Users Guide For

XLSConverterX

By SoftInterface, Inc.

Version 1.X
Windows 95/98/2000/XP

Contents

XLSConverterX Users Manual

What is XLSConverterX?

XLSConverterX is an ActiveX component designed to assist you, the developer, in quickly adding an Excel conversion/manipulation utility to your applications. XLSConverterX is designed to manipulate comma delimited CSV, and MS Excel (and any files MS Excel can open) files. A tool like this can be very helpful when you need to move, convert or manipulate Excel like data. HTML, Text, RTF and CSV are available for conversion as well as several database formats and older versions of Excel, beginning with version 2.0.

In addition to doing file conversions, XLSConverterX also adds enhanced features and some basic data massaging methods to further enhance its usefulness. If you need to include Excel manipulation capability in your program, XLSConverterX offers numerous specialized processes including:

- Copy specific sheet data from one worksheet to another within the same or a different workbook
- Concatenate/append specified data from a whole folder of workbooks to a single sheet
- Copy an entire worksheet (including formatting) to the same or a different workbook, and specify location within the target workbook
- Delete a single or whole range of sheets within a workbook
- Add a new sheet and specify where to place the new sheet within the workbook
- Move a sheet to a specific place within a workbook
- Rename an existing sheet

In addition to the worksheet manipulation, many other special processes for CSV and/or text files have been built into XLSConverterX. These include:

- (*.TXT) Append (Concatenate) files. Original file(s) content is placed at the end of the Target file.
- (*.TXT) Remove empty lines
- (*.CSV) Surround field with quotes
- (*.CSV) Pad field with spaces
- (*.CSV) Change comma to other delimiter
- (*.CSV) Remove Empty Lines
- (*.CSV) Include specified ROWS, discard all others
- (*.CSV) Include specified COLUMNS, discard all others

- (*.CSV) Remove control characters
- (*.CSV) Trim excess commas

XLSConverterX encapsulates all the details required for quick integration. Furthermore, an extensive sample Visual Basic application is provided to get you up to speed quickly and demonstrates XLSConverterX's capability and usability. Although you may not be using Visual Basic, the approach for all development environments will be similar.

Simply install the product, and add the component to your development environment. Once in place, the routines can be accessed to programmatically convert and manipulate Excel files. The code required to integrate this component in an application is relatively small considering the functionality it provides.

This Reference section of this document discusses the Properties, Methods, and Events that are exposed by the XLSConverterX component. Please check Distributing XLSConverterX for information concerning which files are required when distributing XLSConverterX.

Please Note: In those cases where an Excel file is used, this product automates MS Excel to convert/manipulate the files; therefore, MS Excel is required. The user is encouraged to make sure Excel is properly licensed on the PC.

At SoftInterface, Inc. we are constantly enhancing and improving our products. Please visit our web site to see what is new and tell us what you would like to see in our products (WWW.SoftInterface.COM). ***In addition, it is important to register your products to ensure you have the latest version and support.***

XLSConverterX Features

- Advanced Excel manipulation special processes include moving, deleting, adding and copying whole sheets or portions of sheets.
- Append/concatenate Excel files into a single sheet
- For Excel users, selectable sheets; syntax is simple (i.e. "1,2,4-10" or "*" for all)
- Numerous comma delimited file (CSV) manipulation methods
- Concatenate text files and remove empty lines
- Quick integration (sample source code provided)
- Supports MS Excel and CSV formats
- Crop/extract specific columns/rows from an Excel spreadsheet
- Can be adapted for most languages

What is an ActiveX control?

ActiveX controls are an extension of Microsoft's COM (Component Object Model) technology, providing unprecedented compatibility with almost any rapid application development environment. ActiveX controls, sometimes referred to as *reusable components*, give you, the programmer, the easiest way to incorporate advanced functionality into your applications with little or no programming on your part.

What You Will Need to Use XLSConverterX

The minimum hardware and software requirements to install and support the use of XLSConverterX are:

- Microsoft Excel on the target machine if you intend to use the Excel features. NOTE: You are responsible for purchasing and proper licensing of this product
- IBM or compatible PC/AT (Pentium or higher CPU) with 16 MB of memory and one hard disk drive with 3 MB of space
- VGA or SVGA display adapter
- Microsoft Windows 95, Windows 98, Windows 2000, or Windows XP
- Development environment supporting 32-bit OCX controls such as Microsoft's Visual Basic (4.x or greater)

Installing

When you run the setup program to install XLSConverterX on your computer, you will be able to specify where on your hard drive to install. It is preferable to install it in the suggested directory for consistency (although not required).

Run the Setup.EXE that came with the XLSConverterX media. You may do this by clicking the *Start* button from the taskbar and select the *Run...* menu option. Then type the path and location of the Setup.EXE program. For example:

```
A:setup
```

Then press ENTER. Thereafter, follow the installation instructions on the screen.

Uninstalling

It is highly suggested that you uninstall XLSConverterX before upgrading to a newer version of the product.

To uninstall XLSConverterX click the *Start* button from the taskbar and select *Settings* then *Control Panel*. Within the control panel, select the *Add/Remove Programs* icon. Double click on the XLSConverterX entry in the list box or push the *Add/Remove* button to uninstall.

All files copied during the installation will be removed (only if other programs are not currently dependent on them). Furthermore, if files have been added to the installation directory (i.e. program files you created) then the uninstall wizard will report that not all directories could be deleted. You will have to manually remove those files.

Distributing XLSConverterX

The necessary files needed to deploy your applications with XLSConverterX are discussed here. XLSConverterX is a self-registering ActiveX component.

The following table lists dependencies of XLSConverterX (**Items in bold were shipped with XLSConverterX.OCX**):

XLSCONVERTERX IMMEDIATE DEPENDENCIES	VERSION IN USE AT TIME OF SHIPPING
MSVBVM60.DLL	**6.0.89.64**
ADVAPI32.DLL	**5.0.2195.5992**
OLE32.DLL	**5.0.2195.6089**
OLEAUT32.DLL	**2.40.4518.0**
SCRRUN.DLL	**5.6.0.6626**
CSVSPECIALPROCESSING.DLL	**0.0.0.0**
KERNEL32.DLL	5.0.2195.6079
USER32.DLL	5.0.2195.6097
GDI32.DLL	5.0.2195.5907

Dependency implies that the file(s) must be installed before XLSConverterX.OCX is installed and registered.

Troubleshooting XLSConverterX - Updates

The SoftInterface, Inc. web site (www.SoftInterface.COM) will have the links required for all XLSConverterX:

- Frequently Asked Questions
- Bug Lists
- Latest Patches/Downloads

Please first review this manual, then the SoftInterface, Inc. Web site for assistance.

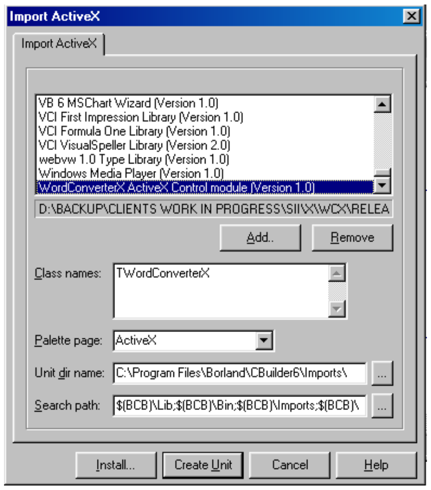
If you are still having trouble, you may e-mail for support at Support@SoftInterface.COM.

Using Borland C++ Builder

C++ Builder version 6 and up is suggested. You will want to import the XLSConverterX components before using Borland C++ Builder.

C++ Builder: Importing Components

After installation of this product, it can be made available from the 'ActiveX' tab of the C++ Builder controls palette by:



VB 6 MSChart Wizard (Version 1.0)
VCI First Impression Library (Version 1.0)
VCI Formula One Library (Version 1.0)
VCI VisualSpeller Library (Version 2.0)
webvw 1.0 Type Library (Version 1.0)
Windows Media Player (Version 1.0)
WordConverterX ActiveX Control module (Version 1.0)

D:\BACKUP\CLIENTS WORK IN PROGRESS\SII\X\WCX\RELEA

Class names: TWordConverterX

Palette page: ActiveX

Unit dir name: C:\Program Files\Borland\CBuilder6\Imports\

Search path: $(BCB)\Lib;$(BCB)\Bin;$(BCB)\Imports;$(BCB)\

i. Selecting the **"Component\Import ActiveX Control…"** menu item.

ii. Selecting **XLSConverterX ActiveX Control Module** from the list of available components.

iii. Clicking the **Install** button to create a package. The name and location of the package is of your choosing.

iv. After this has completed the component palette will be updated as a result of rebuilding the package.

