MaxsUtilLib Automation Programming Guide

July 2022, mtanabe

Contents

Summary

MaxsUtilLib is an in-process COM server of the single-threaded apartment (STA) model. It provides three automation objects.

1) InputBox for running a dialog and interactively collecting text input by a user.
2) ProgressBox for displaying the constantly changing progress of a job with an ability to cancel.
3) VersionInfo for retrieving version number and other product information from an executable's version resource.

This document describes how MaxsUtilLib can be incorporated into applications and how automation objects it exposes can be programmed to achive desired effects.

If you are familiar with how automation works, jump to the Programming Techniques section and start coding.

Automation relies on the COM technology. We are not going into that. But, if you are interested, there is a comprehensive guide. Refer to the Microsoft documentation at https://docs.microsoft.com/en-us/windows/win32/com/component-object-model--com--portal.

Applicable Library

This documentation applies to the library version,

MaxsUtilLib version 1.0 (x64 or x86)

Client-Server Binding

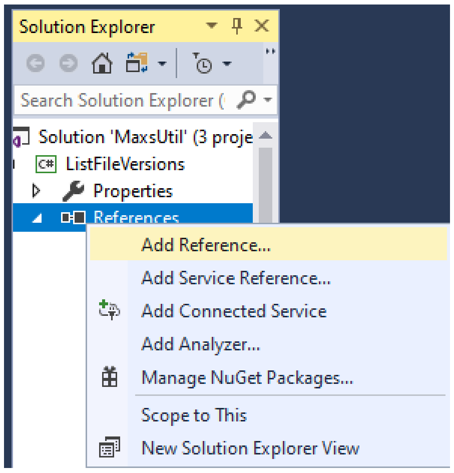
How do I use MaxsUtilLib in my program? The answer depends on the type of code you write. MaxsUtilLib is an automation server providing services that can automate aspects of your program. Your program is a client because it receives the services. Before you can use the services, you must create an instance of the server and bind it to your code.

There are two types of binding mechanisms availabe to an automation client. Early and late bindings. Let's briefly review how the two bindings work and how they can be put to use.

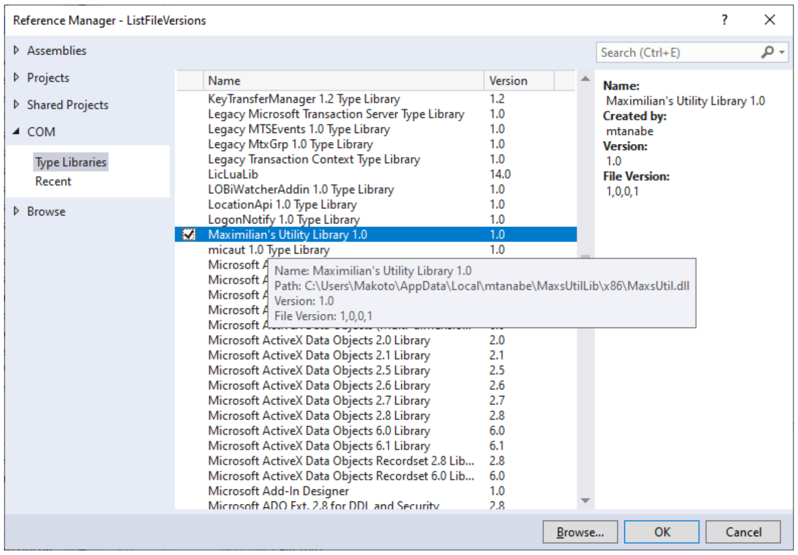
Early Binding

Managed code in C# or VB .NET can use the technique of early binding to match calling statements to unmanaged code in the COM object that implements the functions. This is accomplished at coding time by referencing the COM server's type library and generating an interop assembly, a connector between calling code in the client and the function table of the server. Automation calls are thus resolved at design time. That's why it's called early binding.

