,
                 BSTR destinationPath,
                 boolean overwriteFlag)
```

- **devicePath –** the path on the device of the files to copy. The path can contain wildcards (* and/or ?).
- **destinationPath** – the destination path on the desktop.
- **overwriteFlag** – if set to FALSE, the files will not be overwritten in the destination path if they already exists. If set to TRUE, the files will be overwritten in the destination path if they already exists.

Return Value:

The method returns True for success and False otherwise.

Events:

The **PullFile** method fires several events while it is copying files to help monitor its progress:

- `PullInfo`
- `PreFilePull`
- `PostFilePull`
- `FilePullProgress`

Example:

```
SregoCETPx1.PullFile "\my documents\test1\sample.txt", _
                     "c:\temp", True


SregoCETPx1.PullFile "\my documents\test1\*.txt", _
                     "c:\temp", True
```

A more detailed example of pulling files with the **PullFile** method is shown below. This example utilizes the events fired by **PullFile** to display the progress with ListBox and ProgressBar controls:

```
Private Sub PullCommand_Click()

    OverallProgressBar.Visible = True
    FileProgressBar.Visible = True

    List1.Clear

    If SregoCETPx1.PullFile("\my documents\test1\*.txt", _
                            "c:\temp", True) Then

        List1.AddItem "File Pull Completed Successfully."

    Else

        List1.AddItem "Unable to Pull file."

    End If

    List1.AddItem ""

    OverallProgressBar.Visible = False
    FileProgressBar.Visible = False


End Sub


Private Sub SregoCETPx1_PullInfo(ByVal fileCount As Long)

    ' Event fired when PullFile starts so you will no how many files
    ' it plans on pulling from the device.  This event also initializes
    ' the progress indicators.

    List1.AddItem "Files to Pull: " + Str(fileCount)
    List1.AddItem ""

    OverallProgressBar.Value = 0
    OverallProgressBar.Min = 0
    If fileCount > 0 Then
        OverallProgressBar.Max = fileCount
    Else
        OverallProgressBar.Max = 1
    End If
```

```
    DoEvents

End Sub


Private Sub SregoCETPx1_PreFilePull(ByVal id As Long, _
                                    ByVal sourceFile As String, _
                                    ByVal destFile As String, _
                                    ByVal size As Double)

   ' Event fired before each individual file is Pulled.
   ' Event is also used to update the progress indicator.

     List1.AddItem "File: " & Str(id)
     List1.AddItem "Source: " & sourceFile
     List1.AddItem "Destination: " & destFile
     List1.AddItem "Size: " & Str(size) & " bytes"

     FileProgressBar.Value = 0
     FileProgressBar.Min = 0
     FileProgressBar.Max = 1000

     DoEvents

End Sub

Private Sub SregoCETPx1_PostFilePull(ByVal id As Long, _
                                     ByVal sourceFile As String, _
                                     ByVal destFile As String, _
                                     ByVal size As Double, _
                                     ByVal status As Long)

    ' Event Fired after each individual file is Pulled.
    ' Also contains the status of file copy (0 - fail, 1 - success,
    ' 2 - not Pulled because overwrite protection)


     If status = 1 Then  ' file successfully pulled

        List1.AddItem "File Successfully Pulled."

     ElseIf status = 2 Then  ' file exists and was not overwritten

        List1.AddItem "File exists and was not overwritten."

     Else

        List1.AddItem "File Pull Failed. "

     End If

    List1.AddItem ""

    OverallProgressBar.Value = id + 1

    DoEvents

End Sub

Private Sub SregoCETPx1_FilePullProgress(ByVal id As Long, _
                                         ByVal progress As Long)

  ' This event is fired as blocks of the file is pushed to the device.
```

```
        FileProgressBar.Value = progress

        DoEvents

    End Sub
```

PullFileEx

The **PullFileEx** method will copy one or more files from the device to the desktop. The **PullFileEx** method is similar to the **PullFile** method except it provides more options.

To pull multiple files, the device path can contain wildcards (? and/or *). If pulling a single file, the destination path can specify a different filename to rename the copied file; otherwise, the destination path must be a directory.

The **PullFileEx** method provides an option for pulling only files on the device that have a newer last write date than the files of the same name on the desktop. This method can be used for synchronizing files between the desktop and the device.

Syntax:

```
boolean PullFileEx(BSTR devicePath,
                   BSTR destinationPath,
                   BSTR options)
```

- **devicePath –** the path on the device of the file(s) to copy. The path can contain wildcards (* and/or ?).
- **destinationPath** – the destination path on the desktop.
- **options** – the option string can contain one following flags:
    - /Y - Overwrite a file if it already exists. By default, existing files will not be overwritten. Using this option is equivalent to using the **PullFile** method with the overwriteFlag set to 1.
    - /D - Copies only files whose last write time on the device is newer than the destination file's last write time on the desktop. If the source file does not already exist on the destination path, it will be copied.
    - /A – Copies the standard File Properties from the source file to the destination file. These properties include Read Only, Hidden, System, Archive, Normal, Temporary.

Return Value:

The method returns True for success and False otherwise.

Events:

The **PullFileEx** method fires several events while it is copying files to help monitor its progress (these events are the same as those fired by the **PullFile** method):

- `PullInfo`
- `PreFilePull`
- `PostFilePull`
- `FilePullProgress`

Example:

Copy file from device to desktop's temp directory, no options:

```
SregoCETPx1.PullFileEx "\my documents\test\sample.txt", "c:\temp", ""
```

Copy file from device to desktop's temp directory and rename file:

```
SregoCETPx1.PullFileEx "\my documents\test\sample.txt", "c:\temp\tmp.txt", ""
```

Copy all files ending with .txt from device to desktop's temp directory:

```
SregoCETPx1.PullFileEx "\my documents\test\*.txt", "c:\temp", ""
```

Copy all files ending with .txt from device to desktop's temp directory only if the file does not exists on the desktop:

```
SregoCETPx1.PullFileEx "\my documents\test\*.txt", "c:\temp", "/Y"
```

Copy all files ending with .txt from device to desktop's temp directory only if the source file has a new time than a file of the same name on the desktop::

```
SregoCETPx1.PullFileEx "\my documents\test\*.txt", "c:\temp", "/D"
```

The **PullFile** method example illustrating the use of events to track progress is the same for **PullFileEx**. **PullFileEx** will fire the same methods in the same way as PullFile.

PushFile

The **PushFile** method will copy one or more files from the desktop to the device. To push multiple files, the source path can contain wildcards (? and/or *). If pulling a single file, the destination path can specify a different filename to rename the copied file; otherwise, the destination path must be a directory. The **PushFileEx** method is similar to the **PushFile** method only it provides more options.

Syntax:

```
boolean PushFile(BSTR sourcePath, BSTR devicePath,
                 short overWriteFlag)
```

- **sourcePath –** the path on the desktop of the file(s) to copy. The path can contain wildcards (* and/or ?).
- **destinationPath** – the destination path on the device.
- **overwriteFlag** – if set to FALSE, the files will not be overwritten in the destination path if they already exists. If set to TRUE, the files will be overwritten in the destination path if they already exists.

Return Value:

The method returns True for success and False otherwise.

Events:

The **PushFile** method fires several events while it is copying files to help monitor its progress:

- `PushInfo`
- `PreFilePush`
- `PostFilePush`
- `FilePushProgress`

Example:

```
SregoCETPx1.PushFile "c:\temp\sample.txt", "\my documents", True

SregoCETPx1.PushFile "c:\temp\*.txt", "\my documents", True
```

A more detailed example of pushing files with the **PushFile** method is shown below. This example utilizes the events fired by **PushFile** to display the progress with ListBox and ProgressBar controls:

```
Private Sub PushFileCommand_Click()

    OverallProgressBar.Visible = True
    FileProgressBar.Visible = True

    List1.Clear

    If SregoCETPx1.PushFile("c:\temp\*.txt", _
                            "\my documents\test1", True) Then

        List1.AddItem "File Push Completed Successfully."

    Else

        List1.AddItem "Unable to Push file."

    End If

    List1.AddItem ""

    OverallProgressBar.Visible = False
    FileProgressBar.Visible = False

End Sub

Private Sub SregoCETPx1_PushInfo(ByVal fileCount As Long)

    ' Event fired when PushFile starts so you will no how many files
    ' it plans on pushing to the device, also used to initialize the
    ' progress indicators

    List1.AddItem "Files to Push: " + Str(fileCount)
    List1.AddItem ""

    OverallProgressBar.Value = 0
    OverallProgressBar.Min = 0
    If fileCount > 0 Then
        OverallProgressBar.Max = fileCount
    Else
        OverallProgressBar.Max = 1
    End If


    DoEvents

End Sub


Private Sub SregoCETPx1_PreFilePush(ByVal id As Long, _
                                    ByVal sourceFile As String, _
                                    ByVal destFile As String, _
                                    ByVal size As Double)

   ' Event fired before each individual file is pushed.
   ' Event is also used to update the progress indicator.

     List1.AddItem "File: " & Str(id)
     List1.AddItem "Source: " & sourceFile
     List1.AddItem "Destination: " & destFile
     List1.AddItem "Size: " & Str(size) & " bytes"

     FileProgressBar.Value = 0
     FileProgressBar.Min = 0
```

```
     FileProgressBar.Max = 1000

     DoEvents

End Sub

Private Sub SregoCETPx1_PostFilePush(ByVal id As Long, _
                                     ByVal sourceFile As String, _
                                     ByVal destFile As String, _
                                     ByVal size As Double, _
                                     ByVal status As Long)

    ' Event Fired after each individual file is Pushed.
    ' Also contains the status of file copy (0 - fail, 1 - success,
    ' 2 - not pushed because overwrite protection)

    If status = 1 Then  ' file successfully pushed

        List1.AddItem "File Successfully Pushed."

    ElseIf status = 2 Then  ' file exists and was not overwritten

        List1.AddItem "File exists and was not overwritten."

    ElseIf status = 3 Then ' source file is not newer than dest file."

        List1.AddItem "Source File is not newer than destination file."

    Else
        List1.AddItem "File Push Failed."

    End If

    List1.AddItem ""

    OverallProgressBar.Value = id + 1

    DoEvents

End Sub


Private Sub SregoCETPx1_FilePushProgress(ByVal id As Long, _
                                         ByVal progress As Long)

    ' This event is fired as blocks of the file is pushed to the device.

    FileProgressBar.Value = progress

    DoEvents

End Sub
```

PushFileEx

The **PushFileEx** method will copy one or more files from the desktop to the device. The **PushFileEx** method is similar to the **PushFile** method except it provides more options.

To push multiple files, the source path can contain wildcards (? and/or *). If pushing a single file, the destination path can specify a different filename to rename the copied file; otherwise, the destination path must be a directory.

The **PushFileEx** method provides an option for pushing only files on the desktop that have a newer modified date than the file on the device. This method can be used for synchronizing files between the desktop and the device.

Syntax:

```
boolean PushFileEx(BSTR sourcePath, BSTR devicePath,
                   BSTR options)
```

- **sourcePath –** the path on the desktop of the file(s) to copy. The path can contain wildcards (* and/or ?).
- **destinationPath** – the destination path on the device.
- **options** – the option string can contain one following flags (separated by a space):

    - /Y - Overwrite a file if it already exists. By default, existing files will not be overwritten. Using this option is equivalent to using the **PushFile** method with the overwriteFlag set to 1.

    - /D - Copies only files whose source time on the desktop is newer than the destination file time on the device.

    - /A – Copies the standard File Properties from the source file to the destination file. These properties include Read Only, Hidden, System, Archive, Normal, Temporary.

Return Value:

The method returns True for success and False otherwise.

Events:

The **PushFileEx** method fires several events while it is copying files to help monitor its progress:

- `PushInfo`
- `PreFilePush`

- `PostFilePush`
- `FilePushProgress`

Example:

Copy file from desktop to the device, no options:

```
SregoCETPx1.PushFileEx "c:\temp\sample.txt", "\my documents", ""
```

Copy file from desktop to the device, renaming file on device, no options:

```
SregoCETPx1.PushFileEx "c:\temp\sample.txt", "\my documents\ppc.txt", ""
```

Copy files ending with .txt from desktop to the device, no options:

```
SregoCETPx1.PushFileEx "c:\temp\*.txt", "\my documents\test", ""
```

Copy files ending with .txt from desktop to the device only if the file does not exists on the device:

```
SregoCETPx1.PushFileEx "c:\temp\*.txt", "\my documents\test", "/Y"
```

Copy files ending with .txt from the desktop to the device only if the source file's last write time is newer than the same file's last write time on the device.