C++ Builder: Passing Arguments

All arguments to the methods or events can be viewed by the wrapper class, TXLSConverterX, created by the importing mechanism.

Passing string arguments that do not get modified by the calling method can be done by using the WideString data type as illustrated below:

```
  WideString wsSourceFile,wsTargetFile;
  wsSourceFile = txtWordSourceFile->Text;
  wsTargetFile = txtWordTargetFile->Text;
  bResult = WCX->ConvertWordDoc(wsSourceFile, wsTargetFile,
cWordType[cbWordConversionType->ItemIndex]);
```

Although none exist in TXLSConverterX, passing strings arguments that *get* modified by the calling method can done by using the WideString data type as illustrated below:

```
/*
VARIANT_BOOL __fastcall TCompareFilesX::DirCompareResultsGet(long
ResultIndex, BSTR* FileName, BSTR* DirMaster, BSTR* DirSource,long*
IsFile, long* OnlyInSource,long* OnlyInMaster, long*
SizeDifferent,long* DateDifferent,long* ContentDifferent)
*/
    WideString ws_file_name,ws_dir_mstr,ws_dir_src;
    if (CompareFilesX1->DirCompareResultsGet(ii, &ws_file_name,
&ws_dir_mstr, &ws_dir_src, &bIsFile, &bOnlyInSource, &bOnlyInMaster,
&bSizeDifferent, &bDateDifferent, &bContentDifferent) == FALSE)
      break;// Exit for loop

    AnsiString as_file_name = AnsiString(ws_file_name);
    AnsiString as_dir_mstr = AnsiString(ws_dir_mstr);
AnsiString as_dir_src = AnsiString(ws_dir_src);
```

Using CreateObject() To Create an Instance

Often, the method for using a control requires you to create an object reference to the control. Assuming that you have registered the control on a computer the typical syntax, such as ADO would use, would be:

```
Set oObject = Server.CreateObject("ObjectName.ClassName")
```

In the case of XLSConverterX, please use “XLSConverterX” as the Object name and XLSConverterXCtrl as the Class name. For example:

Set oObject = Server.CreateObject("**XLSConverterX.XLSConverterXCtrl**")

XLSConverterX Reference

Properties

ErrorString

Data Type: String

Default Value: “ ”

Description: If any of the methods return an error value, this property will contain a descriptive sentence describing what is wrong.

ExcelVersion

Data Type: Double

Default Value: “ ”

Description: Version of the installed Excel application.

FilesJustCreatedCount

Data Type: Long

Default Value: 0

Description: When using OpenXLSSaveAs(), this propery gets updated. The number of files created is available through this property. See also the FilesJustCreated() method.

InputFile_PassWord

Data Type: String

Default Value: “ ”

Description: If you Input file requires a password to open, use this property to specify it before calling any methods.

InputFile_PassWord_TW

Data Type: String

Default Value: “ ”

Description: If you Input file requires a password to write, use this property to specify it before calling any methods.

Key

Data Type: Long Integer

Default Value: 0

Description: Set this parameter to the registration/serial number you receive upon purchase of XLSConverterX to change the product from shareware to release mode.

OptimizeForSpeed

Data Type: Boolean

Default Value: False

Description: When using Excel file manipulation methods (i.e. CopySheetData(), AddSheet(), MoveSheet() etc.), XLSConverterX must open MS Excel. Calling such methods can cause XLSConverterX to create and destroy an Excel application EACH time it is called. To only create and destroy MS Excel once for numerous calls to such methods, set this property to TRUE.

Note: There is a trade off between memory conservation and speed that must be understood. When setting this property to TRUE, your application has the potential to be much faster, as in those cases where many Excel methods are called. However, Excel has a tendency to eat more and more memory if it is not destroyed periodically. Therefore, setting this property to FALSE optimizes for memory conservation.

OutputFile_PassWord

Data Type: String

Default Value: “ ”

Description: If you Output file requires a password to open, use this property to specify it before calling any methods.

OutputFile_PassWord_TW

Data Type: String

Default Value: “ ”

Description: If you Input file requires a password to write, use this property to specify it before calling any methods.

SheetsJustProcessedCount

Data Type: Long

Default Value: 0

Description: When using OpenXLSSaveAs(), this propery gets updated. The number of sheets processed is available through this property. See also the SheetsJustProcessed() method.

SkipEmptySheets

Data Type: Boolean

Default Value: False

Description: If set to FALSE XLSConverterX will ignore any blank sheets that are found in the subject workbook.

StateFile

Data Type: String

Default Value: WindowsFolder\XLSCX.INI

Description: Path and filename where state variables for this component will be saved. All the properties associated with this component are saved to an INI file. You can specify which INI file to use.

Why is this property needed? Because if you have multiple instances of this component, either in the same or different application running on the same PC, you may want different property settings for each instance.

See Also: UseStateFile, StateFileSave(), StateFileLoad()

UseStateFile

Data Type: Boolean

Default Value: True

Description: When TRUE the StateFile is loaded and utilized; when FALSE the default properties are used.

See Also: StateFile, StateFileSave(), StateFileLoad()

Verbose

Data Type: Boolean

Default Value: False

Description: If set to FALSE XLSConverterX will suppress warnings and prompts that Excel might generate.

Methods: Excel File Manipulation

In nearly all the file manipulation methods there will be an original (*sPathOriginal*) and target (*sPathTarget*) file specified as a parameter. Therefore, it should be understood that the original file does not have to be modified, but can be if you desire it to be. To modify the original file, specify it as both the original and target. To create a new file based on the original file, while leaving the original file unchanged, specify a target file that is different from the original file.

AddSheet

Description: Add a new blank worksheet to a specified location within a workbook.

Parameters:

AddSheet(*sPathOriginal* As String, *sPathTarget* As String, *sSheetToAdd* As String, *sSheetBefore* As String, *sSheetAfter* As String, *bOverwrite* As Boolean) As Long

Parameter	**Meaning**
sPathOriginal	Excel file to use as the original file.
sPathTarget	Can be same as sPathOriginal, otherwise a new workbook is created with both the original and newly added sheet(s).
sSheetToAdd	Name of the sheet to add. Not required, can use "" to get default Excel name.
sSheetBefore	Added sheet will be placed before this sheet (Name or #). If sSheetBefore = "firstfirst" then it is placed as the first sheet in the workbook.
sSheetAfter	Added sheet will be placed after this sheet (Name or #). If sSheetAfter = "lastlast" then it is placed as the last sheet.
bOverwrite	If *sSheetToAdd* already exists, it will be overwritten if this parameter is set to TRUE. Otherwise error -212 is returned.

Notes: If both *sSheetBefore* and *sSheetAfter* are empty strings then the new sheet will be the last sheet.

Return Values:

0: Success

-3: Unable to create Excel Application. Is it installed?

-4: Unable to destroy Excel Application.

-200: Excel reported an error

-201: Shareware Expired

-204: Source Sheet does not exist

-208: Target Path folder does not exist, even after attempting to create it.

-210: SheetBefore or SheetAfter does not exist

-212: "Target Sheet already exists. Set overwrite to TRUE to copy over existing sheet."

Example Code:

Taken from the sample VB program provided, when AddSheet() is called, a data collection form is displayed, the data is collected then passed to the component for processing.

```
Case SP_XLS_ADD_SHEET
    frmXLSAddSheet.Show vbModal   'display form
    If (frmXLSAddSheet.bErrorOccurred = True) Then
        Exit Sub
    Else   'collect necessary data
        strSheetToAdd = frmXLSAddSheet.sTargetSheet
        strSheetBefore = frmXLSAddSheet.sSheetBefore
        strSheetAfter = frmXLSAddSheet.sSheetAfter
        blnOverwrite = frmXLSAddSheet.bOverwrite
    End If
           'call component with necessary arguments
    lngResult = XLSConv1.AddSheet(strSourceFile, strTargetFile,
                strSheetToAdd, strSheetBefore, strSheetAfter,
                blnOverwrite)
```

ChangeNumberFormat

Description: Changes the number format of selected cells.