In Visual Studio, go to Solution Explorer and right click References. Select Add Reference.



In the Reference Manager, expand COM and select Type Libraries. A list of available automation services appear. Find and pick Maximilian's Utility Library 1.0.

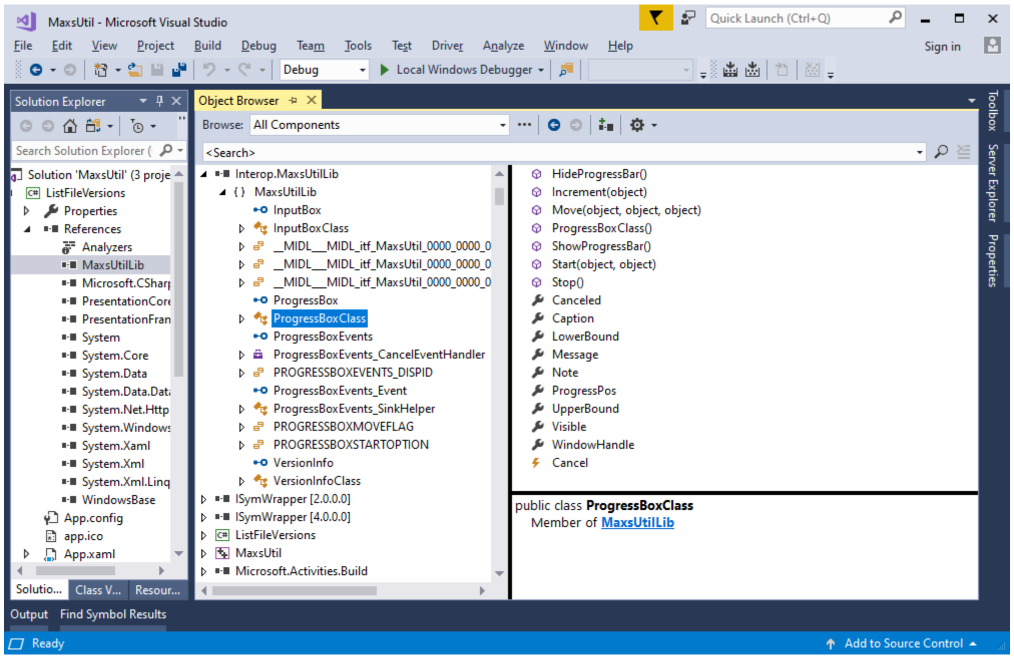


	Name	Version
	KeyTransferManager 1.2 Type Library	1.2
	Legacy Microsoft Transaction Server Type Library	1.0
	Legacy MTSEvents 1.0 Type Library	1.0
	Legacy MtxGrp 1.0 Type Library	1.0
	Legacy Transaction Context Type Library	1.0
	LicLuaLib	14.0
	LOBiWatcherAddin 1.0 Type Library	1.0
	LocationApi 1.0 Type Library	1.0
	LogonNotify 1.0 Type Library	1.0
✓	Maximilian's Utility Library 1.0	1.0
	micaut 1.0 Type Library	1.0
	Microsoft A	
	Microsoft A	
	Microsoft A	
	Microsoft A	
	Microsoft A	
	Microsoft ActiveX Data Objects 2.0 Library	2.0
	Microsoft ActiveX Data Objects 2.1 Library	2.1
	Microsoft ActiveX Data Objects 2.5 Library	2.5
	Microsoft ActiveX Data Objects 2.6 Library	2.6
	Microsoft ActiveX Data Objects 2.7 Library	2.7
	Microsoft ActiveX Data Objects 2.8 Library	2.8
	Microsoft ActiveX Data Objects 6.0 Library	6.0
	Microsoft ActiveX Data Objects 6.1 Library	6.1
	Microsoft ActiveX Data Objects Recordset 2.8 Lib...	2.8
	Microsoft ActiveX Data Objects Recordset 6.0 Lib...	6.0
	Microsoft Add-In Designer	1.0
	Microsoft ADO Ext. 2.8 for DDL and Security	2.8