```
SregoCETPx1.PushFileEx "c:\temp\*.txt", "\my documents\test", "/D"
```

The **PushFile** method example illustrating the use of events to track progress is the same for **PushFileEx**. **PushFileEx** will fire the same methods in the same way as **PushFile**.

TouchFile

The **TouchFile** method is used to update the last write time on the specified file(s) with the current time. This method may have limited uses, but it can be helpful if you want to ensure certain files will be synchronized with **PullFileEx** without actually modifying the files.

This method can touch multiple files by specifying wildcards (? and/or *) in the device path.

Syntax:

```
boolean TouchFile( BSTR devicePath )
```

- **devicePath –** the path on the device of the file(s) to touch. The path can contain wildcards (? and/or *).

Return Value:

The method returns True for success and False otherwise.

Events:

The Touch method fires several events while it is touches files to help monitor its progress:

- TouchInfo
- PreFileTouch
- PostFileTouch

Example:

```
SregoCETPx1.TouchFile "\my documents\sample.txt"

SregoCETPx1.TouchFile "\my documents\*.txt"
```

A more detailed example of touching files with the **TouchFile** method is shown below. This example utilizes the events fired by **TouchFile** to display the progress with ListBox and ProgressBar controls:

```
Private Sub Command9_Click()

    List1.Clear
    OverallProgressBar.Visible = True

    If SregoCETPx1.TouchFile("\my documents\test\test?.*") Then

         List1.AddItem "File Touch Completed Successfully."

    Else

        List1.AddItem "Unable to Touch files."

    End If

    OverallProgressBar.Visible = False

End Sub

Private Sub SregoCETPx1_TouchInfo(ByVal count As Long)

    List1.AddItem "Files to Touch: " + Str(count)
    List1.AddItem ""


End Sub


Private Sub SregoCETPx1_PreFileTouch(ByVal id As Long, _
                                     ByVal file As String)
```

```
    ' Event fired before each individual file is touched.
    ' Event is also used to update the progress indicator.

      List1.AddItem "File: " & Str(id)
      List1.AddItem "Filename: " & file

      DoEvents

End Sub


Private Sub SregoCETPx1_PostFileTouch(ByVal id As Long, _
                                      ByVal file As String, _
                                      ByVal status As Long)

    ' Event Fired after each individual file is touched.
    ' Also contains the status of file touch (0 - fail, 1 - success)

    If status = 1 Then  ' file successfully copied

        List1.AddItem "File Successfully Touched."

     Else

        List1.AddItem "File Touch Failed. "

    End If

    List1.AddItem ""

    OverallProgressBar.Value = id + 1

    DoEvents

End Sub
```

GetFileLastWriteTime

The **GetFileLastWriteTime** method retrieve the last write time for a specified file on the device.

Syntax:

```
boolean GetFileLastWriteTime(BSTR devicePath,
                             VARIANT* lastWriteTime)
```

- **devicePath –** the path on the device to retrieve the last write time for..
- **lastWriteTime** – returns a variant containing the date and time for the specified file on the device.

Return Value:

The method returns True for success and False otherwise.

Example:

```
Dim createTime as Variant

If SregoCETPx1.GetFileLastWriteTime("\my documents\test.txt", createTime) Then

    Debug.Print createTime

End If
```

StartApplication

The **StartApplication** method will invoke an application on the device.

Syntax:

```
boolean StartApplication(BSTR commandLine)
```

- **commandLine** – the command-line to execute on the device. The command-line can contain an executable name as well as options.

Return Value:

The method returns True for success and False otherwise.

StartApplicationEx

The **StartApplicationEx** method will invoke an application on the device.

Syntax:

```
boolean StartApplicationEx(BSTR appName,
                           BSTR parameters)
```

- **appName** – the application name to execute on the device.
- **parameters** – the parameters to pass to the application.

Return Value:

The method returns True for success and False otherwise.

SetReadOnlyAttr

The **SetReadOnlyAttr** method will change the read only for a file on the device.

Syntax:

```
boolean SetReadOnlyAttr(BSTR filename,
                        boolean readOnlyFlag)
```

- **filename** – The complete path to the file on the device to modify.
- **readOnlyFlag** – if set to TRUE, the file will be set to read only; if set to FALSE, the file will not be read only.

Return Value:

The method returns True for success and False otherwise.

Example:

```
SregoCeTPx1.SetReadOnlyAttr "\my documents\test.txt", True
```

VerifyFile

The **VerifyFile** method will verify that a file exists on the device.

Syntax:

```
boolean VerifyFile(BSTR filename)
```

- **filename** – The complete path to the file on the device to check.

Return Value:

The method returns True if the file exists and False otherwise.

Example:

```
If VerifyFile( "\my documents\test.txt") then

    .
    .
    .
```

```
End If
```

VerifyDirectory

The **VerifyDirectory** method will verify that a directory exists on the device.

Syntax:

```
boolean VerifyDirectory(BSTR directory)
```

- **directory** – The directory on the device to check.

Return Value:

The method returns True if the directory exists and False otherwise.

Example:

```
If VerifyDirectory( "\my documents\testDir") then

      .
      .
      .

End If
```

SetFileAttributes

The **SetFileAttributes** method will set the specified file's attributes to the specified properties.

Syntax:

```
boolean SetFileAttributes(BSTR filename,
                          long attributes)
```

- **filename** – The file on the device to modify.
- **attributes** – the file property can be a combination of the following values ORed together:

```
FILE_ATTRIBUTE_READONLY = 1
FILE_ATTRIBUTE_HIDDEN = 2
FILE_ATTRIBUTE_SYSTEM  = 4
```

```
FILE_ATTRIBUTE_ARCHIVE = 32
FILE_ATTRIBUTE_NORMAL = 128
FILE_ATTRIBUTE_TEMPORARY = 256
```

Return Value:

The method returns True if the directory exists and False otherwise.

Example:

```
Const FILE_ATTRIBUTE_READONLY = 1

Private Sub Command1_Click()

    Dim stat As Boolean
    Dim attr As Long

    attr = SregoCETPx1.GetFileAttributes("\my documents\test.txt")

    Debug.Print "Properties: " + Str(attr)

    attr = attr Or FILE_ATTRIBUTE_READONLY

    stat = SregoCETPx1.SetFileAttributes("\my documents\test.txt", attr)

    Debug.Print "Status: " + Str(stat)

    attr = SregoCETPx1.GetFileAttributes("\my documents\test.txt")

    Debug.Print "Properties: " + Str(attr)

End Sub
```

GetFileAttributes

The **GetFileAttributes** method will retrieve the current file properties from a file on the connected device.

Syntax:

```
long GetFileAttributes(BSTR filename)
```

- **filename** – The file on the device to get properties from.

Return Value:

The method returns -1 if there was a problem or the properties which can be the following values ORed together:

```
FILE_ATTRIBUTE_ARCHIVE = 32
FILE_ATTRIBUTE_COMPRESSED  = 2048
FILE_ATTRIBUTE_DIRECTORY = 16
FILE_ATTRIBUTE_ENCRYPTED  = 64
FILE_ATTRIBUTE_HIDDEN = 2
FILE_ATTRIBUTE_INROM    = 64
FILE_ATTRIBUTE_NORMAL = 128
FILE_ATTRIBUTE_READONLY = 1
FILE_ATTRIBUTE_REPARSE_POINT = 1024
FILE_ATTRIBUTE_ROMMODULE = 8192
FILE_ATTRIBUTE_TEMPORARY  = 256
FILE_ATTRIBUTE_SPARSE_FILE = 512
```

Example:

See SetFileProperties.

GetFileSize

The G**etFileSize** method will retrieve the size in bytes of the specified file on the device. The method will either return the high or low part of the file size.

Syntax:

```
long GetFileSize(BSTR filename, long mode)
```

- **filename** – The file on the device to get the size of.
- **Mode** – 0 for the low part and 1 for the high part of the 64bit size value.

Return Value:

0 if unsuccessful. Use VerifyFile or GetErrorMessage to see if a file is 0 or and error does exists.

Example:

```
Dim s as Long
s = SregoCETPx1.GetFileSize("\my documents\test.txt", 0)
```

GetFileSizePC

The G**etFileSizePC** method will retrieve the size in bytes of the specified file on the desktop. The method will either return the high or low part of the file size.

Syntax:

```
long GetFileSizePC(BSTR filename, long mode)
```

- **filename** – The file on the device to get the size of.
- **Mode** – 0 for the low part and 1 for the high part of the 64bit size value.

Return Value:

0 if unsuccessful. Use VerifyFile or GetErrorMessage to see if a file is 0 or and error does exists.

File Events

This section describes all of the events fired by File Manipulation methods. These events can be used to monitor progress as in updating a progress bar, percentage complete display, etc. It is not required to implement these methods if you do not need to monitor the progress.

MoveInfo

The **MoveInfo** event is fired once after the **MoveFile** method is called. This event tells how many files the method will attempt to move. This event can be used to monitor the progress by knowing how many files will be moved; the **PreFileMove** and **PostFileMove** methods will be fired for each file moved.

Syntax:

```
void MoveInfo(long fileCount)
```

- **fileCount** – the number of files that will be moved.

Example:

See the **MoveFile** method example.

CopyInfo

The **CopyInfo** event is fired once after the **CopyFile** method is called. This event tells how many files the method will attempt to copy. This event can be used to monitor the progress by knowing how many files will be copied; the **PreFileCopy** and **PostFileCopy** methods will be fired for each file copied.

Syntax:

```
void CopyInfo(long fileCount)
```

- **fileCount** – the number of files that will be copied.

Example:

See the **CopyFile** method example.

DeleteInfo

The **DeleteInfo** event is fired once after the **DeleteFile** method is called. This event tells how many files the method will attempt to delete. This event can be used to monitor the progress by knowing how many files will be deleted; the **PreFileDelete** and **PostFileDelete** methods will be fired for each file deleted.

Syntax:

```
void DeleteInfo(long fileCount)
```

- **fileCount** – the number of files that will be deleted.

Example:

See the **DeleteFile** method example.

PullInfo

The **PullInfo** event is fired once after the **PullFile** or the **PullFileEx** method is called. This event tells how many files the method will attempt to pull. This event can be used to monitor the progress by knowing how many files will be pulled; the **PreFilePull** and **PostFilePull** methods will be fired for each file pulled. The **FilePullProgress** event is used to monitor the progress of an individual file transfer.

Syntax:

```
void PullInfo(long count)
```

- **fileCount** – the number of files that will be pulled.

Example:

See the **PullFile** method example.

PushInfo

The **PushInfo** event is fired once after the **PushFile** or the **PushFileEx** method is called. This event tells how many files the method will attempt to push. This event can be used to monitor the progress by knowing how many files will be pushed; the **PreFilePush** and **PostFilePush** methods will be fired for each file pushed. The **FilePushProgress** event is used to monitor the progress of an individual file transfer.

Syntax:

```
void PushInfo(long count)
```

- **fileCount** – the number of files that will be pushed..

Example:

See the **PushFile** method example.

TouchInfo

The **TouchInfo** event is fired once after the **TouchFile** method is called. This event tells how many files the method will attempt to touch. This event can be used to monitor the progress by knowing how many files will be touched; the **PreFileTouch** and **PostFileTouch** methods will be fired for each file touched.

Syntax:

```
void TouchInfo(long count)
```

- **fileCount** – the number of files that will be touched.

Example:

See the **TouchFile** method example.

PreFileMove

This **PreFileMove** event is fired before each file is moved by the **MoveFile** method. Since the **MoveFile** method can be used to move multiple files at once with wildcards, this method can be used to update a progress indicator. The **PostFileMove** event is fired after each individual file is moved along with the status.

Syntax:

```
void PreFileMove(long id, BSTR sourceFile,
                 BSTR destFile, double size)
```

- **id** – the file id of the file about to be moved. The id value will be 0 to **FileCount** (sent in the **MoveInfo** event) minus 1.
- **sourceFile** – the name of the source file about to be moved.
- **destFile** – the destination of the file being moved.
- **Size** – the size in bytes of the file to be moved.

Example:

See the **MoveFile** method example.

PreFileCopy

This event is fired after each file is copied with the **CopyFile** method. Since the **CopyFile** method can be used to copy multiple files at once with wildcards, this event can be used to update a progress indicator. The **PostFileCopy** event is fired after each individual file is copied along with the status.

Syntax:

```
void PreFileCopy(long id, BSTR sourceFile,
                 BSTR destFile, double size)
```

- **id** – the file id of the file about to be copied. The id value will be 0 to **FileCount** (sent in the **CopyInfo** event) minus 1.
- **sourceFile** – the name of the source file about to be copied.
- **destFile** – the destination of the file being copied.
- **Size** – the size in bytes of the file to be copied.

Example:

See the **CopyFile** method example.

PreFileDelete

This event is fired after each file is deleted with the **DeleteFile** method. Since the **DeleteFile** method can be used to delete multiple files at once with wildcards, this event can be used to update a progress indicator. The **PostFileDelete** event is fired after each individual file is deleted along with the status.

Syntax:

```
void PreFileDelete(long id, BSTR filename,
                   double size)
```

- **id** – the file id of the file about to be deleted. The id value will be 0 to **FileCount** (sent in the **DeleteInfo** event) minus 1.
- **sourceFile** – the name of the source file about to be deleted.
- **destFile** – the destination of the file being deleted.
- **Size** – the size in bytes of the file to be deleted.

Example:

See the **DeleteFile** method example.

PreFilePull

This event is fired after each file is pulled with the **PullFile** and **PullFileEx** methods. Since the **PullFile** and **PullFileEx** methods can be used to pull multiple files at once with wildcards, this event can be used to update a progress indicator. The **PostFilePull** event is fired after each individual file is pulled along with the status.

Syntax:

```
void PreFilePull(long id, BSTR sourceFile,
                 BSTR destFile, double size)
```

- **id** – the file id of the file about to be pulled. The id value will be 0 to **FileCount** (sent in the **PullInfo** event) minus 1.
- **sourceFile** – the name of the source file about to be pulled.
- **destFile** – the destination of the file being pulled.
- **Size** – the size in bytes of the file to be pulled.

Example:

See the **PullFile** method example.

PreFilePush

This event is fired after each file is pushed with the **PushFile** and **PushFileEx** methods. Since the **PushFile** and **PushFileEx** methods can be used to push multiple files at once with wildcards, this event can be used to update a progress indicator. The **PostFilePush** event is fired after each individual file is pushed along with the status.

Syntax:

```
void PreFilePush(long id, BSTR sourceFile,
                 BSTR destFile, double size)
```

- **id** – the file id of the file about to be pushed. The id value will be 0 to **FileCount** (sent in the **PushInfo** event) minus 1.
- **sourceFile** – the name of the source file about to be pushed.
- **destFile** – the destination of the file being pushed.
- **Size** – the size in bytes of the file to be pushed.

Example:

See the **PushFile** method example.

PreFileTouch

This event is fired after each file is touched with the **TouchFile** method. Since the **TouchFile** method can be used to touch multiple files at once with wildcards, this event can be used to update a progress indicator. The **PostFileTouch** event is fired after each individual file is touched along with the status.

Syntax:

```
void PreFileTouch(long id, BSTR file)
```

- **id** – the file id of the file about to be touched. The id value will be 0 to **FileCount** (sent in the **TouchInfo** event) minus 1.
- **sourceFile** – the name of the source file about to be touched.
- **destFile** – the destination of the file being touched.
- **Size** – the size in bytes of the file to be touched.

Example:

See the **TouchFile** method example.

PostFileMove

This event is fired after each file is moved with the **MoveFile** method. Since the **MoveFile** method can be used to move multiple files at once with wildcards, this event can be used to update a progress indicator.

Syntax:

```
void PostFileMove(long id, BSTR sourceFile,
                  BSTR destFile,
                  double size, long status)
```

- **id** – the file id of the file that was just moved. The id value will be 0 to **FileCount** (sent in the **MoveInfo** event) minus 1.
- **sourceFile** – the name of the source file just moved.
- **destFile** – the destination of the file just moved.
- **Size** – the size in bytes of the file just moved.

Example:

See the **MoveFile** method example.

PostFileCopy

This event is fired after each file is copied with the **CopyFile** method. Since the **CopyFile** method can be used to copy multiple files at once with wildcards, this event can be used to update a progress indicator.

Syntax:

```
void PostFileCopy(long id, BSTR sourceFile,
                  BSTR destFile,
                  double size, long status)
```

- **id** – the file id of the file that was just copied. The id value will be 0 to **FileCount** (sent in the **CopyInfo** event) minus 1.
- **sourceFile** – the name of the source file just copied.
- **destFile** – the destination of the file just copied.
- **Size** – the size in bytes of the file just copied.

Example:

See the **CopyFile** method example.

PostFileDelete

This event is fired after each file is copied with the **DeleteFile** method. Since the **DeleteFile** method can be used to delete multiple files at once with wildcards, this event can be used to update a progress indicator.

Syntax:

```
void PostFileDelete(long id, BSTR filename,
                    double size, long status)
```

- **id** – the file id of the file that was just deleted. The id value will be 0 to **FileCount** (sent in the CopyInfo event) minus 1.
- **sourceFile** – the name of the source file just deleted.
- **destFile** – the destination of the file just deleted.
- **Size** – the size in bytes of the file just deleted.

Example:

See the **DeleteFile** method example.

PostFilePull

This event is fired after each file is pulled with the **PullFile** method. Since the **PullFile** method can be used to pull multiple files at once with wildcards, this event can be used to update a progress indicator.

Syntax:

```
void PostFilePull(long id, BSTR sourceFile,
                  BSTR destFile,
                  double size, long status)
```

- **id** – the file id of the file that was just pulled. The id value will be 0 to **FileCount** (sent in the **CopyInfo** event) minus 1.
- **sourceFile** – the name of the source file just pulled.
- **destFile** – the destination of the file just pulled.
- **Size** – the size in bytes of the file just pulled.

Example:

See the **PullFile** method example.

PostFilePush

This event is fired after each file is pushed with the **PushFile** method. Since the **PushFile** method can be used to push multiple files at once with wildcards, this event can be used to update a progress indicator.

Syntax:

```
void PostFilePush(long id, BSTR sourceFile,
                  BSTR destFile,
                  double size, long status)
```

- **id** – the file id of the file that was just pushed. The id value will be 0 to **FileCount** (sent in the **PushInfo** event) minus 1.
- **sourceFile** – the name of the source file just pushed.
- **destFile** – the destination of the file just pushed.
- **Size** – the size in bytes of the file just pushed.

Example:

See the **PushFile** method example.

PostFileTouch

This event is fired after each file is touched with the **TouchFile** method. Since the **TouchFile** method can be used to touch multiple files at once with wildcards, this event can be used to update a progress indicator.

Syntax:

```
void PostFileTouch(long id, BSTR file, long status)
```

- **id** – the file id of the file that was just touched. The id value will be 0 to **FileCount** (sent in the **TouchInfo** event) minus 1.
- **sourceFile** – the name of the source file just touched.
- **destFile** – the destination of the file just touched.
- **Size** – the size in bytes of the file just touched.

Example:

See the **TouchFile** method example.

FilePullProgress

The **FilePullProgress** event is fired during the transfer process of each individual file pulled with the **PullFile** method. This event can be used to monitor the progress of an individual file transfer.

Syntax:

```
void FilePullProgress(long id, long progress)
```

- **id** – the file id of the file being pulled. The id value will be 0 to **FileCount** (sent in the **PullInfo** event) minus 1.
- **progress** – a value from 0 to 1000. The value will reach 1000 when the transfer is completed.

Example:

See the **PullFile** method example.

FilePushProgress

The **FilePushProgress** event is fired during the transfer process of each individual file pushed with the **PushFile** method. This event can be used to monitor the progress of an individual file transfer.

Syntax:

```
void FilePushProgress(long id, long progress)
```

- **id** – the file id of the file being pushed. The id value will be 0 to **FileCount** (sent in the **PushInfo** event) minus 1.
- **progress** – a value from 0 to 1000. The value will reach 1000 when the transfer is completed.

Example:

See the **PushFile** method example.

Database Methods

The methods in this section work with a CE database. The database can be the default volume on the CE Device or an external volume (.cdb file).

MountDatabaseVolume

This method mounts the database contained in the specified .CDB file. All CE databases other than the default database must be mounted before they can be accessed. The external databases files will have a .CDB extension.

Syntax:

```
string MountDatabaseVolume(string filename, long flags)
```

- **Filename** – name of .cdb file to mount
- **Flags** – OPEN_EXISTING (3) is currently the only supported flag

Return Value;

Returns the volumeId (CEGUID) of the mounted volume as a string. Returns an empty string ("") if the volume could not be mounted.

Example:

```
Private Sub MountDBCommand_Click()

    List1.Clear

    Dim filename As String
    Dim id As String

    filename = "\my documents\Nwind.cdb"

    id = SregoCETPx1.MountDatabaseVolume(filename, OPEN_EXISTING)

    If id <> "" Then

        List1.AddItem "Volume Mounted"
        List1.AddItem "Volume Filename: " + filename
        List1.AddItem "Volume Id: " + id

    Else

        List1.AddItem "Unable to mount Volume: " + filename

    End If

End Sub
```

UnmountDatabaseVolume

This method unmounts a database. The database is specified with its volume id (string returned by the **MountDatabaseVolume** method).

Syntax:

```
Boolean UnmountDatabaseVolume( string volumeId )
```

- **volumeId** – volumeId of the database to close specified as a string

Return Value:

Returns True if successful and False otherwise.

Example:

```
SregoCETPx1.UnmountDatabaseVolume volumeId
```

GetDatabaseVolumeList

This method retrieves a list of mounted database volumes. The default CE database should always be in the list ("SystemHeap").

Syntax:

```
Boolean GetDatabaseVolumeList( variant idList,
                               variant nameList,
                               variant count)
```

- **idList** – variant that will return an array of volume ids (CEGUID values).
- **nameList** – variant that will return an array of volume names.
- **Count** – variant that will return the number of items in the arrays.

Return Value:

Returns True if successful and False otherwise.

Example:

```
Private Sub GetVolumeListCommand_Click()

    Dim i As Integer
```

```
    List1.Clear

    List1.AddItem "Volume List"
    List1.AddItem ""

    Dim idList As Variant
    Dim nameList As Variant
    Dim count As Variant

    If SregoCETPx1.GetDatabaseVolumeList(idList, nameList, count) Then

        For i = 0 To count - 1

            List1.AddItem Str(i) + " -  Volume Name: " + nameList(i)
            List1.AddItem "       Volume Id: " + idList(i)
            List1.AddItem ""
        Next

    End If

End Sub
```

GetDatabaseList

This method retrieves a list of available database in the specified volume. To get a list of databases in the default CE database, specify the volumeId as an empty string (“”). The list can also be filtered by dbType. The dbType is a user defined number assigned to the database when they are created.

Syntax:

```
Boolean GetDatabaseList( string volumeId,
                         Long dbType,
                         Variant idList,
                         Variant count )
```

- **volumeId** – volume id to search for a list of databases.
- **dbType** – user defined id that can be assigned to databases. If a value is specified here, then only database’s with this type value will be returned. If 0 is used, all databases in the volume will be returned.
- **idList** – variant used to return the array of database ids.
- **Count** – variant used to return the count of items in array.

Return Value:

Returns True if successful and False otherwise.

Example:

```
Private Sub GetDBListCommand_Click()

    Dim i As Integer
    Dim j As Integer

    List1.Clear

    List1.AddItem "Database List"
    List1.AddItem ""

    Dim volIdList As Variant
    Dim volNameList As Variant
    Dim volCount As Variant

    If SregoCETPx1.GetDatabaseVolumeList(volIdList, _
                      volNameList, volCount) Then

        For i = 0 To volCount - 1

            List1.AddItem Str(i) + " - Volume: " + volNameList(i)
            List1.AddItem ""

            Dim idList As Variant
            Dim nameList As Variant
            Dim count As Variant

            If SregoCETPx1.GetDatabaseList(volIdList(i), 0, idList, _
                                nameList, count) Then

                For j = 0 To count - 1

                    List1.AddItem Str(j) + " - " + _
                       CStr(idList(j)) + " - " + nameList(j)

                Next

            End If
        Next
        List1.AddItem ""
    End If

End Sub
```

OpenDatabaseByName

This method opens the specified database in the specified volume. This method has been replaced by the **OpenDatabaseByNameEx** method to simplify the SortPropId specification. In this method, the SortPropId can be composed with the sample MakeSortId function in the example; however, the **OpenDatabaseByNameEx** method allows the sortPropId and sortPropType to be specified independently.

Syntax:

```
Long OpenDatabaseByName( string volumeId,
                         string dbName,
                         long sortPropId,
                         long flags)
```

- **volumeId** – the volume id of the volume containing the database to open. If you specify an empty string "" for the volume id, all volumes will be searched to find the database to open.
- **dbName** – the name of the database to open.
- **sortPropId** – the sort property to use for the database. Use 0 to use the default sort property.
- **Flags** – the action flag. CEDB_AUTOINCREMENT (1) is used to automatically go to the next record after each read. If 0 is used, the current position is not incremented after each read.

Return Value:

Returns the database handle of the open database if successful and 0 otherwise.

Example:

```
Private Sub OpenTableCommand_Click()

   List1.Clear

  ' Mount a Database Volume
    Dim filename As String
    Dim volumeId As String

    filename = "\my documents\Nwind.cdb"
    volumeId = SregoCETPx1.MountDatabaseVolume(filename, OPEN_EXISTING)

    If volumeId <> "" Then

        List1.AddItem "Volume Mounted: " + filename
        List1.AddItem "Volume Id: " + volumeId
        List1.AddItem ""
```

```
    Else

        List1.AddItem "Unable to mount Volume: " + filename
        Exit Sub

    End If


 ' Open Database in mounted Volume
    Dim dbHandle As Long
    dbHandle = SregoCETPx1.OpenDatabaseByName(volumeId, _
                   "Employees", 0, CEDB_AUTOINCREMENT)

    If dbHandle <> 0 Then  ' make sure database was opened successfully

        List1.AddItem "Database Opened"
        List1.AddItem "     Handle: " + Str(dbHandle)

        ' do something here with the open database


        SregoCETPx1.CloseDatabase dbHandle
        List1.AddItem "Database Closed"


    End If

    SregoCETPx1.UnmountDatabaseVolume volumeId

End Sub
```

Function to create sort id from a Property Type and Property Id:

```
      Function MakeSortId(propType As Long, propId As Long) As Long

            MakeSortId = propId * 2 ^ 16 Or propType

      End Function
```

OpenDatabaseByNameEx

This method opens the specified database in the specified volume. This method replaces the **OpenDatabaseByName** method as it allows the sortPropId and sortPropType to be specified independently.

Syntax:

```
      Long OpenDatabaseByNameEx( string volumeId,
                                 string dbName,
                                 long sortPropId,
```

```
                                    long sortPropType,
                                    long flags)
```

- **volumeId** – the volume id of the volume containing the database to open. If you specify an empty string "" for the volume id, all volumes will be searched to find the database to open.
- **dbName** – the name of the database to open.
- **sortPropId** – the sort property to use for the database. Use 0 to use the default sort property.
- **sortPropType** – the sort property type. The value must be one of the following:

  CEVT_I2 (2) - short
  CEVT_UI2 (18) - unsigned short
  CEVT_I4 (3) - long
  CEVT_UI4 (19) - unsigned long
  CEVT_FILETIME (64) - DateTime
  CEVT_LPWSTR (31) - String
  CEVT_BLOB (65) - BLOB
  CEVT_BOOL (11) - Boolean
  CEVT_R8 (5) – Double

- **Flags** – the action flag. CEDB_AUTOINCREMENT (1) is used to automatically go to the next record after each read. If 0 is used, the current position is not incremented after each read.

Return Value:

Returns the database handle of the open database if successful and 0 otherwise.

Example:

```
Private Sub OpenTableCommand_Click()

   List1.Clear

  ' Mount a Database Volume
    Dim filename As String
    Dim volumeId As String

    filename = "\my documents\Nwind.cdb"
    volumeId = SregoCETPx1.MountDatabaseVolume(filename, OPEN_EXISTING)

    If volumeId <> "" Then

        List1.AddItem "Volume Mounted: " + filename
        List1.AddItem "Volume Id: " + volumeId
        List1.AddItem ""
```

```
    Else

        List1.AddItem "Unable to mount Volume: " + filename
        Exit Sub

    End If


 ' Open Database in mounted Volume
    Dim dbHandle As Long
    dbHandle = SregoCETPx1.OpenDatabaseByName(volumeId, _
                   "Employees", 0, CEDB_AUTOINCREMENT)

    If dbHandle <> 0 Then  ' make sure database was opened successfully

        List1.AddItem "Database Opened"
        List1.AddItem "     Handle: " + Str(dbHandle)

        ' do something here with the open database


        SregoCETPx1.CloseDatabase dbHandle
        List1.AddItem "Database Closed"


    End If

    SregoCETPx1.UnmountDatabaseVolume volumeId

End Sub
```

CloseDatabase

This method closes an open database. The database is closed by specifying its handle. The handle is returned by the **OpenDatabaseByName** method.

Syntax:

```
Boolean CloseDatabase( long dbHandle )
```

- **dbHandle** – handle of open database to close.

Return Value:

Returns True if successful and False otherwise.

ReadDatabaseRecord

Reads the next record from the specified database. The record properties are stored in memory and must be accessed with the **GetCurrentRecProp** set of methods. If the database was opened with the AUTOINCREMENT flag, the current record will be automatically advanced after the read. If the AUTOINCREMENT flag was not specified, then you move the current record with one of the **DatabaseSeek** methods.

Syntax:

```
Boolean ReadDatabaseRecord( long dbHandle )
```

- **dbHandle** – the handle of the database to read a record from

Return Value:

Return true if a record was successfully read and false otherwise.

Example:

```
Private Sub ReadCommand_Click()

    List1.Clear


  ' Mount a Database Volume
    Dim filename As String
    Dim volumeId As String

    filename = "\my documents\Nwind.cdb"
    volumeId = SregoCETPx1.MountDatabaseVolume(filename, OPEN_EXISTING)

    If volumeId <> "" Then

        List1.AddItem "Volume Mounted: " + filename
        List1.AddItem "Volume Id: " + volumeId
        List1.AddItem ""

    Else

        List1.AddItem "Unable to mount Volume: " + filename
        Exit Sub

    End If


  ' Open Database in mounted Volume
    Dim dbHandle As Long
    dbHandle = SregoCETPx1.OpenDatabaseByName(volumeId, _
                  "Employees", 0, CEDB_AUTOINCREMENT)

    If dbHandle <> 0 Then  ' make sure database was opened successfully

        Dim recCount As Long
        recCount = 0
```

```
' read each record in database
  While SregoCETPx1.ReadDatabaseRecord(dbHandle)
      Dim pCount As Long
      Dim recId As Long

      recId = SregoCETPx1.GetCurrentRecId
      pCount = SregoCETPx1.GetCurrentRecPropCount

      List1.AddItem Str(recCount) + " - RecId: " + _
                Str(recId) + "  PropCnt: " + Str(pCount)

      Dim i As Long

      For i = 0 To pCount - 1   ' display each property in record

          Dim pId As Long
          Dim pVal As Variant
          Dim pLen As Long
          Dim pFlags As Long
          Dim pType As Long

          pId = SregoCETPx1.GetCurrentRecPropId(i)  ' Property Id
          pVal = SregoCETPx1.GetCurrentRecPropValue(i)' Property value
          pLen = SregoCETPx1.GetCurrentRecPropLen(i)  ' Property Data Len
        ' Property Flags (CEDB_PROPNOTFOUND or CEDB_PROPDELETE)
          pFlags = SregoCETPx1.GetCurrentRecPropFlags(i)

          ' Property Type ( CEVT_I2, CEVT_UI2, CEVT_I4, CEVT_UI4,
          '                 CEVT_FILETIME, CEVT_LPWSTR, CEVT_BLOB,
          '                 CEVT_BOOL, CEVT_R8)
          pType = SregoCETPx1.GetCurrentRecPropType(i)
          Dim tempStr As String


          Select Case pType

              Case CEVT_I4, CEVT_I2, CEVT_UI4, CEVT_UI2, CEVT_R8

                  tempStr = Val(pVal)

              Case CEVT_LPWSTR

                  tempStr = pVal

              Case CEVT_BOOL

                  If pVal Then tempStr = "True" Else tempStr = "False"

              Case CEVT_BLOB
                  ' not yet supported
                  tempStr = "BLOB"

              Case CEVT_FILETIME

                  tempStr = CStr(pVal)

              Case Else

                  tempStr = "***" ' unhandled type

          End Select
```

```
                List1.AddItem ("    " + Str(pId) + " = " + tempStr)

            Next
            List1.AddItem ""
            recCount = recCount + 1
        Wend
        SregoCETPx1.CloseDatabase dbHandle
    End If
    SregoCETPx1.UnmountDatabaseVolume volumeId
End Sub
```

GetCurrentRecId

Returns the Record Id for the current record.

Syntax:

```
Long GetCurrentRecId()
```

Return Value:

The return value is the record Id for the current record.

GetCurrentRecPropCount

Returns the number of properties in the current record.

Syntax:

```
Long GetCurrentRecPropCount( long pos )
```

Return Value:

The return value is the number of properties in the current record.

GetCurrentRecPropId

Returns the property id of the specified property in the current record.

Syntax:

```
Long GetCurrentRecPropId( long pos )
```

- **Pos** – is the position of the property in the current record. Pos must be greater than or equal to 0 and must be less than the value returned by **GetCurrentRecPropCount**.

Return Value:

The return value is the property id for the specified position.

GetCurrentRecPropValue

Returns the value of the specified property in the current record.

Syntax:

```
Variant GetCurrentRecPropValue( long pos )
```

- **Pos** – is the position of the property in the current record. Pos must be greater than or equal to 0 and must be less than the value returned by **GetCurrentRecPropCount**.

Return Value:

The return value is a variant. You can use VarType in VB to determine the type of variant returned or you can use the **GetCurrentRecPropType** method.

GetCurrentRecPropBlobStr

Returns the value of the specified property in the current record. This method is similar to **GetCurrentRecPropValue** except that it can be used with properties that are Blobs storing Unicode strings. The string value returned will be automatically converted to multibyte.

Syntax:

```
BSTR GetCurrentRecPropBlobStr(long pos)
```

- **Pos** – is the position of the property in the current record. Pos must be greater than or equal to 0 and must be less than the value returned by **GetCurrentRecPropCount**.

Return Value:

The return value is a multibyte string.

GetCurrentRecPropBlob

Returns the value of the specified property in the current record. This method is similar to **GetCurrentRecPropValue** and **GetCurrentRecPropBlobStr** except that it will return a byte array containing a Blob property's contents.

Syntax:

```
boolean GetCurrentRecPropBlob(long pos,
                              VARIANT* data,
                              VARIANT* size)
```

- **Pos** – is the position of the property in the current record. Pos must be greater than or equal to 0 and must be less than the value returned by **GetCurrentRecPropCount**.
- **Data** – is a variant that will return a Byte array containing the Blob's contents.
- **Size** – is a variant that will return an integer value telling the number of bytes in the array.

Return Value:

The return value is True for success and False for failure. If the property is not a Blob, the method will return False.

Example:

```
Dim blobData as Variant
Dim blobSize as Variant

If SregoCETPx1.GetCurrentRecPropBlob(i, blobData, blobSize) Then

    For j = 0 To blobSize - 1

         Debug.Print "Blob(" + Str(j) + ") = " + Str(blobData(j))

    Next

End If
```

GetCurrentRecPropLen

Returns the size of the specified property in the current record. The method is useful for properties that are strings or blobs.

Syntax:

```
Variant GetCurrentRecPropLen( long pos )
```

- **Pos** – is the position of the property in the current record. Pos must be greater than or equal to 0 and must be less than the value returned by **GetCurrentRecPropCount**.

Return Value:

The return value is the size of a record property in bytes.

GetCurrentRecPropType

Returns the data type of the specified property in the current record.

Syntax:

```
Variant GetCurrentRecPropType( long pos )
```

- **Pos** – is the position of the property in the current record. Pos must be greater than or equal to 0 and must be less than the value returned by **GetCurrentRecPropCount**.

Return Value:

The return value is long with one of the following value:

CEVT_I2 (2) - short
CEVT_UI2 (18) - unsigned short
CEVT_I4 (3) - long
CEVT_UI4 (19) - unsigned long
CEVT_FILETIME (64) - DateTime
CEVT_LPWSTR (31) - String
CEVT_BLOB (65) - BLOB
CEVT_BOOL (11) - Boolean
CEVT_R8 (5) - Double

GetSortInfo

This method returns the sort specification for a specified database. The sort specification lists up to 4 sort items including their sort type and sort flags. A database id is used to specify a database to query. **GetDatabaseId** and **CreateNewDatabase** both return the database id.

Syntax:

```
boolean GetSortInfo(BSTR volumeId, long databaseId,
                    VARIANT* sortIdList,
                    VARIANT* sortTypeList,
                    VARIANT* sortFlagsList,
                    VARIANT* count)
```

- **volumeId** – volume id to create the database in. Use an empty string to use the internal volume.

- **databaseId** – the id of the database to delete. Use the **GetDatabaseId** method to get the database id.
- **sortIdList** – returns a variant array of long integer values containing the property ids of each indexed attribute.
- **sortTypeList** – returns a variant array of long integer values containing the property type of each index attribute.

  CEVT_I2 (2) - short
  CEVT_UI2 (18) - unsigned short
  CEVT_I4 (3) - long
  CEVT_UI4 (19) - unsigned long
  CEVT_FILETIME (64) - DateTime
  CEVT_LPWSTR (31) - String
  CEVT_BLOB (65) - BLOB
  CEVT_BOOL (11) - Boolean
  CEVT_R8 (5) – Double

- **sortFlagList** – returns a variant array of long integer values containing the sort flags. The value can be one or more of the following values ORed together:

  CEDB_SORT_DEFAULT (0) - Ascending
  CEDB_SORT_DESCENDING (1)
  CEDB_SORT_CASEINSENSITIVE (2)
  CEDB_SORT_UNKNOWNFIRST (4)
  CEDB_SORT_GENERICORDER (8)

Example:

```
  Dim sortIdList As Variant
  Dim sortTypeList As Variant
  Dim sortFlagsList As Variant
  Dim sortItemCount As Variant
  Dim i As Long

 SregoCETPx1.GetSortInfo "", databaseId, _
                sortIdList, sortTypeList, _
                sortFlagsList, sortItemCount

For i = 0 To sortItemCount - 1

    List1.AddItem "Sort Prop Id: " + Str(sortIdList(i))
    List1.AddItem "Prop Type: " + Str(sortTypeList(i))
    List1.AddItem "Flags: " + Str(sortFlagsList(i))
   List1.AddItem ""

Next
```

CreateNewRecord

This method creates a new record in memory that can be populated and then written to an open database. Calling this method will always cancel any record created by a previous call. Use the **SetNewRecordProperty** or the **SetNewRecordPropertyEx** method to populate the record and **WriteDatabaseRecord** to add the record to the database.

Syntax:

```
boolean CreateNewRecord( long propertyCount )
```

- **propertyCount** - the number of properties that the record will have.

Return Value:

Returns True if successful and False otherwise.

Example:

```
SregoCETPx1.CreateNewRecord 9

SregoCETPx1.SetNewRecordProperty 0, CEVT_I2, -12345
SregoCETPx1.SetNewRecordProperty 1, CEVT_UI2, 12345
SregoCETPx1.SetNewRecordProperty 2, CEVT_I4, -123456789
SregoCETPx1.SetNewRecordProperty 3, CEVT_UI4, 123456789
SregoCETPx1.SetNewRecordProperty 4, CEVT_FILETIME, Date
SregoCETPx1.SetNewRecordProperty 5, CEVT_LPWSTR, "String Test"
SregoCETPx1.SetNewRecordProperty 6, CEVT_BLOB, "Blob String Test"
SregoCETPx1.SetNewRecordProperty 7, CEVT_BOOL, True
SregoCETPx1.SetNewRecordProperty 8, CEVT_R8, 1234.5678

Dim recId As Long

recId = SregoCETPx1.WriteDatabaseRecord(dbHandle)

If recId <> 0 Then

   List1.AddItem "Record Added - id: " + Str(recId)

End If
```

SetNewRecordProp

This method sets the properties in a record created by the **CreateNewRecord** method. This method uses the property position (0 to property count -1) as the property id. The **SetNewRecordPropEx** method will let you specify a property id other than the position. After using the **CreateNewRecord** method (which specifies the number of attributes in the new record), you must populate each attribute with the **SetNewRecordProp** or **SetNewRecorPropEx** methods.

Syntax:

```
Boolean SetNewRecordProp( long pos, long dbType,
                          variant value )
```

- **Pos** - the property in the record to set. The pos value must be 0 to Property Count -1. The property count is specified in the **CreateNewRecord** call. With **SetNewRecordProp**, the position is always used as the Property Id. If you wish to specify a property id that is different from the position, use **SetNewRecordPropEx**.
- **dbType** - the type of the property. It can be one of the following values:

  CEVT_I2 (2) - short
  CEVT_UI2 (18) - unsigned short
  CEVT_I4 (3) - long
  CEVT_UI4 (19) - unsigned long
  CEVT_FILETIME (64) - DateTime
  CEVT_LPWSTR (31) - String
  CEVT_BLOB (65) - BLOB
  CEVT_BOOL (11) - Boolean
  CEVT_R8 (5) – Double

- **Value** – is a variant used to specify the value to store in the property. The value you specify must correspond with the dbType parameter.

Return Value:

Returns True if successful and False otherwise.

Example:

See CreateNewRecord

SetNewRecordPropEx

This method sets the properties in a record created by the **CreateNewRecord** method. This method differs from **SetNewRecordProp** in that the property id can be specified independently from the position.

Syntax:

```
Boolean SetNewRecordPropEx( long pos, long propId,
                            long dbType, variant value )
```

- **Pos** - the position of the record property to set in the record. The pos value must be 0 to Property Count -1.
- **propId** – the property id of the property to set the value for.
- **dbType** - the type of the property. It can be one of the following values:

  CEVT_I2 (2) - short
  CEVT_UI2 (18) - unsigned short
  CEVT_I4 (3) - long
  CEVT_UI4 (19) - unsigned long
  CEVT_FILETIME (64) - DateTime
  CEVT_LPWSTR (31) - String
  CEVT_BLOB (65) - BLOB
  CEVT_BOOL (11) - Boolean
  CEVT_R8 (5) – Double

- **Value** – is a variant used to specify a value to store in the property. The value you specify must correspond with the dbType parameter.

If the dbType is CEVT_BLOB, then the value must be either a String or Byte Array. If a String is used, the string value will be converted to Unicode, and the Unicode string value will be stored as the value. If the value is a Byte array (something like Dim bArray(5) as Byte), then the contents of the array will be stored as the value.

Return Value:

Returns True if successful and False otherwise.

WriteDatabaseRecord

This method will write the record created in memory with the **CreateNewRecord** method and populated with calls to the **SetNewRecordProp** or **SetNewRecordPropEx** method to a specified database.

Syntax:

```
long WriteDatabaseRecord( long dbHandle )
```

- **dbHandle** – a handle to the database to wish to write the record to.

Example:

See CreateNewRecord

Return Value:

Returns the record id of the newly added record.

UpdateDatabaseRecord

This method will write the record created in memory with the **CreateNewRecord** method and populated with calls to the **SetNewRecordProp** and **SetNewRecordPropEx** methods to a specified database.

Syntax:

```
long UpdateDatabaseRecord( long dbHandle,
                           long recordId )
```

- **dbHandle** – a handle to the database to wish to update the record in.
- **recordId** – the id of the record in the specified database to update.

Return Value:

Returns record id of the updated record.

CreateNewSortSpec

This method creates a new sort specification which will be used when the **CreateNewDatabase** method is used. The individual items in the sort specification must be set with the **SetSortSpec** or **SetSortSpecEx** methods.

The **CreateNewSortSpec** will clear any previously defined sort specification.

The **CreateNewSortSpec** must be called before the **CreateNewDatabase** method.

Syntax:

```
boolean CreateNewSortSpec(long sortOrderCount)
```

- **sortOrderCount** – the number of sort properties to use. The SetSortSpec or SetSortSpecEx methods must be called to set each sort property.

Return Value:

Returns True for success and False for failure.

Example:

```
Dim databaseId As Long

SregoCETPx1.CreateNewSortSpec 4

SregoCETPx1.SetSortSpec 0, CEVT_I2, CEDB_SORT_DEFAULT
SregoCETPx1.SetSortSpec 1, CEVT_LPWSTR, CEDB_SORT_DEFAULT
SregoCETPx1.SetSortSpec 2, CEVT_I4, CEDB_SORT_DEFAULT
SregoCETPx1.SetSortSpec 3, CEVT_I2, CEDB_SORT_DEFAULT

databaseId = SregoCETPx1.CreateNewDatabase("", "TEST_DB", 123, 0)
```

SetSortSpec

This method is used to specify each of the sort properties defined by the **CreateNewSortSpec** method. The sort specifications are used to define the indexes for a newly created database.

The **SetSortSpec** method is somewhat obsolete because the position of the sort specification must be for the as the position of the attribute in the table that is being sorted. While this limitation is overcome by having the sort properties are the first four properties of the table, it is still limiting. The **SetSortSpecEx** method allows the position of the sort spec to be set independently from the property id.

Syntax:

```
boolean SetSortSpec(long pos, long sortType, long flags)
```

- **Pos** - the property in the sort specification to set. The pos value must be 0 to Sort Order Count -1. The sort order count is specified in the CreateNewSortSpec call. With SetSortSpec, the position is always used as the Property Id of the attribute to sort. If you wish to specify a property id that is different from the position, use SetSetSpecEx.
- **sortType** – the type of the sort property. The value must be one of the following:

  CEVT_I2 (2) - short
  CEVT_UI2 (18) - unsigned short
  CEVT_I4 (3) - long
  CEVT_UI4 (19) - unsigned long
  CEVT_FILETIME (64) - DateTime
  CEVT_LPWSTR (31) - String
  CEVT_BLOB (65) - BLOB
  CEVT_BOOL (11) - Boolean
  CEVT_R8 (5) – Double

- **flags** – The type of sorting to perform. The value can be one of the following or a combination of the flags ORed together:

  CEDB_SORT_DEFAULT (0) - Ascending
  CEDB_SORT_DESCENDING (1)
  CEDB_SORT_CASEINSENSITIVE (2)
  CEDB_SORT_UNKNOWNFIRST (4)
  CEDB_SORT_GENERICORDER (8)

Return Value:

Returns True for success and False for failure.

Example:

```
Dim databaseId As Long

SregoCETPx1.CreateNewSortSpec 4

SregoCETPx1.SetSortSpec 0, CEVT_I2, CEDB_SORT_DEFAULT
SregoCETPx1.SetSortSpec 1, CEVT_LPWSTR, CEDB_SORT_DEFAULT
SregoCETPx1.SetSortSpec 2, CEVT_I4, CEDB_SORT_DEFAULT
SregoCETPx1.SetSortSpec 3, CEVT_I2, CEDB_SORT_DEFAULT

databaseId = SregoCETPx1.CreateNewDatabase("", "TEST_DB", 123, 0)
```

SetSortSpecEx

This method is used to specify each of the sort properties defined by the **CreateNewSortSpec** method. The sort specifications are used to define the indexes for a newly created database. This method differs from **SetSortSpec** in that it allows the property type to be specified independently from the sort specification position.

Syntax:

```
boolean SetSortSpecEx(long pos, long sortPropId,
                      long sortType, long flags)
```

- **Pos** - the property in the sort specification to set. The pos value must be 0 to Sort Order Count -1. The sort order count is specified in the **CreateNewSortSpec** call. Unlike **SetSortSpec** where the position is always used as the Property Id of the attribute to sort, **SetSortSpecEx** allows the property id to be different than the sort spec position.
- **sortPropId** – the property id of the property to sort on.
- **sortType** – the type of the sort property. The value must be one of the following:

  CEVT_I2 (2) - short
  CEVT_UI2 (18) - unsigned short
  CEVT_I4 (3) - long
  CEVT_UI4 (19) - unsigned long
  CEVT_FILETIME (64) - DateTime
  CEVT_LPWSTR (31) - String
  CEVT_BLOB (65) - BLOB
  CEVT_BOOL (11) - Boolean
  CEVT_R8 (5) – Double

- **flags** – The type of sorting to perform. The value can be one of the following or a combination of the flags ORed together:

  CEDB_SORT_DEFAULT (0) - Ascending
  CEDB_SORT_DESCENDING (1)
  CEDB_SORT_CASEINSENSITIVE (2)
  CEDB_SORT_UNKNOWNFIRST (4)
  CEDB_SORT_GENERICORDER (8)

Return Value:

Returns True for success and False for failure.

CreateNewDatabase

This method is used to create a new database. If a sort specification is needed for the new table, the **CreateSortSpec** and **SetSortSpec** or **SetSortSpecEx** methods must be used.

Syntax:

```
long CreateNewDatabase(BSTR volumeId, BSTR dbName,
                       long dbType, long flags)
```

- **volumeId** – volume id to create the database in. Use an empty string ("") to use the default database.
- **dbName** – the name of the database to create. The name is limited to 32 characters.
- **dbType** – the user-defined type identifier for the database. Use 0 if a type identifier is not needed.
- **Flags** – The flags value can be set to 0 or to the following value:

CEDB_NOCOMPRESS (65536)

Return Value:

The return value is the database id of the newly created database or one of the following values:

ERROR_DISK_FULL (112)
ERROR_INVALID_PARAMETER (87)
ERROR_DUP_NAME (52)

Example:

```
Dim databaseId As Long

SregoCETPx1.CreateNewSortSpec 4

SregoCETPx1.SetSortSpec 0, CEVT_I2, CEDB_SORT_DEFAULT
SregoCETPx1.SetSortSpec 1, CEVT_LPWSTR, CEDB_SORT_DEFAULT
SregoCETPx1.SetSortSpec 2, CEVT_I4, CEDB_SORT_DEFAULT
SregoCETPx1.SetSortSpec 3, CEVT_I2, CEDB_SORT_DEFAULT

databaseId = SregoCETPx1.CreateNewDatabase("", "TEST_DB", 123, 0)
```

DeleteDatabase

This method is used to delete a database from the default volume or a mounted volume.

Syntax:

```
boolean DeleteDatabase(BSTR volumeId,
                       long databaseId)
```

- **volumeId** – volume id to create the database in. Use an empty string to use the default volume.
- **databaseId** – the id of the database to delete. Use the GetDatabaseId method to get the database id.

Return Value:

Returns True if successful and False otherwise.

Example:

```
Dim databaseId As Long

databaseId = SregoCETPx1.CreateNewDatabase(volumeId, "TEST_DB", 123, 0)

If databaseId = 0 Then

    MsgBox "Unable to create new database"
    MsgBox SregoCETPx1.GetLastCeErrorMessage


    databaseId = SregoCETPx1.GetDatabaseId(volumeId, "TEST_DB")

    SregoCETPx1.DeleteDatabase volumeId, databaseId

    Exit Sub

End If
```

GetDatabaseId

This method returns the database id for a specified database.

Syntax:

```
long GetDatabaseId(BSTR volumeId, BSTR dbName)
```

- **volumeId** – volume id you wish to search. Use an empty string to use the default volume.
- **dbName** – the name of the database to return the database id for..

Return Value:

The Database id is returned for the specified database. If the database cannot be found, 0 will be returned.

DeleteDatabaseRecord

This method deletes a specific record from a database. The record to delete is specified by its record id. Several methods return a record id: **GetCurrentRecId**, **CreateNewRecord**, and **UpdateDatabaseRecord**.

Syntax:

```
boolean DeleteDatabaseRecord(long dbHandle,
                             long recordId)
```

- **dbHandle** – the handle of open database to delete a record from.
- **recordId** – the id of the record to delete.

Return Value:

The return value is True if the record is successfully deleted and False otherwise.

DatabaseSeekToBeginning

Seeks to the beginning record of the specified database. The database is specified by its handle which is returned by the **OpenDatabaseByName** method.

Syntax:

```
Boolean DatabaseSeekToBeginning( long dbHandle )
```

- **dbHandle** – the handle of the database to seek.

Return Value:

The return value is TRUE for success and FALSE for failure.

DatabaseSeekToEnd

Seeks to the last record of the specified database. The database is specified by its handle which is returned by the **OpenDatabaseByName** method.

Syntax:

```
Boolean DatabaseSeekToEnd( long dbHandle )
```

- **dbHandle** – the handle of the database to seek.

Return Value:

The return value is TRUE for success and FALSE for failure.

DatabaseSeekToCeoid

Seeks to the beginning record of the specified database. The database is specified by its handle which is returned by the **OpenDatabaseByName** method.

Syntax:

```
Boolean DatabaseSeekToCeoid( long dbHandle,
                             long ceoid )
```

- **dbHandle** – the handle of the database to seek.
- **Ceoid** – the id of the record to seek to.

Return Value:

The return value is TRUE for success and FALSE for failure.

DatabaseSeekFromCurrent

This method seeks a specific number of records from the current record position. The database is specified by its handle which is returned by the **OpenDatabaseByName** method.

Syntax:

```
long DatabaseSeekFromCurrent(long dbHandle,
                             long recordCount)
```

- **dbHandle** – a handle to the database to wish to update the record in.
- **recordCount** – the number of records to seek. The value can be negative or positive.

Return Value:

The return value is 1 for success and 0 for failure.

DatabaseSeekRecord

This method seeks to a specific record in the database. To use the **DatabaseSeekRecord** method, you must meet the following criteria:

- Open the database with the `CEDB_NO_AUTOINCREMENT` mode. If the auto-increment mode is used, then you will potentially skip records when reading multiple records matching the search criteria.

- The database must be opened with the appropriate sort property specified. The **DatabaseSeekRecord** can only use the sort property specified when the database was opened.

The **DatabaseSeekRecord** sets the current record to a record matching the search criteria (if a record meets the search criteria). You must still read the record with **ReadDatabaseRecord**.

The database is specified by its handle which is returned by the **OpenDatabaseByName** method.

Syntax:

```
long DatabaseSeek(long dbHandle, long mode,
                  long propId, long propType,
                  VARIANT propValue)
```

- **dbHandle** – a handle to the database to wish to update the record in.
- **mode** – the seek mode. The seek mode must be one of the following values:

  CEDB_SEEK_VALUESMALLER (16)
  CEDB_SEEK_VALUEFIRSTEQUAL (32)
  CEDB_SEEK_VALUEGREATER (64)
  CEDB_SEEK_VALUENEXTEQUAL (128)

- **propId** – the property Id of the sort property.
- **propValue** – the value to search for. The value must be of the appropriate type.

Return Value:

The return value is 0 if a record matching the search criteria is not found; otherwise, the return value is the id of the record.

Example:

```
    List1.Clear

    Dim databaseId As Long

    SregoCETPx1.CreateNewSortSpec 4
    SregoCETPx1.SetSortSpec 0, CEVT_I2, CEDB_SORT_DEFAULT
    SregoCETPx1.SetSortSpec 1, CEVT_LPWSTR, CEDB_SORT_DEFAULT
    SregoCETPx1.SetSortSpec 2, CEVT_I4, CEDB_SORT_DEFAULT
    SregoCETPx1.SetSortSpec 3, CEVT_I2, CEDB_SORT_DEFAULT

    databaseId = SregoCETPx1.CreateNewDatabase("", "TEST_DB", 123, 0)

    If databaseId = 0 Then

        MsgBox "Unable to create new database"
        MsgBox SregoCETPx1.GetLastCeErrorMessage


        databaseId = SregoCETPx1.GetDatabaseId("", "TEST_DB")

        SregoCETPx1.DeleteDatabase "", databaseId

        Exit Sub

    End If

    Dim dbHandle As Long


' Open Database in mounted Volume

    dbHandle = SregoCETPx1.OpenDatabaseByName("", "TEST_DB", _
                          0, CEDB_NO_AUTOINCREMENT) ' no auto incement

    If dbHandle = 0 Then  ' make sure database was opened successfully

        List1.AddItem "Unable to open Database"
        Exit Sub

    End If

    List1.AddItem "Database Opened"
    List1.AddItem "     Handle: " + Str(dbHandle)

    Dim i As Long
    Dim j As Long

    Dim buffer As String
    Dim longBuffer As Long

    For j = 1 To 5
        For i = 1 To 10

             SregoCETPx1.CreateNewRecord 5

             SregoCETPx1.SetNewRecordProperty 0, CEVT_I2, i * 10
             buffer = "String Test " & Str(i)
             SregoCETPx1.SetNewRecordProperty 1, CEVT_LPWSTR, buffer
             longBuffer = Int(Rnd() * 1000)
             SregoCETPx1.SetNewRecordProperty 2, CEVT_I4, longBuffer
             SregoCETPx1.SetNewRecordProperty 3, CEVT_I2, 10 - i
             SregoCETPx1.SetNewRecordProperty 4, CEVT_BLOB, "Blob String " & Str(i)

             Dim recId As Long

             recId = SregoCETPx1.WriteDatabaseRecord(dbHandle)

             If recId <> 0 Then

                 List1.AddItem "Record Added - id: " + Str(recId)
```

```
            Else

                List1.AddItem "Unable to add record"

            End If

            List1.AddItem ""

        Next
    Next

' Read Records

    Dim seekValue As Integer

    seekValue = 10 ' value to search for

    If SregoCETPx1.DatabaseSeek(dbHandle, CEDB_SEEK_VALUEFIRSTEQUAL, _
                                0, CEVT_I2, seekValue) = 0 Then

        List1.AddItem "Unable to seek to record"

            MsgBox SregoCETPx1.GetLastCeErrorMessage

    Else

        Dim recCount As Long
        recCount = 0

        Do

           If SregoCETPx1.ReadDatabaseRecord(dbHandle) Then


                Dim pCount As Long

                recId = SregoCETPx1.GetCurrentRecId
                pCount = SregoCETPx1.GetCurrentRecPropCount

                List1.AddItem Str(recCount) + " - RecId: " + Str(recId) + _
                                            "  PropCnt: " + Str(pCount)

                For i = 0 To pCount - 1   ' display each property in record

                    DoEvents

                    Dim pId As Long
                    Dim pVal As Variant
                    Dim pLen As Long
                    Dim pFlags As Long
                    Dim pType As Long

                    pId = SregoCETPx1.GetCurrentRecPropId(i)       ' Property Id
                    pVal = SregoCETPx1.GetCurrentRecPropValue(i)   ' Property value
                    pLen = SregoCETPx1.GetCurrentRecPropLen(i)     ' Property Data Length
                    pFlags = SregoCETPx1.GetCurrentRecPropFlags(i)
                                    ' Property Flags (CEDB_PROPNOTFOUND or
                              CEDB_PROPDELETE)

                    pType = SregoCETPx1.GetCurrentRecPropType(i)
                                    ' Property Type ( CEVT_I2, CEVT_UI2, CEVT_I4,
                                    ' CEVT_UI4, CEVT_FILETIME, CEVT_LPWSTR, CEVT_BLOB,
                                    ' CEVT_BOOL, CEVT_R8)

                    Dim tempStr As String


                    Select Case pType

                        Case CEVT_I4, CEVT_I2, CEVT_UI4, CEVT_UI2, CEVT_R8
```

```
                        tempStr = Val(pVal)

                    Case CEVT_LPWSTR

                        tempStr = pVal

                    Case CEVT_BLOB

                        tempStr = SregoCETPx1.GetCurrentRecPropBlobStr(i)


                    Case Else

                        tempStr = "***" ' unhandled type

                End Select

                List1.AddItem ("    " + Str(pId) + " = " + tempStr)

                DoEvents

            Next

            List1.AddItem ""

            recCount = recCount + 1
        Else

            List1.AddItem "Unable to read record."

        End If

    Loop While SregoCETPx1.DatabaseSeek(dbHandle, CEDB_SEEK_VALUENEXTEQUAL, _
                                        0, CEVT_I2, seekValue)

End If


SregoCETPx1.CloseDatabase dbHandle

databaseId = SregoCETPx1.GetDatabaseId("", "TEST_DB")

SregoCETPx1.DeleteDatabase "", databaseId
```

DatabaseChangeSortSpec

This method can be used to change the sort spec on an existing database.

Syntax:

```
boolean DatabaseChangeSortSpec(long databaseId)
```

- **databaseId** – a handle to the database to wish to update the record in. Use the **GetDatabaseId** method to determine the databaseId from a database name.

Before the **DatabaseChangeSortSpec** method can be used, the **CreateSortSpec** and **SetSortSpecEx** methods must be used to define the new sort spec.

Since changing the sort spec causes the device to reindex the database according to the new criteria, the time required to change the sort spec will depend on the amount of work required to process the data.

Example:

```
Dim databaseId As Long

databaseId = SregoCETPx1.GetDatabaseId("", "TEST_DB")

SregoCETPx1.SetSortSpecEx 0, 1, 31, 2

If SregoCETPx1.DatabaseChangeSortSpec(databaseId) Then

   MsgBox "Success"

Else

   MsgBox "Fail"
   MsgBox SregoCETPx1.GetLastCeErrorMessage

End If
```

Working with .cdb files created from .mdb files

When you convert a .mdb file to a .cdb file (when you copy it to the device with ActiveSync), there are several additional tables that get created:

MSysTables
MSysFields
MSysIndexes
MSysProcs

You can use the two of these to get Metadata from the .cdb file:

MSysTables
- TableName
- TableId
- TableFlags (0 for user, 3 for system)

MSysFields
- TableId
- FieldName
- FieldId
- Len
- Type (2=INT16, 18=UINT16, 3=INT32, 19=UINT32, 64=FileTIme, 31=String, 65=Blob, 11=BOOL, 5=DOUBLE)

Here are two sample functions to get the meta data from the tables:

```
Function GetTableListFromCDB(cdbFile As String, _
                             tableId() As Long, _
                             tableName() As String, _
                             tableFlags() As Long, _
                             tableCount As Integer) As Boolean

    tableCount = 0

  ' Mount a Database Volume
    Dim filename As String
    Dim volumeId As String

    volumeId = SregoCETPx1.MountDatabaseVolume(cdbFile, OPEN_EXISTING)

    If volumeId = "" Then
        MsgBox "Unable to mount database volume"
        GetTableListFromCDB = False
        Exit Function
    End If