Parameters:

ChangeNumberFormat(sPathOriginal As String, PathTarget As String, sSheet As String, sRangeToChange As String, sNewFormat As String)

Parameter	Meaning
sPathOriginal	Excel file from which to change the number format
sPathTarget	Can be same as *sPathOriginal*, otherwise a new workbook is created.
sSheet	Sheet to change number formatting
sRangeToChange	Range of cells that will have new formatting applied to
sNewFormat	The new number format that will be applied to sRangeToChange

Return Values:

0: Success

-3: Unable to create Excel Application. Is it installed?

-4: Unable to destroy Excel Application.

-200: Excel reported an error.

-201: Shareware Expired.

-208: Target path folder does not exist, even after attempting to create it.

-213: Invalid or missing sheet specified.

ChangePassword

Description: Changes the password of an Excel workbook.

Parameters:
ChangePassword(*sPathOriginal* As String, *sPathTarget* As String,
sOriginalFilePW As String, *sOriginalFilePWToWrite* As String,
sTargetFilePW As String, *sTargetFilePWToWrite* As String) As Long

Parameter	Meaning
sPathOriginal	Excel file from which to copy the sheet.
sPathTarget	Excel file to which data is copied.
sOriginalFilePW	Original file password to open XLS file
sOriginalFilePWToWrite	Original file password to write within the XLS file
sTargetFilePW	Target file password to open the XLS file
sTargetFilePWToWrite	Target file password to write within the XLS file

Notes: *sPathOriginal* AND *sPathTarget* can be the same Excel file, in which case the passwords of the file are changed or removed.

Return Values:

0: Success

-3: Unable to create Excel Application. Is it installed?

-4: Unable to destroy Excel Application.

-200: Excel reported an error

-201: Shareware Expired

-208: Target Path folder does not exist, even after attempting to create it.

Example Code:

This simple example opens “C:\input\A.XLS” and changes its password to open from “Apples” to “Oranges”:

```
lErr = XLSCX.ChangePassword("C:\input\A.XLS", "C:\input\A.XLS",
                            "Apples", "", "Oranges", "")
```

CopySheet

Description: Copies an entire worksheet from one Excel file to another, or the same Excel file. This includes values, formulas, formatting etc.

Parameters:

CopySheet(*sPathOriginal* As String, *sSheetOriginal* As String, *sPathTarget* As String, *sSheetTarget* As String, *sSheetBefore* As String, *sSheetAfter* As String, *lActionIfAlreadyExists* As Long) As Long

Parameter	Meaning
sPathOriginal	Excel file from which to copy the sheet.
sSheetOriginal	Sheet Name or number to copy.
sPathTarget	Excel file to which data is copied.
sSheetTarget	Renames the copied sheet to string specified in *sSheetTarget*. This is only available when a single sheet is being copied.
sSheetBefore	Copied sheet will be placed before this sheet (Name or #). If sSheetBefore = "firstfirst" then it is placed as the first sheet in the workbook.
sSheetAfter	Copied sheet will be placed after this sheet (Name or #). If sSheetAfter = "lastlast" then it is placed as the last sheet.
lActionIfAlreadyExists	If sheet name being copied already exists in the target workbook, 1 of 4 actions will be done: 0 = Copy sheet and give similar name as original sheet. 1 = Overwrite duplicates 2 = Skip duplicates 3 = Stop Copying and report an error

Notes: *sPathOriginal* AND *sPathTarget* can be the same Excel file, in which case the positioning information (*sSheetBefore*, *sSheetAfter*) is used. Otherwise, a new workbook file is created, and saved as *sPathTarget*. If *sPathOriginal* and *sPathTarget* are different, all positioning information is ignored, since it will be the only sheet in the new workbook. If both *sSheetBefore* and *sSheetAfter* are specified, *sSheetAfter* is used.. When a single sheet is being copied, a new name may be specified in *sSheetTarget* for the new sheet. This is not available when copying multiple sheets.

Return Values:

0: Success

-3: Unable to create Excel Application. Is it installed?

-4: Unable to destroy Excel Application.

-200: Excel reported an error

-201: Shareware Expired

-202: Source File does not exist

-203: Target File does not exist

-204: Source Sheet does not exist

-208: Target Path folder does not exist, even after attempting to create it.

-210: SheetBefore or SheetAfter does not exist

-211: "SheetBefore AND SheetAfter have not been specified. Use different target file to create a new workbook with worksheet copy."

-212: "Target Sheet already exists. Set overwrite to TRUE to copy over existing sheet."

Example Code:

Taken from the sample VB program, the component is passed the necessary arguments to copy a worksheet.

```
frmXLSCopySheet.Setup     'calls an ancillary form to
                          'collect data
If (frmXLSCopySheet.bErrorOccurred = True) Then
    Exit Sub
End If
    'assign data from form to variables
strSheetOriginal = frmXLSCopySheet.sOriginalSheet
strSheetTarget = frmXLSCopySheet.sTargetSheet
strSheetBefore = frmXLSCopySheet.sSheetBefore
strSheetAfter = frmXLSCopySheet.sSheetAfter
    'call the component with the necessary arguments
lngResult = XLSConv1.CopySheet(strSourceFile,
            strSheetOriginal, strTargetFile,
            strSheetTarget, strSheetBefore, strSheetAfter,
            lOverwrite)
Unload frmXLSCopySheet
```

CopySheetData

Description: Copies data (values or formulas) you specify from a worksheet to the same/different worksheet. The data can be also be copied to the same or different Excel workbook file.

Parameters:

CopySheetData(*sPathOriginal* As String, *sSheetOriginal* As String, *sRangeToCopyFrom* As String, *sPathTarget* As String, *sSheetTarget* As String, *sRangeToCopyTo* As String, *bDoFormula* As Boolean, *bCopyNameOfSheet* As Boolean, *bAddSheetsIfNecessary* As Boolean) As Long

Parameter	Meaning
sPathOriginal	Excel file which data is copied FROM.
sSheetOriginal	Sheet Name or number FROM which data is copied.
sRangeToCopyFrom	Range of cells FROM which to copy data. If empty, it copies the range of used cells.

sPathTarget	Excel file TO which data is copied. If folder does not exist, XLSConverterX will attempt to create it. NOTE: *sPathOriginal* and *sPathTarget* can be the same or different Excel file.
sSheetTarget	Sheet Name or number TO which data is copied.
sRangeToCopyTo	Range of cells from which to copy data TO. Typical Excel syntax is acceptable. For example "A1:B10", "J5:K20", etc. If this parameter is empty, it will use the range specified by: 1) sRangeToCopyFrom only if sRangeToCopyFrom is not empty otherwise 2) it will use the same used range of cells as found in sSheetOriginal. If this range does not have the same dimensions as the sRangeToCopyFrom data may be lost in the copy operation. If set to "Below", the copied data is placed below the currently used range. If set to "Right", the copied data is placed to the right of the currently used range. HINT: "Below" and "Right" can be used to concatenate/append multiple sheets of data into one. Specifying a single cell (i.e. "B10") will cause the data to be placed beginning at that cell. If you specify a smaller range than the copied range, the data will be truncated. Specifying a larger range than the copied range causes invalid data to be copied to the target sheet.
bDoFormula	If TRUE, CopySheetData() will copy the formula vs. value of each cell.
bCopyNameOfSheet	Typically only used if the sSheetOriginal specifies a sheet number (vs. Name). The target sheet name will be the same as the original.
bAddSheetsIfNecessary	If sheets must be added to accommodate the sSheetTarget request, they will be if this parameter is set to TRUE. Otherwise, this function returns –205.

Notes**:** *sPathOriginal* combined with *sSheetOriginal* and *sRangeToCopyFrom* are used to specify **what** data to copy. *sPathTarget* combined with *sSheetTarget* and *sRangeToCopyTo* specify the **where** to copy to.