Name: Maximilian's Utility Library 1.0
Path: C:\Users\Makoto\AppData\Local\mtanabe\MaxsUtilLib\x86\MaxsUtil.dll
Version: 1.0
File Version: 1,0,0,1

Name:
Maximilian's Utility Library 1.0
Created by:
mtanabe
Version:
1.0
File Version:
1,0,0,1

Visual Studio generates a COM interop assembly based on the type library exposed by MaxsUtil.dll. The COM server MaxsUtilLib shows up under References in the Solutions view. Double clicking the added item brings up the Object Browser window listing the server's class and interface definitions.



Solution 'MaxsUtil' (3 proje
ListFileVersions
Properties
References
Analyzers
MaxsUtilLib
Microsoft.CSharp
PresentationCore
PresentationFran
System
System.Core
System.Data
System.Data.Dat
System.Net.Http
System.Windows
System.Xaml
System.Xml
System.Xml.Linq
WindowsBase
App.config
app.ico
App.xaml

Interop.MaxsUtilLib
{ } MaxsUtilLib
InputBox
InputBoxClass
_MIDL__MIDL_itf_MaxsUtil_0000_0000_0
_MIDL__MIDL_itf_MaxsUtil_0000_0000_0
_MIDL__MIDL_itf_MaxsUtil_0000_0000_0
ProgressBox
ProgressBoxClass
ProgressBoxEvents
ProgressBoxEvents_CancelEventHandler
PROGRESSBOXEVENTS_DISPID
ProgressBoxEvents_Event
ProgressBoxEvents_SinkHelper
PROGRESSBOXMOVEFLAG
PROGRESSBOXSTARTOPTION
VersionInfo
VersionInfoClass
ISymWrapper [2.0.0.0]
ISymWrapper [4.0.0.0]
ListFileVersions
MaxsUtil
Microsoft.Activities.Build

HideProgressBar()
Increment(object)
Move(object, object, object)
ProgressBoxClass()
ShowProgressBar()
Start(object, object)
Stop()
Canceled
Caption
LowerBound
Message
Note
ProgressPos
UpperBound
Visible
WindowHandle
Cancel

public class **ProgressBoxClass**
Member of MaxsUtilLib

You are ready to automate your C# application.

Late Binding

A scripting language like JScript does not have ability to early bind an automation object. Instead, it uses the method of late binding It's an IDispatch-based method whereby the client application creates an instance of the COM class from the server library, and binds to a desired service of the class object by calling an invocation method on the object's standard IDispatch interface passing a method or property name that represents the service. It's called late binding because call resolution happens at runtime.

Let's look at a code snippet in JScript and see how late binding works.

```
var inputbox = new ActiveXObject("MaxsUtilLib.InputBox");
if(inputbox.Show("Enter text") > 0) {
  var userinput = inputbox.InputValue;
}
```

Executing the scripted statements runs a dialog and retrieves text entered by a user. But, how did it do that?

This is what happens behind the scene. In the first statement, 'new' operates on ActiveXObject of 'InputBox' creating an instance of the InputBox automation object. The instruction translates to the script host (wscript.exe) calling DllGetClassObject of MaxsUtil.dll for an IDispatch interface of ProgID MaxsUtilLib.InputBox registered with the system. The host stores a pointer to the IDispatch it obtains from the new statement in variable inputbox. In the statement that follows, the host sees expression 'Show(…)' stemming from variable name inputbox. Since it knows inputbox holds a pointer to a IDispatch interface, the host makes a query with the IDispatch to see if 'Show' maps to a method or property on InputBox. It calls GetIDsOfNames on IDispatch to verify that 'Show' is indeed a valid name of a method defined on the object. GetIDsOfNames returns an integer identifier of a method the name represents. The host calls GetTypeInfo on IDispatch to find out what parameters the method accepts. After validating Show's argument of 'Enter text' against the parameter list, the host finally calls Invoke on IDispatch passing the string argument as a parameter for method Show along with the method identifier. The server internally looks up for a calling address of the Show method on a function table and calls the method. As a result, the user sees a dialog run.