 ' Open Database in mounted Volume
    Dim dbHandle As Long
    dbHandle = SregoCETPx1.OpenDatabaseByName(volumeId, _
                          "MSysTables", 0, CEDB_AUTOINCREMENT)
```

```
    If dbHandle <> 0 Then  ' make sure database was opened successfully

        While SregoCETPx1.ReadDatabaseRecord(dbHandle) ' read each record

            tableName(tableCount) = SregoCETPx1.GetCurrentRecPropValue(0)

            tableId(tableCount) = SregoCETPx1.GetCurrentRecPropValue(1)
            tableFlags(tableCount) = SregoCETPx1.GetCurrentRecPropValue(2)

            tableCount = tableCount + 1

        Wend

        SregoCETPx1.CloseDatabase dbHandle

    End If

    SregoCETPx1.UnmountDatabaseVolume volumeId


    GetTableListFromCDB = True

End Function


Function GetFieldListFromCDB(cdbFile As String, tableId As Long, _
                             fieldName() As String, fieldId() As Long, _
                             fieldLen() As Long, fieldType() As Long, _
                             fieldCount As Integer) As Boolean

    fieldCount = 0

  ' Mount a Database Volume
    Dim filename As String
    Dim volumeId As String

    volumeId = SregoCETPx1.MountDatabaseVolume(cdbFile, OPEN_EXISTING)

    If volumeId = "" Then
        MsgBox "Unable to mount database volume"
        GetFieldListFromCDB = False
        Exit Function
    End If

    Dim recTableId As Long


 ' Open Database in mounted Volume
    Dim dbHandle As Long
    Dim id As Long
    dbHandle = SregoCETPx1.OpenDatabaseByName(volumeId, _
                "MSysFields", 0, CEDB_AUTOINCREMENT)

    If dbHandle <> 0 Then  ' make sure database was opened successfully

        While SregoCETPx1.ReadDatabaseRecord(dbHandle) ' read each record

            recTableId = SregoCETPx1.GetCurrentRecPropValue(0)

            If recTableId = tableId Then

                fieldName(fieldCount) = SregoCETPx1.GetCurrentRecPropValue(1)
```

```
                id = SregoCETPx1.GetCurrentRecPropValue(2)
                fieldId(fieldCount) = ShiftRight(id, 24)

                fieldLen(fieldCount) = SregoCETPx1.GetCurrentRecPropValue(3)
                fieldType(fieldCount) = SregoCETPx1.GetCurrentRecPropValue(4)

                fieldCount = fieldCount + 1

            End If

        Wend

        SregoCETPx1.CloseDatabase dbHandle

    End If

    SregoCETPx1.UnmountDatabaseVolume volumeId


    GetFieldListFromCDB = True

End Function

' Power2 from http://search.devx.com/
Function Power2(ByVal exponent As Long) As Long
    Static res(0 To 31) As Long
    Dim i As Long

    ' rule out errors
    If exponent < 0 Or exponent > 31 Then Err.Raise 5

    ' initialize the array at the first call
    If res(0) = 0 Then
        res(0) = 1
        For i = 1 To 30
            res(i) = res(i - 1) * 2
        Next
        ' this is a special case
        res(31) = &H80000000
    End If

    ' return the result
    Power2 = res(exponent)

End Function

' ShiftRight from http://search.devx.com/
Function ShiftRight(ByVal value As Long, ByVal times As Long) As Long
    ' we need to create a mask of 1's corresponding to the
    ' digits in VALUE that will be retained in the result
    Dim mask As Long, signBit As Long

    ' return zero if too many times
    If times >= 32 Then Exit Function
    ' return the value if zero times
    If times = 0 Then ShiftRight = value: Exit Function

    ' evaluate the sign bit in advance
    signBit = (value < 0) And Power2(31 - times)
    ' create a mask with 1's for the digits that will be preserved
    If times < 31 Then
        ' if times=31 then the mask is zero
        mask = Not (Power2(times) - 1)
```

```
    End If
    ' clear all the digits that will be discarded, and
    ' also clear the sign bit
    value = (value And &H7FFFFFFF) And mask
    ' do the shift, without any problem, and add the sign bit
    ShiftRight = (value \ Power2(times)) Or signBit
End Function
```

To use these function, you do something like this:

```
    List1.Clear

    Dim tableName(100) As String
    Dim tableId(100) As Long
    Dim tableFlags(100) As Long  ' 0 - user,  3 - system
    Dim tableCount As Integer

    Dim fieldId(100) As Long
    Dim fieldName(100) As String
    Dim fieldLen(100) As Long
    Dim fieldType(100) As Long
    Dim fieldCount As Integer

    Dim i As Integer
    Dim j As Integer

    tableCount = 0

    Dim cdbFile As String

    cdbFile = "\my documents\gps.cdb"

    If GetTableListFromCDB(cdbFile, tableId, tableName, _
                           tableFlags, tableCount) Then

        For i = 0 To tableCount - 1

            If tableFlags(i) = 0 Then  ' only show user tables
                List1.AddItem "Table: " + tableName(i) + _
                         "  ID: " + Str(tableId(i))

                If GetFieldListFromCDB(cdbFile, tableId(i), fieldName, _
                          fieldId, fieldLen, fieldType, fieldCount) Then

                    For j = 0 To fieldCount - 1

                        List1.AddItem ("    " + Str(fieldId(j)) + _
                              " - " + fieldName(j) + "  Type: " + _
                              Str(fieldType(j)) + _
                              "  Len: " + Str(fieldLen(j))) + _
                              "  ID: " + Str(fieldId(j))

                    Next

                    List1.AddItem ""

               End If

            End If
```

```
        Next


    End If
```

The output looks like this:

```
Table: Note ID: 805306398
  1 - Latitude Type: 8 Len: 8 ID: 1
  2 - Text Type: 12 Len: 80 ID: 2
  3 - Max PDOP Type: 8 Len: 8 ID: 3
  4 - Max HDOP Type: 8 Len: 8 ID: 4
  5 - Corr Type Type: 12 Len: 36 ID: 5
  6 - Rcvr Type Type: 12 Len: 36 ID: 6
  7 - GPS Date Type: 93 Len: 16 ID: 7
  8 - GPS Time Type: 93 Len: 16 ID: 8
  9 - Update Status Type: 12 Len: 36 ID: 9
  10 - Feat Name Type: 12 Len: 20 ID: 10
  11 - Datafile Type: 12 Len: 20 ID: 11
  12 - Unfilt Pos Type: 8 Len: 8 ID: 12
  13 - Filt Pos Type: 8 Len: 8 ID: 13
  14 - Data Dictionary Name Type: 12 Len: 20 ID: 14
  15 - GPS Week Type: 4 Len: 4 ID: 15
  16 - GPS Second Type: 8 Len: 8 ID: 16
  17 - Point_ID Type: 4 Len: 4 ID: 17
  0 - Longitude Type: 8 Len: 8 ID: 0
```

This code sample gets a list of all user tables in the .cdb and display them in a listbox along with each field in that table. The key things here are the TableId and the FieldId. These number are what tie the metadata to the real records in the table.

If you are really going to read a real table, you would do something like the following. My sample .cdb file is gps.cdb and it has a table in it called note.

```
' Mount a Database Volume
    Dim filename As String
    Dim volumeId As String


    volumeId = SregoCETPx1.MountDatabaseVolume(cdbFile, OPEN_EXISTING)

    If volumeId = "" Then
        List1.AddItem "Unable to mount Volume: " + cdbFile
        Exit Sub
    End If

 ' Open Database in mounted Volume
    Dim dbHandle As Long
    dbHandle = SregoCETPx1.OpenDatabaseByName(volumeId, _
                "note", 0, CEDB_AUTOINCREMENT)

    If dbHandle <> 0 Then  ' make sure database was opened successfully

        Dim recCount As Integer
        recCount = 0

        While SregoCETPx1.ReadDatabaseRecord(dbHandle) ' read each record

            Dim pCount As Long
            Dim recId As Long
```

```
            recId = SregoCETPx1.GetCurrentRecId
            pCount = SregoCETPx1.GetCurrentRecPropCount

            List1.AddItem Str(recCount) + " - RecId: " + _
                    Str(recId) + "  PropCnt: " + Str(pCount)


            For i = 0 To pCount - 1    ' display each property in record

             Dim pId As Long
                Dim pVal As Variant
                Dim pLen As Long
                Dim pFlags As Long
                Dim pType As Long

                pId = SregoCETPx1.GetCurrentRecPropId(i)
                pVal = SregoCETPx1.GetCurrentRecPropValue(i)
                pLen = SregoCETPx1.GetCurrentRecPropLen(i)
                pFlags = SregoCETPx1.GetCurrentRecPropFlags(i)
                pType = SregoCETPx1.GetCurrentRecPropType(i)

                Dim tempstr As String


                Select Case pType

                    Case CEVT_I4, CEVT_I2, CEVT_UI4, CEVT_UI2, CEVT_R8

                        tempstr = Val(pVal)

                    Case CEVT_LPWSTR

                        tempstr = pVal

                    Case CEVT_BOOL

                        If pVal Then tempstr = "True" Else tempstr = "False"

                    Case CEVT_BLOB

                        tempstr = SregoCETPx1.GetCurrentRecPropBlobStr(i)

'

                    Case CEVT_FILETIME

                        tempstr = CStr(pVal)

                    Case Else

                        tempstr = "***" ' unhandled type

                End Select

                List1.AddItem ("    " + Str(pId) + " = " + tempstr)

                Next


                recCount = recCount + 1

        Wend
```

```
        SregoCETPx1.CloseDatabase dbHandle

    End If

    SregoCETPx1.UnmountDatabaseVolume volumeId
```

The output looks like this:

```
0 - RecId:  805306419  PropCnt:  18
   0 = 170.981562551
   1 = -44.989698305
   2 = GPS PFTools SDK V2.01
   3 = 4.33935880661011
   4 = 2.99778294563293
   5 = Uncorrected
   6 = GeoXT
   7 = 5/27/2006 6:00:00 PM
   8 = 3:07:06 PM
   9 = New
   10 = Note
   11 = Papakaio_Sub.ssf
   12 = 0
   13 = 0
   14 =
   15 = 1377
   16 = 97640
   17 = 28
1 - RecId:  805306420  PropCnt:  18
   0 = 170.981563698
   1 = -44.989698691
   2 = GPS PFTools SDK V2.01
   3 = 4.40460824966431
   4 = 2.97817635536194
   5 = Uncorrected
   6 = GeoXT
   7 = 5/27/2006 6:00:00 PM
   8 = 3:07:06 PM
   9 = New
   10 = Note
   11 = Papakaio_Sub.ssf
   12 = 0
   13 = 0
   14 =
   15 = 1377
   16 = 97640
   17 = 26
```

The property id of each field in the record can be looked up in the FieldId list populated by the previous calls.

Registry Methods

This section contains method to read and write to the Window CE device's registry.

RegGetValue

This method returns the data for a specific value in the device's registry.

Syntax:

```
VARIANT RegGetValue(long key, BSTR subKeyName,
                    BSTR valueName)
```

- **Key** – the Registry Key to query. The key value must be one of the following:

  HKEY_CLASSES_ROOT (0)
  HKEY_CURRENT_USER (1)
  HKEY_LOCAL_MACHINE (2)
  HKEY_USERS (3)

- **subKeyName** – the name of the sub-key to query.
- **valueName** – the value name to query. If this value is an empty string, the default value will be returned.

Return Value:

The method returns a variant containing the registry value. Use VarType to determine the type of the variant.

Example:

```
Dim v As Variant

v = SregoCETPx1.RegGetValue(HKEY_LOCAL_MACHINE, _
        "Software\Srego\test", "TestValue")

Debug.Print "VarType: " & VarType(v)
```

To read a binary value from the registry:

```
Dim v As Variant
Dim i As Long

v = SregoCETPx1.RegGetValue(HKEY_LOCAL_MACHINE, _
          "Software\Srego\test", "BinaryValue")

If VarType(v) = vbArray Or vbByte Then
```

```
        For i = 0 To UBound(v) - 1
              List1.AddItem v(i)
        Next
    End If
```

RegSetStringValue

This method sets a value in the device's registry. If the value or key does not exist, it will be created. This method is used to set value of a String type.

Syntax:

```
boolean RegSetStringValue(long key, BSTR subKeyName,
                          BSTR valueName, BSTR value)
```

- **Key** – the Registry Key for the value to set. The key value must be one of the following:

  HKEY_CLASSES_ROOT (0)
  HKEY_CURRENT_USER (1)
  HKEY_LOCAL_MACHINE (2)
  HKEY_USERS (3)

- **subKeyName** – the name of the sub-key for the value to set..
- **valueName** – the value name to set. If this value is an empty string, the default value will be set.
- **Value** – a string value containing the value to set.

Return Value:

The return value is TRUE for success and FALSE for failure.

Example:

```
Dim stat As Boolean

stat = SregoCETPx1.RegSetStringValue(HKEY_LOCAL_MACHINE, _
        "Software\Srego\test", "TestValue", "Test Value 123")

If Not stat Then

    MsgBox "Unable to set Reg Value"

End If
```

RegSetNumValue

This method sets a numeric value in the device's registry. If the value or key does not exist, it will be created.

Syntax:

```
boolean RegSetNumValue(long key, BSTR subKeyName,
                       BSTR valueName, long value)
```

- **Key** – the Registry Key for the value to set. The key value must be one of the following:

  HKEY_CLASSES_ROOT (0)
  HKEY_CURRENT_USER (1)
  HKEY_LOCAL_MACHINE (2)
  HKEY_USERS (3)

- **subKeyName** – the name of the sub-key for the value to set..
- **valueName** – the value name to set. If this value is an empty string, the default value will be set.
- **Value** – a long value containing the value to set.

Return Value:

The return value is TRUE for success and FALSE for failure.

RegSetBinaryValue

This method sets a value in the device's registry to a binary value. The binary value must be an array of BYTE values. If the value or key does not exist, it will be created.

Syntax:

```
boolean RegSetBinaryValue(long key, BSTR subKeyName,
                          BSTR valueName,
                          VARIANT* byteArray,
                          long size)
```

- **Key** – the Registry Key for the value to set. The key value must be one of the following:

HKEY_CLASSES_ROOT (0)
HKEY_CURRENT_USER (1)
HKEY_LOCAL_MACHINE (2)
HKEY_USERS (3)

- **subKeyName** – the name of the sub-key for the value to set.
- **valueName** – the value name to set. If this value is an empty string, the default value will be set.
- **Value** – a variant array containing BYTE values.
- **Size** – the number of bytes in the value array.

Example:

```
Dim stat As Boolean
Dim barray as Variant
ReDim barray(5) As Byte

barray(0) = 10
barray(1) = 20
barray(2) = 30
barray(3) = 40
barray(4) = 50

stat = SregoCETPx1.RegSetBinaryValue(HKEY_LOCAL_MACHINE, _
                    "Software\Srego\test", _
                    "BinaryValue", barray, 5)

If Not stat Then

      MsgBox "Unable to set Reg Value"

End If
```

To read a binary value from the registry:

```
Dim v As Variant
Dim i As Long

v = SregoCETPx1.RegGetValue(HKEY_LOCAL_MACHINE, _
          "Software\Srego\test", "BinaryValue")

If VarType(v) = vbArray Or vbByte Then

       For i = 0 To UBound(v) - 1

              List1.AddItem v(i)

       Next

End If
```

RegSetBinaryValueFromByteBuffer

This method sets a value in the device's registry to a binary value. The binary value for this method is set with the ByteBufferAddValue methods. If the value or key does not exist, it will be created.

Syntax:

```
boolean RegSetBinaryValueFromByteBuffer(long key,
                            BSTR subKeyName,
                            BSTR valueName)
```

- **Key** – the Registry Key for the value to set. The key value must be one of the following:

  HKEY_CLASSES_ROOT (0)
  HKEY_CURRENT_USER (1)
  HKEY_LOCAL_MACHINE (2)
  HKEY_USERS (3)

- **subKeyName** – the name of the sub-key for the value to set.
- **valueName** – the value name to set. If this value is an empty string, the default value will be set.

Example:

```
 Dim stat As Boolean

 SregoCETPx1.ByteBufferClear

 SregoCETPx1.ByteBufferAddValue 10
 SregoCETPx1.ByteBufferAddValue 20
 SregoCETPx1.ByteBufferAddValue 30
 SregoCETPx1.ByteBufferAddValue 40
 SregoCETPx1.ByteBufferAddValue 50

stat = SregoCETPx1.RegSetBinaryValueFromByteBuffer(2,
               "Software\Srego\test", "BinaryValue")

If Not stat Then

    MsgBox "Unable to set Reg Value"

End If
```

RegDeleteValue

This method will delete a value in the device's registry.

Syntax:

```
boolean RegDeleteValue(long key, BSTR subKeyName,
                       BSTR valueName)
```

- **Key** – the Registry Key for the value to delete from. The key value must be one of the following:

  HKEY_CLASSES_ROOT (0)
  HKEY_CURRENT_USER (1)
  HKEY_LOCAL_MACHINE (2)
  HKEY_USERS (3)

- **subKeyName** – the name of the sub-key for the value to delete from.
- **valueName** – the value name to delete. If this value is an empty string, the default value will be deleted.

Return Value:

The return value is TRUE for success and FALSE for failure.

RegDeleteSubKey

This method will delete a key from the device's registry. A key cannot be deleted if it has any values or subkeys.

Syntax:

```
boolean RegDeleteSubKey(long key, BSTR parentSubKeyName,
                        BSTR subKeyName)
```

- **Key** – the Registry Key for the value to delete from. The key value must be one of the following:

  HKEY_CLASSES_ROOT (0)
  HKEY_CURRENT_USER (1)
  HKEY_LOCAL_MACHINE (2)
  HKEY_USERS (3)

- **parentSubKeyName** – the name of the parent sub-key for the value to delete from.
- **subKeyName** – the name of the sub-key for the value to delete.

Return Value:

The return value is TRUE for success and FALSE for failure.

RegGetKeyList

This method will return a list of sub-key names for a specified key in the device's registry.

Syntax:

```
boolean RegGetKeyList(long key, BSTR subKeyName,
                      VARIANT* keyList,
                      VARIANT* count)
```

- **Key** – the Registry Key to search. The key value must be one of the following:

  HKEY_CLASSES_ROOT (0)
  HKEY_CURRENT_USER (1)
  HKEY_LOCAL_MACHINE (2)
  HKEY_USERS (3)

- **subKeyName** – the name of the subkey to search.
- **keyList** – a variant to return an array of subkey name.
- **count** – a variant that will return the count of subkeys in the keyList array.

Return Value:

The return value is TRUE for success and FALSE for failure.

Example:

```
Dim list As Variant
Dim count As Variant
Dim i As Long

If SregoCETPx1.RegGetKeyList(HKEY_LOCAL_MACHINE, _
        "Software\Microsoft", list, count) Then

     For i = 0 To count - 1
```

```
            List1.AddItem Str(i) + " - " + list(i)

        Next

    End If
```

RegGetValueNameList

This method returns a list of value names for a specific key on the device's registry.

Syntax:

```
boolean RegGetValueNameList(long key, BSTR subKeyName,
                            VARIANT* valueNameList,
                            VARIANT* count)
```

- **Key** – the Registry Key to search. The key value must be one of the following:

  HKEY_CLASSES_ROOT (0)
  HKEY_CURRENT_USER (1)
  HKEY_LOCAL_MACHINE (2)
  HKEY_USERS (3)

- **subKeyName** – the name of the subkey to search.
- **valueNameList** – a variant to return an array of value names.
- **Count** – a variant to return the count of names in the valueNameList.

Return Value:

The return value is TRUE for success and FALSE for failure.

Example:

```
Dim list As Variant
Dim count As Variant
Dim i As Long

If SregoCETPx1.RegGetValueNameList(HKEY_LOCAL_MACHINE, _
              "Software\Microsoft\clock", list, count) Then

    For i = 0 To count - 1

        List1.AddItem Str(i) + " - " + list(i)
```

```
        Next

    End If
```

Invoke Methods

InvokeFunction

This method will remotely execute a function in a dlls on the connected device. This method encapsulates the Block mode of the CeRapiInvoke function.

Syntax:

```
long InvokeFunction(BSTR dllName, BSTR functionName,
                    VARIANT input, long inputSize,
                    VARIANT* output,
                    VARIANT* outputSize)
```

- **dllName** – the name of the DLL on the device containing the function.
- **functionName** – the name of the function in the specified DLL to invoke.
- **input** – a Byte array containing the input data to pass to the function.
- **inputSize** – the number of items in the input byte array.
- **Output –** returns a variant Byte array of data returned by the invoked function.
- **ouputSize –** returns a variant long value containing the number of bytes in the output byte array.

Return Value:

The return value is one of the return value of the invoked dll function. A negative value is considered a failure.

The following example provides a VB call to invoke a dll on the device. The sample below also includes a sample function to create a long and a double from a byte array. The dll used for this example is provided below the VB code.

Example:

```
Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" _
(Destination As Any, Source As Any, ByVal Length As Long)


Private Sub Command1_Click()

    Dim inputBuffer(5) As Byte
```

```
    inputBuffer(0) = 15
    inputBuffer(1) = 25
    inputBuffer(2) = 35
    inputBuffer(3) = 45
    inputBuffer(4) = 55

    Dim inputSize As Long

    inputSize = 5

    Dim outputBuffer As Variant
    Dim outputSize As Variant

    Dim retVal As Long

    retVal = SregoCETPx1.InvokeFunction("invoketest.dll", "TestInvokeFunction", _
                            inputBuffer, inputSize, outputBuffer, outputSize)

    MsgBox "retVal = " + Str(retVal)  ' retVal = 350

    If retVal >= 0 Then  ' positive value means success

        Dim value As Long

        value = 0

        If outputSize >= 4 Then

            value = ConvertDataToLong(outputBuffer, pos)

            MsgBox "Ouput Value = " + Str(value)   ' Output Value = 175

        End If

    Else

        MsgBox "Unable to Invoke function."

    End If

End Sub

Function ConvertDataToDouble(data As Variant, pos) As Double

    Dim oldVal(8) As Byte
    Dim newVal As Double
    Dim i As Long

    For i = 0 To 7 Step 1

        oldVal(i) = data(i + pos)

    Next

    CopyMemory newVal, oldVal(0), 8
    ConvertDataToDouble = newVal

End Function

Function ConvertDataToLong(data As Variant, pos) As Long

    Dim oldVal(4) As Byte
    Dim newVal As Long
    Dim i As Long

    For i = 0 To 3

        oldVal(i) = data(i + pos)

    Next
```

```
    CopyMemory newVal, oldVal(0), 4
    ConvertDataToLong = newVal

End Function
```

The following shows the sample dll code that was called:

```
// invokeTest1.cpp : Defines the entry point for the DLL application.
//

#include "stdafx.h"
#include "invoketest.h"

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                                     )
        {
    switch (ul_reason_for_call)
                {
                case DLL_PROCESS_ATTACH:
                case DLL_THREAD_ATTACH:
                case DLL_THREAD_DETACH:
                case DLL_PROCESS_DETACH:
                        break;
                }
    return TRUE;
        }

extern "C"
        {
        __declspec(dllexport) int TestInvokeFunction(DWORD inputSize, BYTE* input,
                  DWORD *outputSize, BYTE **output, PVOID reserved);


        }

int TestInvokeFunction(DWORD inputSize, BYTE* input, DWORD *outputSize,
                          BYTE **output, PVOID reserved)
        {

        char *list=(char *)input;


        long sum=0;

        for (int i=0; i<(int)inputSize; i++)
                {
                sum+=list[i];
                }

        *output = (BYTE*)LocalAlloc(LPTR, sizeof(long));
        memcpy(*output, &sum, sizeof(sum));
        *outputSize = sizeof(sum);

        return sum*2;
        }
```

The TestInvokeFunction sums the byte array and stores the values as a long in the output array. A long is 4 bytes, so the output size will be 4. The return value of the function is 2 times the byte array sum. This example performs a meaningless task, but it illustrates how to read the input array and return something in the output array.

InvokeFunctionWithString

This method will remotely execute a function in a dlls on the connected device. This method encapsulates the Block mode of the CeRapiInvoke function. This method is very similar to InvokeFunction only instead of accepting a variant byte array for an input, it will accept a string which is automatically converted to a Unicode string. Since a string is a common input parameter and it is a chore to get the string appropriately converted, this method will provide an easier way to meet this specific need; however, the InvokeFunction method can perform the same task.

Syntax:

Syntax:

```
long InvokeFunction(BSTR dllName, BSTR functionName,
                    BSTR input,
                    VARIANT* output,
                    VARIANT* outputSize)
```

- **dllName** – the name of the DLL on the device containing the function.
- **functionName** – the name of the function in the specified DLL to invoke.
- **input** – a string containing an input data.
- **Output –** returns a variant Byte array of data returned by the invoked function.
- **ouputSize –** returns a variant long value containing the number of bytes in the output byte array.

Return Value:

The return value is one of the return value of the invoked dll function. A negative value is considered a failure.