Return Values:

0: Success

-3: Unable to create Excel Application. Is it installed?

-4: Unable to destroy Excel Application.

-200: Excel reported an error

-201: Shareware Expired

-202: Original File does not exist

-203: Target File does not exist

-204: Original Sheet does not exist

-205: Target Sheet does not exist, be sure the 'Add sheets if necessary option' is set to True.

-206: Target Workbook cannot have two similarly named sheets. Original sheet name already exists in Target.

-207: Target Path file cannot have a wild card (i.e. *.XLS)

-208: Target Path folder does not exist, even after attempting to create it.

-200: Excel reported an error

Example Code:

Taken from the sample VB program, this code snippet demonstrates a call to the component to copy specific data from a spreadsheet after collecting the necessary data and passing it to the component.

```
Case SP_XLS_COPY_SHEET_DATA
    frmCopySheetData.Setup 'calls an ancillary form to collect data
    If (frmCopySheetData.bErrorOccurred = True) Then
        Exit Sub
    End If
            'assign data from the form to variables
    strSheetOriginal = frmCopySheetData.sOriginalSheet
    strRangeToCopyFrom = frmCopySheetData.sSpecifiedRangeToCopyFrom
    strSheetTarget = frmCopySheetData.sTargetSheet
    strRangeToCopyTo = frmCopySheetData.sSpecifiedRangeToCopyFromTo
    blnDoFormula = frmCopySheetData.bCopyFormula
    blnCopyNameOfSheet = frmCopySheetData.bCopySheetName
    blnAddSheetsIfNecessary = True
            'call the component with necessary arguments
    lngResult = XLSConv1.CopySheetData(strSourceFile,
                strSheetOriginal, strRangeToCopyFrom,
                strTargetFile, strSheetTarget, strRangeToCopyTo,
                blnDoFormula, blnCopyNameOfSheet,
                blnAddSheetsIfNecessary)
    Unload frmCopySheetData       'the ancillary form is unloaded
```

CopySheetDataEx

Description: This function does exactly what CopySheetData() does, however, you can now specify ranges of sheets to copy from and to.

Parameters:

CopySheetData(*sPathOriginal* As String, *sSheetOriginal* As String, *sRangeToCopyFrom* As String, *sPathTarget* As String, *sSheetTarget* As String, *sRangeToCopyTo* As String, *bDoFormula* As Boolean, *bCopyNameOfSheet* As Boolean, *bAddSheetsIfNecessary* As Boolean) As Long

Parameter	Meaning
sPathOriginal	Excel file which data is copied FROM.
sSheetOriginal	Sheet Names or numbers FROM which data is copied. You may specify ranges i.e."2-4,10" or "*" for all OR "2-4,10" or "Sheet1,Sheet4"
sRangeToCopyFrom	Range of cells FROM which to copy data. If empty, it copies the range of used cells.
sPathTarget	Excel file TO which data is copied. If folder does not exist, XLSConverterX will attempt to create it. NOTE: *sPathOriginal* and *sPathTarget* can be the same or different Excel file.
sSheetTarget	Sheet Names or numbers TO which data is copied. You may specify ranges i.e."2-4,10" or "*" for all OR "2-4,10" or "Sheet1,Sheet4"
sRangeToCopyTo	Range of cells from which to copy data TO. Typical Excel syntax is acceptable. For example "A1:B10", "J5:K20", etc.

	If this parameter is empty, it will use the range specified by: 1) sRangeToCopyFrom only if sRangeToCopyFrom is not empty otherwise 2) it will use the same used range of cells as found in sSheetOriginal. If this range does not have the same dimensions as the sRangeToCopyFrom data may be lost in the copy operation. If set to "Below", the copied data is placed below the currently used range. If set to "Right", the copied data is placed to the right of the currently used range. HINT: "Below" and "Right" can be used to concatenate/append multiple sheets of data into one. Specifying a single cell (i.e. "B10") will cause the data to be placed beginning at that cell. If you specify a smaller range than the copied range, the data will be truncated. Specifying a larger range than the copied range causes invalid data to be copied to the target sheet.
bDoFormula	If TRUE, CopySheetData() will copy the formula vs. value of each cell.
bCopyNameOfSheet	Typically only used if the sSheetOriginal specifies a sheet number (vs. Name). The target sheet name will be the same as the original.
bAddSheetsIfNecessary	If sheets must be added to accommodate the sSheetTarget request, they will be if this parameter is set to TRUE. Otherwise, this function returns –205.

Notes**:** *sPathOriginal* combined with *sSheetOriginal* and *sRangeToCopyFrom* are used to specify **what** data to copy. *sPathTarget* combined with *sSheetTarget* and *sRangeToCopyTo* specify the **where** to copy to.

Return Values:

0: Success

-3: Unable to create Excel Application. Is it installed?

-4: Unable to destroy Excel Application.

-200: Excel reported an error

-201: Shareware Expired

-202: Original File does not exist

-203: Target File does not exist

-204: Original Sheet does not exist

-205: Target Sheet does not exist, be sure the 'Add sheets if necessary option' is set to True.

-206: Target Workbook cannot have two similarly named sheets. Original sheet name already exists in Target.

-207: Target Path file cannot have a wild card (i.e. *.XLS)

-208: Target Path folder does not exist, even after attempting to create it.

-200: Excel reported an error

Example Code:

Taken from the sample VB program, this code snippet demonstrates a call to the component to copy specific data from a spreadsheet after collecting the necessary data and passing it to the component.

```
Case SP_XLS_COPY_SHEET_DATA
    frmCopySheetData.Setup 'calls an ancillary form to collect data
    If (frmCopySheetData.bErrorOccurred = True) Then
        Exit Sub
     End If
            'assign data from the form to variables
     strSheetOriginal = frmCopySheetData.sOriginalSheet
     strRangeToCopyFrom = frmCopySheetData.sSpecifiedRangeToCopyFrom
     strSheetTarget = frmCopySheetData.sTargetSheet
     strRangeToCopyTo = frmCopySheetData.sSpecifiedRangeToCopyFromTo
     blnDoFormula = frmCopySheetData.bCopyFormula
     blnCopyNameOfSheet = frmCopySheetData.bCopySheetName
     blnAddSheetsIfNecessary = True
            'call the component with necessary arguments
     lngResult = XLSConv1.CopySheetDataEx(strSourceFile,
                 strSheetOriginal, strRangeToCopyFrom,
                 strTargetFile, strSheetTarget, strRangeToCopyTo,
                 blnDoFormula, blnCopyNameOfSheet,
                 blnAddSheetsIfNecessary)
     Unload frmCopySheetData       'the ancillary form is unloaded
```

DeleteSheet

Description: Delete a sheet or range of sheets within a workbook. Due to Excel limitations, at least one sheet must remain in a workbook at all times.

Parameters:

DeleteSheet(*sPathOriginal* As String, *sPathTarget* As String, *sSheetStart* As String, *sSheetEnd* As String, *sSheetExcept* As String) As Long

Parameter	Meaning
SPathOriginal	Excel file from which to delete sheet(s).
SSheetOriginal	Sheet Name or number to delete.
SPathTarget	Can be same as sPathOriginal, otherwise a new workbook is created containing only the remaining sheets, leaving sPathOriginal unchanged.
SSheetStart	Used to either specify the deletion of a single sheet, or the beginning of a range of sheets to delete. Note: the range is inclusive, meaning sSheetStart is deleted.
SSheetEnd	Used to specify the end of a range of sheets to delete. Note: the range is inclusive, meaning sSheetEnd is deleted.
SSheetExcept	Used to specify the only sheet to remain in the workbook. All other sheets will be deleted. NOTE: If sSheetExcept is not specified, sSheetStart must be. If not -213 is returned.

Return Values:

0: Success

-3: Unable to create Excel Application. Is it installed?

-4: Unable to destroy Excel Application.

-200: Excel reported an error

-201: Shareware Expired

-208: Target Path folder does not exist, even after attempting to create it.