That is how late binding works. Note also that in a late bound application, syntactical errors, e.g., a wrong method name, or a malformed parameter list, are only caught at runtime, because validation happens only when the code is executed. Such errors are less likely in an early bound application because the IDE like Visual Studio validates the code on the fly against the type library and flags syntactical errors as you type.

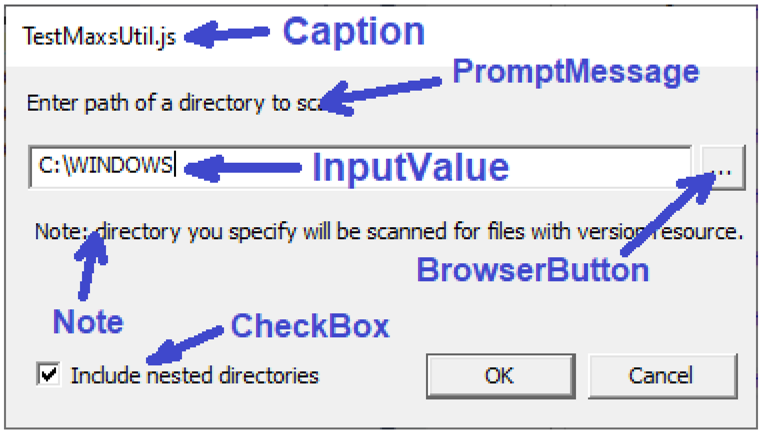
Programming Techniques

We will use JScript and C# snippets to illustrate coding practices that work for useful effects.

Programming with InputBox

Use this automation object to collect a text input from a user in a modal dialog.

The figure below shows correspondence between the UI elements and some of the properties of InputBox.


TestMaxsUtil.js
Caption
PromptMessage
Enter path of a directory to sc
C:\WINDOWS
InputValue
...
Note: directory you specify will be scanned for files with version resource.
BrowserButton
Note
CheckBox
Include nested directories
OK
Cancel

Basic Technique

First thing first, let's learn to run a simple text input dialog and collect text from a user.

JScript

```
var inputbox = new ActiveXObject("MaxsUtilLib.InputBox");
inputbox.Caption = WScript.ScriptName;
if (inputbox.Show("Enter path of a file") <= 0)
  WScript.Quit(2);
var pathname = inputbox.InputValue;
```

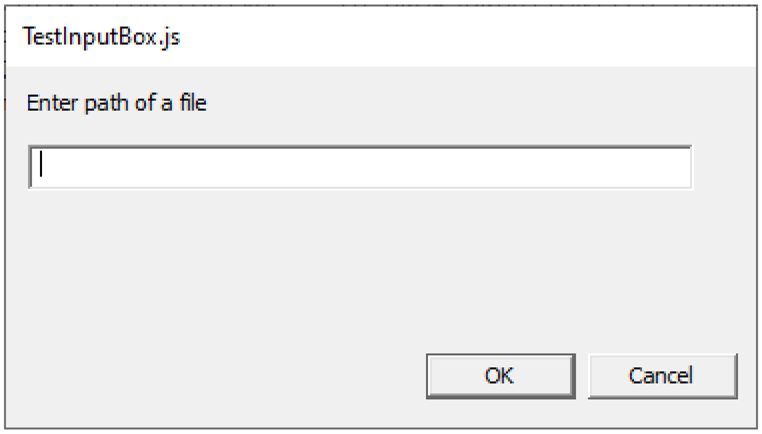
C#

```
using System.Windows.Interop;

var inputbox = new MaxsUtilLib.InputBox();
inputbox.Caption = this.Title;
var hwnd = (new WindowInteropHelper(this)).Handle;
if (inputbox.Show("Enter path of a file ", hwnd) <= 0)
 return;
var pathname = inputbox.InputValue;
```

The expressions in gray are optional. They are meant for passing the HWND handle of a parent window to InputBox.Show. Show still works without a parent handle. However, the dialog window may be pushed back to behind the parent window if the parent window is clicked.