-213: Invalid or missing sheets specified

Example Code:

Taken from the sample VB program, this code snippet demonstrates the collection of data, which is then passed to the component when DeleteSheet() is called.

```
Case SP_XLS_DELETE_SHEET
    frmXLSDeleteSheet.Show vbModal   'display data collection form
    If (frmXLSDeleteSheet.bErrorOccurred = True) Then
        Exit Sub                        'check for error
    Else           'collect the necessary data
        strSheetStart = frmXLSDeleteSheet.sSheetStart
        strSheetEnd = frmXLSDeleteSheet.sSheetEnd
        strSheetExcept = frmXLSDeleteSheet.sSheetExcept
    End If
           'call the component with necessary arguments
    lngResult = XLSConv1.DeleteSheet(strSourceFile, strTargetFile,
                strSheetStart, strSheetEnd, strSheetExcept)
```

DeleteRowsOrColumns

Description: This method allows for the deletion of specified rows or columns from a specified worksheet.

Parameters:

DeleteRowsOrColumns(*sPathOriginal* As String, *sPathTarget* As String, *sSheet* As String, *sRowsColsToDelete* As String, *bDoColumns* As Boolean) As Long

Parameter	Meaning
SPathOriginal	Excel file from which to delete sheet(s).
SPathTarget	Can be same as sPathOriginal, otherwise a new workbook is created containing only the remaining sheets, leaving sPathOriginal unchanged.
sSheet	Worksheet (name or number) to remove columns/rows from.
sRowsColsToDelete	Rows or columns to delete, specified by numbers, or ranges seperated by comma's. For example "1,4,9-12" would specify to remove the columns/rows 1, 4, 9, 10, 11, and 12.
bDoColumns	Set to TRUE to delete columns, otherwise, rows will be deleted.

Return Values:

0: Success

-3: Unable to create Excel Application. Is it installed?

-4: Unable to destroy Excel Application.

-200: Excel reported an error

-201: Shareware Expired

-208: Target Path folder does not exist, even after attempting to create it.

-213: Invalid or missing sheets specified

FilesJustCreated

Description: Gets the fully qualified paths and names of the files created with OpenXLSSaveAs() method.

Parameters:

FilesJustCreated(lIndex As Long) As String

Parameter	Meaning
lIndex	Valid number from 1 to FilesJustCreatedCount property.

Return Values:

A fully qualified path and name of the file created, as specified by lIndex.

GetSheetNames

Description: Gets the names of the sheets specified.

Parameters:

GetSheetNames(spath As String, sSheets As String, sSheetNames() As String)

Parameter	Meaning
sPath	Original Excel file
sSheets	Selected sheets. May be specified as "*", or "1", or "1-3,10,12".
sSheetNames	Array storing selected sheets.

Return Values:

>= 0: Number of sheets found

-3: Cannot create Excel object

-11: Invalid sheet specified (empty string)

-202: Original file does not exist

IsFileUnicode

Description: Deterimines whether or not a file is UNICODE or 8 bit ASCII.

Parameters:

IsFileUnicode(sFileToTest As String) As Boolean

Parameter	Meaning
sFileToTest	Original text file

Return Values:

True: File is a Unicode file

False: File is not a Unicode file

MoveSheet

Description: Move a sheet to a new location within a workbook.

Parameters:

MoveSheet(*sPathOriginal* As String, *sPathTarget* As String, *sSheetToMove* As String, *sSheetBefore* As String, *sSheetAfter* As String) As Long

Parameter	Meaning

sPathOriginal	Excel file within which the sheet will be moved.
sPathTarget	Can be same as *sPathOriginal*, otherwise a new workbook is created.
sSheetToMove	Name of the sheet to move. Can be a name or #.
sSheetBefore	Sheet will be placed before this sheet (name or #). If sSheetBefore = "firstfirst" then it is placed as the first sheet in the workbook
sSheetAfter	Sheet will be placed after this sheet (Name or #). If sSheetAfter = "lastlast" then it is placed as the last sheet. NOTE: If both sSheetBefore and sSheetAfter are empty, then the sheet will be moved to the last position.

Return Values:

0: Success

-3: Unable to create Excel Application. Is it installed?

-4: Unable to destroy Excel Application.

-200: Excel reported an error

-201: Shareware Expired

-204: Source Sheet does not exist

-208: Target Path folder does not exist, even after attempting to create it.

-210: SheetBefore or SheetAfter does not exist

-215: Sheet to move does not exist

Example Code:

Taken from the sample VB program provided, a form is displayed to collect necessary data, which is then passed to the component for processing by the MoveSheet() function of the component.

```
Case SP_XLS_MOVE_SHEET
    frmXLSMoveSheet.Show vbModal      'display the form
        If (frmXLSMoveSheet.bErrorOccurred = True) Then
            Exit Sub   'check for error
        Else   'collect necessary data
            strSheetToMove = frmXLSMoveSheet.sSheetToMove
            strSheetBefore = frmXLSMoveSheet.sSheetBefore
            strSheetAfter = frmXLSMoveSheet.sSheetAfter
        End If
               'call the component with necessary arguments
        lngResult = XLSConv1.MoveSheet(strSourceFile,
                        strTargetFile, strSheetToMove,
                        strSheetBefore, strSheetAfter)
    Unload frmXLSMoveSheet
```

OpenXLSSaveAs

Description: Performs the conversion of an Excel, CSV or any file Excel can open into any of the file formats Excel can Save As.

Parameters:

OpenXLSSaveAs(*sSourceFile* As String, *sSheets* As String, *sTargetFile* As String, *lTargetType* As Long, *bDoFormula* As Boolean) As Long

Parameter	Meaning

sSourceFile	The file to be converted. It can be any file Excel can open. When installing MS Office, be sure to install all the Text and file filters available.
sTargetFile	Name for file resulting from the conversion process.
sSheets	Name or index number of sheet(s) being converted. You can specify single or multiple sheets, by name or index number.
lTargetType	The file format to convert TO, i.e. the format sTargetFile will be saved as. See Excel File Conversion Formats for valid values.
bDoFormula	If TRUE, OpenXLSSaveAs() will convert the formula rather than the value of each cell.

Return Values:

0 = Success

-2 = Source file does not exist

-3 = Unable to create Excel Application. Is it installed?

-4 = Unable to destroy Excel Application.

-10 = General Excel Error, see ErrorString for details

-11 = Invalid Sheet Specified

-201 = Shareware expired

Example Code:

Taken from the sample VB program provided, the necessary arguments are passed to the function OpenXLSSaveAs() in this case.

```
Private Function PerformConversionXLS(strSourceFile As String, _
                                      strSheets As String, _
                                      strTargetFile As String, _
                                      lngTargetType As Long, _
                                      blnDoFormula As Boolean) As Long

    Dim lngConvResult As Long

        'call the component with necessary arguments
    lngConvResult = XLSConv1.OpenXLSSaveAs(strSourceFile, strSheets,
                    strTargetFile, lngTargetType, blnDoFormula)

        'check that a file was created
    If (lngConvResult = 0) Then
        If (FileExists(strTargetFile) = False) Then
            lngConvResult = -100
        End If
    End If

        'assign an error code, if applicable
    Select Case (lngConvResult)
        Case -2
            strErr = "Source file does not exist."
        Case -3
            strErr = "Unable to create Excel Application. Is it
                      installed?"
        Case -4
            strErr = "Unable to destroy Excel Application."
        Case -10
            strErr = "General Excel error."
        Case -100
            strErr = "File does not exist."
    End Select

        'function value returned to caller for additional evaluation
    PerformConversionXLS = lngConvResult
End Function
```

RenameSheet

Description: Rename a worksheet.

Parameters:

RenameSheet(*sPathOriginal* As String, *sPathTarget* As String, *sSheetToRename* As String, *sNewSheetName* As String) As Long

Parameter	Meaning
sPathOriginal	Excel file containing the sheet to be renamed.
sPathTarget	Can be same as *sPathOriginal*, otherwise a new workbook is created with the original and newly renamed sheets.
sSheetToRename	Name of the sheet to rename.
sNewSheetName	sSheetToRename will be renamed to this value.

Return Values:

0: Success

-3: Unable to create Excel Application. Is it installed?

-4: Unable to destroy Excel Application.

-200: Excel reported an error

-201: Shareware Expired

-208: Target Path folder does not exist, even after attempting to create it.

-216: Sheet to rename does not exist

Example Code:

Taken from the sample VB program provided, this code demonstrates how a secondary form is displayed for data collection. The information is transferred to variables then passed to the RenameSheet function of the component.

```
Case SP_XLS_RENAME_SHEET
    frmXLSRenameSheet.Show vbModal       'form to collect data
     If (frmXLSRenameSheet.bErrorOccurred = True) Then
        Exit Sub
    Else   'collect the necessary data
        strSheetToRename = frmXLSRenameSheet.sSheetToRename
        strNewSheetName = frmXLSRenameSheet.sNewSheetName
    End If
           'call the component with necessary arguments
    lngResult = XLSConv1.RenameSheet(strSourceFile, strTargetFile,
                strSheetToRename, strNewSheetName)
```

SearchAndReplaceCellContent

Description: Search for and replace content of cell(s).

Parameters:

SearchAndReplaceCellContent(sPathOriginal As String, sPathTarget As String, sSheet As String, sSearchFor As String, sReplaceWith As String, bCaseSensitive As Boolean, bLookAtPart As Boolean, bMatchByte As Boolean) As Long

Parameter	Meaning

sPathOriginal	Original Excel file
sPathTarget	Can be same as *sPathOriginal*, otherwise a new workbook is created.
sSheet	Name of the sheet to be searched
sSearchFor	Text to search for
sReplaceWith	Text to replace with
bCaseSensitive	TRUE means a case sensitive search
bLookAtPart	If TRUE, replacement will occur only if the sSearchFor comprises the entire cell content.
bMatchByte	May use ASCII or Unicode character sets. If bMatchByte is TRUE then replacement occurs only if both search and replace characters are in the same character set.

Return Values:

0: Success

-3: Unable to create Excel Application. Is it installed?

-4: Unable to destroy Excel Application.

-200: Excel reported an error.

-201: Shareware Expired.

-208: Target Path folder does not exist, even after attempting to create it.

-213: Invalid or missing sheet specified.

Example Code:

SheetsJustProcessed

Description: Gets the names of the sheets processed with OpenXLSSaveAs() method.

Parameters:

SheetsJustProcessed (lIndex As Long) As String

Parameter	Meaning
lIndex	Valid number from 1 to SheetsJustProcessedCount property.

Return Values: Sheet names.

ShowSheetSelectionDialog

Description: A built in form has been provided to allow your users to select which sheet of a given workbook they wish to perform an action on. This form allows your end users to click on a listbox containing sheet names rather than having them specify it by name or index number.

Parameters:

ShowSheetSelectionDialog(*sXLSFile* As String, *bMultipleSelect* As Boolean) As String

Parameter	Meaning
sXLSFile	The source workbook file
bMultipleSelect	Users can select multiple sheets if you allow them to by setting this parameter to TRUE. Values are returned as a comma-delimited list (i.e. "Sheet2, LastSheet").

Example Code:

Taken from the sample VB program from the frmCopySheetData form

```
Private Sub cmdSelectSheetOriginal_Click()
    Dim sSheets As String
    bErrorOccurred = False

        'check if the file exists first
    If Not
       frmTestXLSConverterX.FileExists(frmTestXLSConverterX _
       strSourceFile) Then
       MsgBox "Please specify a file that exists", vbOKOnly, _
       "File Does Not Exist"
    bErrorOccurred = True
    Unload Me
    Exit Sub
  End If

        'displays the dialog box for user to select sheet(s)
  sSheets =
       frmTestXLSConverterX.XLSConv1.ShowSheetSelectionDialog _
       (frmTestXLSConverterX.strSourceFile, False)
  If (sSheets <> "") Then
    txtCopySheetOriginal = sSheets
  End If
End Sub
```

TransposeSheetData

Description: Make all the data in the sheet change location by swapping Cell and Row numbers.

Parameters:

TransposeSheetData(sPathOriginal As String, sPathTarget As String, sSheet As String) As Long

Parameter	Meaning
sPathOriginal	The source workbook file
sPathTarget	Can be the same as *sPathOriginal*, otherwise a new workbook is created
sSheet	Sheet in *sPathOriginal* whose data will be transposed

Return Values:

0: Success

-3: Unable to create Excel Application. Is it installed?

-4: Unable to destroy Excel Application.

-200: Excel reported an error.

-201: Shareware Expired.

-208: Target Path folder does not exist, even after attempting to create it.

-213: Invalid or missing sheet specified.

Example Code:

Methods: Text Files Manipulation

ConcatenateTwoTextFiles

Description: Places the sSourceFile text at the end of sTargetFile.

Parameters:

ConcatenateTwoTextFiles(*sSourceFile* As String, *sTargetFile* As String) As Long

Parameter	Meaning
SSourceFile	This file will be added to the end of sTargetFile
STargetFile	Location where *sSourceFile* will be added; at the end of *sTargetFile*

Return Values:

0: Success

-2: Unable to open original file

-3: Unable to open target file

-201: "Shareware has expired"

Example Code:

The code below is taken from the sample VB program provided. It demonstrates both the ConcatenateTwoTextFiles() and DeleteEmptyLinesFromTextFile methods.

```
Private Function PerformTextSpecialProcess(lngProcess As Long, _
                                           strSourceFile As String, _
                                           strTargetFile As String) _
                                            As Long

    Dim lngResult As Long
    Select Case (lngProcess)   'choose which of the processes to execute
                               'then call the component
        Case SP_CONCATENATE_TEXT_FILES
            lngResult = XLSConv1.ConcatenateTwoTextFiles(strSourceFile,
                        strTargetFile)
        Case SP_REMOVE_EMPTY_LINES_FROM_TEXT_FILE
            lngResult = XLSConv1.DeleteEmptyLinesFromTextFile _
                        (strSourceFile, strTargetFile)
    End Select

    If (lngResult = 0) Then    'checks that a file was created using the
                               'helper function, FileExists
        If (FileExists(strTargetFile) = False) Then
            lngResult = -100
        End If
    End If

    Select Case (lngResult)    'if error, display appropriate message
        Case -1
            strErr = "Unable to Complete Requested Process"
        Case -100
            strErr = "File does not exist."
    End Select
    PerformTextSpecialProcess = lngResult
End Function
```

DeleteEmptyLinesFromTextFile

Description: Removes blank lines from within a text file. Blank lines include those with none or only space characters.

Parameters:

DeleteEmptyLinesFromTextFile(*sSourceFile* As String, *sTargetFile* As String) As Long

Parameter	Meaning
SSourceFile	Text file to modify
STargetFile	If same as sSourceFile, DeleteEmptyLinesFromText will modify the original file. Otherwise, it will create a new, modified file.

Return Values:

0: Success

-2: Unable to open original file

-3: Unable to open target file

-201: "Shareware has expired"

Example Code:

The code below is taken from the sample VB program provided. It demonstrates both the ConcatenateTwoTextFiles() and DeleteEmptyLinesFromTextFile methods.

```
Private Function PerformTextSpecialProcess(lngProcess As Long, _
                                           strSourceFile As String, _
                                           strTargetFile As String) _
                                            As Long

    Dim lngResult As Long
    Select Case (lngProcess)  'choose which of the processes to execute
                              'then call the component
        Case SP_CONCATENATE_TEXT_FILES
            lngResult = XLSConv1.ConcatenateTwoTextFiles(strSourceFile,
                           strTargetFile)
        Case SP_REMOVE_EMPTY_LINES_FROM_TEXT_FILE
            lngResult = XLSConv1.DeleteEmptyLinesFromTextFile _
                           (strSourceFile, strTargetFile)
    End Select

    If (lngResult = 0) Then   'checks that a file was created using the
                              'helper function, FileExists
        If (FileExists(strTargetFile) = False) Then
            lngResult = -100
        End If
    End If

    Select Case (lngResult)   'if error, display appropriate message
        Case -1
            strErr = "Unable to Complete Requested Process"
        Case -100
            strErr = "File does not exist."
    End Select
    PerformTextSpecialProcess = lngResult
End Function
```

Methods: CSV File Manipulation

csvChangeDelimitationCharacter

Description: This method is used to replace the comma delimiter with a delimiter of your choice. Any valid ASCII character can be specified (1-254). For example if we swapped the comma with the pound symbol "#" (ASCII 35) the file before applying this process looks like:

Data1,Data2,Data3,Data4
Data1,Data2,Data3,Data4

After applying this process it looks like:

Data1#Data2#Data3#Data4

Data1#Data2#Data3#Data4

Parameters:

csvChangeDelimitationCharacter(*sInputFile* As String, *sOutputFile* As String, *sChar* As String) As Long

Parameter	Meaning
sPathOriginal	CSV file to modify.
sPathTarget	Can be same as sPathOriginal in which case sPathOriginal will be overwritten. If a different name is specified, XLSConverterX will create a new file.
sChar	Character to replace the comma as delimiter.