Result



Advanced Technique with a File Browser and a Checkbox Option

Next, let's take advantage of advanced features of InputBox. Typing a pathname can be problematic. A typo can creep in and cause the processing that follows to throw an error. InputBox can run a Win32 common dialog for file browsing, let the user choose his file from a list, and return an accurate pathname. To get that effect, set the BrowserButtonType property to an appropriate IB_BROWSERBUTTONTYPE constant. IB_BROWSERBUTTONTYPE selects a type of standard dialog. The type library defines these dialog types.

```
    IB_BROWSERBUTTONTYPE_NONE = 0,
    IB_BROWSERBUTTONTYPE_FILE_OPEN = 1,
    IB_BROWSERBUTTONTYPE_FILE_SAVEAS = 2,
    IB_BROWSERBUTTONTYPE_FOLDER = 3
```

To restrict the listing of files in the browser dialog to a group of files, assign a filter string to the FileFilter property. Build a filter string according to the format,

```
 <File type description> | <One or more extensions> [| <2nd description> |
  <2nd extensions> [| <3rd...> ...] ] ]
```

The example below combines three filters. The first one lists image files of .jpeg, etc. The second audio files. The third all files.

```
 "Image files|*.jpeg;*.jpg;*.png;*.bmp|Audio files|*.mp3;*.wav;All files|*.*"
```

An application may want to show a check box because it has a boolean option that controls an aspect of the processing that follows the input dialog. You can use the CheckBox property to enable that. Assign a character string that describes the option to CheckBox. If the box should have a check mark placed in advance, set the Checked property to true. After Show returns, read Checked to see if the user has disabled it or left it enabled.

JScript

```
var inputbox = new ActiveXObject("MaxsUtilLib.InputBox");
inputbox.Caption = WScript.ScriptName;
inputbox.Note = "Note: Selected image will be processed.";
inputbox.CheckBox = "Translate to gray scale";
inputbox.Checked = true;
inputbox.BrowserButtonType = IB_BROWSERBUTTONTYPE_FILE_OPEN;
inputbox.FileFilter = "Image files|*.jpg;*.png;*.bmp|All files|*.*";
if (inputbox.Show("Enter path of an image file") <= 0)
  WScript.Quit(2);
var pathname = inputbox.InputValue;
var grayscale = inputBox.Checked;
```

C#

```
var inputbox = new MaxsUtilLib.InputBox();
inputbox.Caption = this.Title;
inputbox.Note = "Note: Selected image will be processed.";
inputbox.CheckBox = "Translate to gray scale";
inputbox.Checked = true;
inputbox.BrowserButtonType =
  MaxsUtilLib.IB_BROWSERBUTTONTYPE.IB_BROWSERBUTTONTYPE_FILE_OPEN;
inputbox.FileFilter = "Image files|*.jpg;*.png;*.bmp|All files|*.*";
var hwnd = (new WindowInteropHelper(this)).Handle;
if (inputbox.Show("Enter path of an image file", hwnd) <= 0)
  return;
var pathname = inputbox.InputValue;
var grayscale = inputBox.Checked;
```