Return Values:

0: Success

-2: "Unable to open or save to target file. Could be a sharing violation, or invalid file type.

-3: "Unable to open original file. Check file format. Could be a sharing violation.

-201: "Shareware has expired"

Example Code:

This VB function is representative of how the component may be used with any of the csv special processes. The value returned by the function may then be further evaluated, as necessary.

```
Private Function BeginCSVSpecialProcess(lngProcess As Long, _
                                        strSourceFile As String, _
                                        strTargetFile As String, _
                                        strNewChar As String, _
                                        strRowsToUse As String, _
                                        strColsToUse As String) As Long

    Dim lngConvResult As Long

        'call the appropriate method from the component with the
        'necessary arguments
    Select Case (lngProcess)
        Case SP_CSV_SURROUND_WITH_QUOTES
            lngConvResult = XLSConv1.csvEncaseEachFieldWithQuotes
                            (strSourceFile, strTargetFile)
        Case SP_CSV_PAD_WITH_SPACES
            lngConvResult = XLSConv1.csvPadWithSpaces(strSourceFile,
                            strTargetFile)
        Case SP_CSV_CHANGE_DELIMITER
            lngConvResult = XLSConv1.csvChangeDelimitationCharacter
                            (strSourceFile, strTargetFile, strNewChar)
        Case SP_CSV_REMOVE_EMPTY_LINES
            lngConvResult = XLSConv1.csvRemoveEmptyLines(strSourceFile,
                            strTargetFile)
        Case SP_CSV_INCLUDE_ROWS
            lngConvResult = XLSConv1.csvCropCSVFileRows(strSourceFile,
                            strTargetFile, strRowsToUse)
        Case SP_CSV_INCLUDE_COLS
            lngConvResult = XLSConv1.csvCropCSVFileCols(strSourceFile,
                            strTargetFile, strColsToUse)
        Case SP_CSV_REMOVE_CTL_CHARS
            lngConvResult = XLSConv1.csvRemoveControlCharactersFromFile
                            (strSourceFile, strTargetFile)
        Case SP_CSV_TRIM_EXCESS_COMMAS
            lngConvResult = XLSConv1.csvTrimCSVFile(strSourceFile,
                            strTargetFile)
    End Select

        'if the component returns 0 (success) do a further check that
        'the file was created
    If (lngConvResult = 0) Then
        If (FileExists(strTargetFile) = False) Then
            lngConvResult = -100
        End If
    End If

        'assign an error code, if applicable
    Select Case (lngConvResult)
        Case -2
            strErr = "Unable to open or save to target file.  Could be
                      a sharing violation, or invalid file type."
        Case -3
            strErr = "Unable to open original file. Check file format.
                      Could be a sharing violation."
        Case -100
            strErr = "File does not exist."
    End Select

        'function value returned to caller for additional evaluation
    BeginCSVSpecialProcess = lngConvResult
End Function
```

csvCropCSVFileCols

Description: This method allows you to crop or remove certain columns while keeping others. Those not specified will be removed. For example to keep columns 4-20, 25, and 30 specify: "4-20,25,30" for the *sColsToUse* parameter.

Parameters:

csvCropCSVFileCols(*sInputFile* As String, *sOutputFile* As String, *sColsToUse* As String) As Long

Parameter	Meaning
sPathOriginal	CSV file to use as the original.
sPathTarget	Can be same as *sPathOriginal*, in which case *sPathOriginal* will be modified. If a

	different name is specified, XLSConverterX will create a new file and leave *sPathOriginal* untouched.
sColsToUse	Selection of columns to keep within the CSV file. All others will be discarded. For example to keep cols 4-20, 25, and 30 specify: "4-20,25,30"

Return Values:

0: Success

-2: "Unable to open or save to target file. Could be a sharing violation, or invalid file type.

-3: "Unable to open original file. Check file format. Could be a sharing violation.

-201: "Shareware has expired"

Example Code:

Please review the code sample under csvChangeDelimitationCharacter()

csvCropCSVFileRows

Description: This method allows you to crop or remove certain rows while keeping others. Those not specified will be removed. For example to keep rows 1,9-10, 25, and 30 specify: "1,9-10, 25,30" for the *sRowsToUse* parameter.

Parameters:

csvCropCSVFileRows(*sInputFile* As String, *sOutputFile* As String, *sRowsToUse* As String) As Long

Parameter	Meaning
SPathOriginal	CSV file to modify
SPathTarget	Can be same as sPathOriginal in which case sPathOriginal will be overwritten. If a different name is specified, XLSConverterX will create a new file.
SRowsToUse	/1 = Selection of rows to include. All others will be discarded. For example to keep rows 4-20, 25, and 30 specify: /1 4-20,25,30.

Return Values:

0: Success

-2: "Unable to open or save to target file. Could be a sharing violation, or invalid file type.

-3: "Unable to open original file. Check file format. Could be a sharing violation.

-201: "Shareware has expired"

Example Code:

Please review the code sample under csvChangeDelimitationCharacter()

csvEncaseEachFieldWithQuotes

Description: This method is used to surround each field of a CSV file with quotation marks. For example, the file before applying this process looks like:

Data1,Data2,Data3
Data1,Data2,Data3

After applying this process it looks like:

"Data1","Data2","Data3"

"Data1","Data2","Data3"

Parameters:

csvEncaseEachFieldWithQuotes(*sPathOriginal* As String, *sPathTarget* As String) As Long

Parameter	Meaning
sPathOriginal	CSV file to modify
sPathTarget	Can be same as *sPathOriginal* in which case *sPathOriginal* will be overwritten. If a different name is specified, XLSConverterX will create a new file.

Return Values:

0: Success

-2: "Unable to open or save to target file. Could be a sharing violation, or invalid file type.

-3: "Unable to open original file. Check file format. Could be a sharing violation.

-201: "Shareware has expired"

Example Code:

Please review the code sample under csvChangeDelimitationCharacter()

csvPadForFixedLengthFields

Description: This method will pad each field to a fixed length, with a specifically chosen character. For example, the file before applying this process looks like:

Data001,Data2,Data0003,Data4
Data1,Data000002,Data3,Data04

After applying this process having specified a 10 character field, padded right with a # character, it looks like:

Data001###,Data2#####,Data0003##,Data4#####

Data1#####,Data000002,Data3#####,Data04####

Parameters:

csvPadForFixedLengthFields(sInputFile As String, sOutputFile As String, lLength As Long, sCharToUse As String, bPadLeft As Boolean, sColumn As String) As Long

Parameter	Meaning
sPathOriginal	CSV file to modify
sPathTarget	Can be same as *sPathOriginal* in which case *sPathOriginal* will be overwritten. If a different name is specified, XLSConverterX will create a new file.
lLength	Length to make each field

sChar	Character to pad with, typically a space or “0”
bPadLeft	If TRUE will pad to the left, otherwise will pad to the right
sColumn	Which columns the padding should be applied to. May specify individual columns or ranges such as 1,3-5 or may specify all with an asterisk *

Return Values:

0: Success

-2: "Unable to open or save to target file. Could be a sharing violation, or invalid file type: (") + co.sTargetFile + ")"

-3: "Unable to open original file. Check file format. Could be a sharing violation: (") + co.sOriginalFile + ")"

-201: "Shareware has expired"

Example Code:

csvPadWithSpaces

Description: This method is used to surround each field of a CSV file with space characters, " ". For example, the file before applying this process looks like:

Data1,Data2,Data3,Data4
Data1,Data2,Data3,Data4

After applying this process it looks like:

Data1 , Data2 , Data3 , Data4

Data1 , Data2 , Data3 , Data4

Parameters:

csvPadWithSpaces(*sInputFile* As String, *sOutputFile* As String) As Long

Parameter	Meaning
SPathOriginal	CSV file to modify
SPathTarget	Can be same as *sPathOriginal* in which case *sPathOriginal* will be overwritten. If a different name is specified, XLSConverterX will create a new file.

Return Values:

0: Success

-2: "Unable to open or save to target file. Could be a sharing violation, or invalid file type.

-3: "Unable to open original file. Check file format. Could be a sharing violation.

-201: "Shareware has expired"

Example Code:

Please review the code sample under csvChangeDelimitationCharacter()

csvRemoveControlCharactersFromFile

Description: Sometimes unwanted control characters (like carriage returns) can make their way into the data. Use this special process to remove any unwanted control characters. This does not include the carriage return and linefeed at the end of each line of course.

Parameters:

csvRemoveControlCharactersFromFile(*sInputFile* As String, *sOutputFile* As String) As Long

Parameter	Meaning
sPathOriginal	CSV file to modify
sPathTarget	Can be same as sPathOriginal in which case sPathOriginal will be overwritten. If a different name is specified, XLSConverterX will create a new file.

Return Values:

0: Success

-2: "Unable to open or save to target file. Could be a sharing violation, or invalid file type.

-3: "Unable to open original file. Check file format. Could be a sharing violation.

-201: "Shareware has expired"

Example Code:

Please review the code sample under csvChangeDelimitationCharacter()

csvRemoveEmptyLines

Description: This method is used to remove empty lines from a CSV file. Empty lines constitute those lines that do not have any data between the ",". Blank characters are not considered data. For example, the file before applying this process looks like:

Data1,Data2,Data3,Data4
,, ,
Data1,Data2,Data3,Data4
Data1,Data2,Data3,Data4

After applying this process it looks like:

Data1,Data2,Data3,Data4
Data1,Data2,Data3,Data4

Data1,Data2,Data3,Data4

Parameters:

csvRemoveEmptyLines(sInputFile As String, sOutputFile As String) As Long

Parameter	Meaning
sPathOriginal	CSV file to modify
sPathTarget	Can be same as sPathOriginal in which case sPathOriginal will be overwritten. If a different name is specified, XLSConverterX will create a new file.

Return Values:

0: Success

-2: "Unable to open or save to target file. Could be a sharing violation, or invalid file type.

-3: "Unable to open original file. Check file format. Could be a sharing violation.

-201: "Shareware has expired"

Example Code:

Please review the code sample under csvChangeDelimitationCharacter()

csvTrimCSVFile

Description: This method is used to remove excess commas from a CSV file. Those commas after the last piece of valid data will be removed. For example, the file before applying this process looks like:

Data1,Data2,Data3,Data4,,,
,, ,
Data1,Data2,Data3,Data4,,,
Data1,Data2,Data3,Data4

After applying this process it looks like:

Data1,Data2,Data3,Data4
,,
Data1,Data2,Data3,Data4
Data1,Data2,Data3,Data4
Data1,Data2,Data3,Data4

Notice the space character, though not shown in row 2, column 2, is valid data.

Parameters:

csvTrimCSVFile(*sInputFile* As String, *sOutputFile* As String) As Long

Parameter	Meaning
sPathOriginal	CSV file to modify
sPathTarget	Can be same as sPathOriginal in which case sPathOriginal will be overwritten. If a different name is specified, XLSConverterX will create a new file.

Return Values:

0: Success

-2: "Unable to open or save to target file. Could be a sharing violation, or invalid file type.

-3: "Unable to open original file. Check file format. Could be a sharing violation.

-201: "Shareware has expired"

Example Code:

Please review the code sample under csvChangeDelimitationCharacter()

txtChangeCharacterIgnoreWithinQuotes

Description: Change any single character to any other single character, comma delimiter included, but will ignore instances of the search character that are inside quotation marks.

Parameters:

txtChangeCharacterIgnoreWithinQuotes(sInputFile As String, sOutputFile As String, sChar As String, sCharReplace As String) As Long

Parameter	Meaning
sPathOriginal	CSV file to modify
sPathTarget	Can be the same as *sPathOriginal*, otherwise a file is created with the original and newly moved sheets.
sChar	Character to swap out; searches for this character
sCharReplace	Character to replace sChar

Return Values:

0: Success

-2: "Unable to open or save to target file. Could be a sharing violation, or invalid file type: (") + co.sTargetFile + ")"

-3: "Unable to open original file. Check file format. Could be a sharing violation: (") + co.sOriginalFile + ")"

-201: "Shareware has expired"

Note:
XLSConverterX checks for pairs of quotation marks, line by line. If data input has an opening quote without a closing quote, output for that line may not be as expected.

Methods: General

ExcelIsInstalled

Description: This routine will return TRUE if MS Excel is installed on the PC. NOTE: This method will check the system first time only, for efficiency.

Parameters: none

ExcelIsInstalled()

StateFileLoad

Description: Reloads the saved properties of the component.

Parameters: none

StateFileLoad()

See Also: StateFile and UseStateFile properties and the StateFileSave() method.

StateFileSave

Description: Saves the properties of the component. Allows the user to save a given configuration for the component.

Parameters: none

StateFileSave()

See Also: StateFile and UseStateFile properties and the StateFileLoad() method.

Events

OnError

Description: Event is raised with the error code being returned as well as the error string describing the error.

Parameters:

Public Event OnError(lErr As Long, sErr As String)

Conversion Formats

Excel File Conversion Formats

File Type	File Type	Constant
XlAddIn	Microsoft Excel Add In (*.XLA)	18
XlCSV	Comma Delimited (*.CSV)	6
XlCSVMac	Comma Delimeted Macintosh (*.CSV)	22
XlCSVMSDOS	Comma Delimited DOS (*.CSV)	24
XlCSVWindows	Comma Delimited Windows (*.CSV)	23
XlCurrentPlatformText		-4158
xlDBF2	dBase II (*.DBF)	7
xlDBF3	dBase III (*.DBF)	8
xlDBF4	dBase IV (*.DBF)	11
XlDIF	Data Interchange Format (*.DIF)	9
xlExcel2	Microsoft Excel 2.0 Worksheet (*.XLS)	16
xlExcel2FarEast	Microsoft Excel 2.0 Worksheet Far East (*.XLS)	27
xlExcel3	Microsoft Excel 3.0 Worksheet (*.XLS)	29
xlExcel4	Microsoft Excel 4.0 Worksheet (*.XLS)	33
xlExcel4Workbook	Microsoft Excel 4.0 Workbook (*.XLW)	35
xlExcel5	Microsoft Excel 5.0/95 Workbook (*.XLW)	39
xlExcel7	Microsoft Excel 7.0/95 Workbook (*.XLW)	39
xlExcel9795	Microsoft Excel 97-2000 & 5.0/95 Workbook (*.XLS)	43
XlHTML	Web Page (*.HTM, *.HTML)	44
XlIntlAddIn		26
XlIntlMacro		25
xlSYLK	SYLK (Symbolik Link) (*.SLK)	2

XlTemplate	Template (*.XLT)	17
XlTextMac	Text Macintosh (*.TXT)	19
XlTextMSDOS	Text (MS-DOS) (*.TXT)	21
XlTextPrinter		36
XlTextWindows	Text Windows (*.TXT)	20
XlUnicodeText	Unicode Text (*.TXT)	42
xlWJ2WD1		14
xlWK1	WK1 (1-2-3) *.WK1	5
xlWK1ALL	WK1 All (1-2-3) *.WK1	31
xlWK1FMT	WK1 FMT (1-2-3) *.WK1	30
xlWK3	WK3 (1-2-3) *.WK3	15
xlWK4	WK4 (1-2-3) *.WK4	38
xlWK3FM3	WK3 FM3 (1-2-3) *.WK3	32
XlWKS	WKS (1-2-3) *.WKS	4
XlWorkbookNormal	Microsoft Excel Workbook (*.XLS)	-4143
xlWorks2FarEast		28
xlWQ1		34
xlWJ3		40
xlWJ3FJ3		41