Result

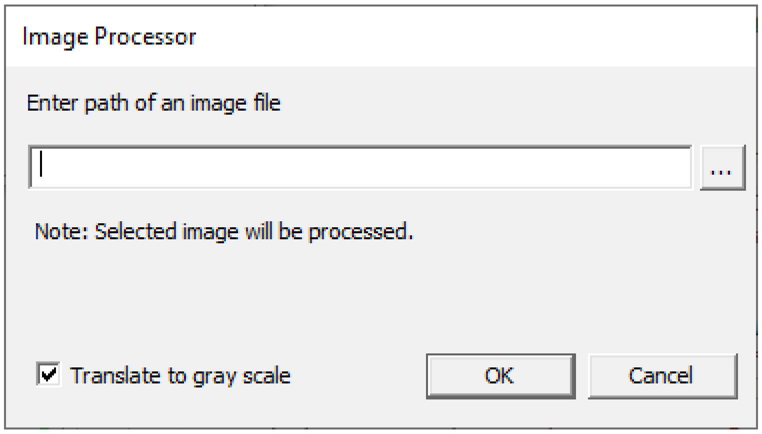
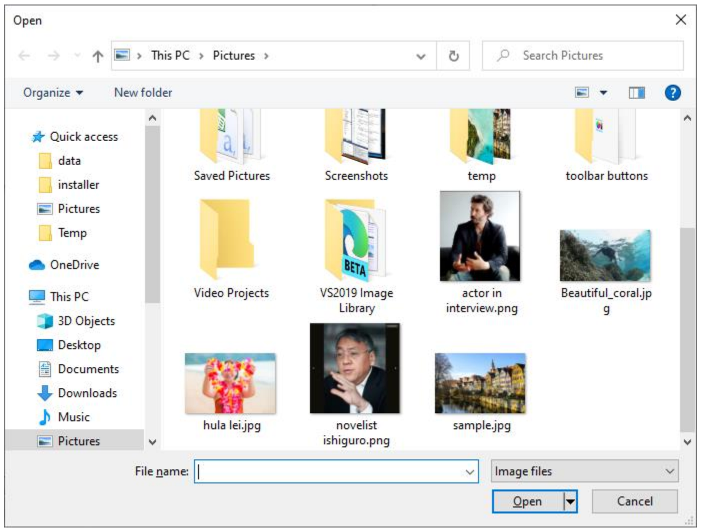


Image Processor

Enter path of an image file

Note: Selected image will be processed.

Clicking the […] button runs the standard File Open dialog of the system. Users can navigate and choose a file from the list.



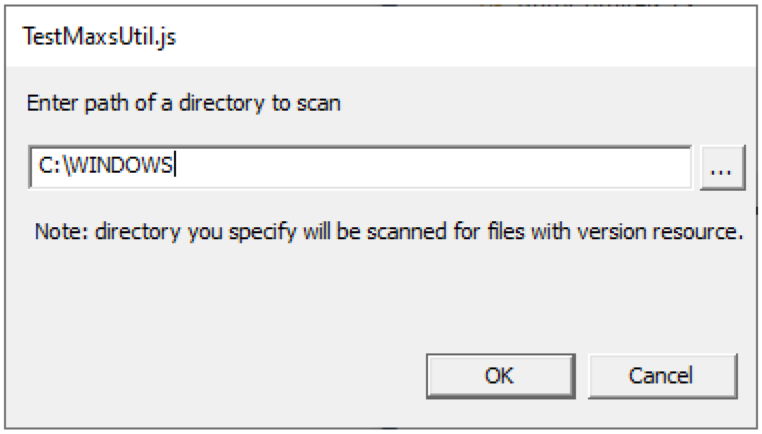
Advanced Technique with a Folder Browser

What if I want the pathname of a directory? Can InputBox run a folder picker type of browser? Yes, it can. Set BrowserButtonType to IB_BROWSERBUTTONTYPE_FOLDER, as shown below.

JScript

```
var inputbox = new ActiveXObject("MaxsUtilLib.InputBox");
inputbox.InitialValue = wsh.ExpandEnvironmentStrings("%windir%");
inputbox.Note = "Note: directory you specify will be scanned for files with
 version resource.";
inputbox.Caption = WScript.ScriptName;
inputbox.BrowserButtonType = IB_BROWSERBUTTONTYPE_FOLDER;
if (inputbox.Show("Enter path of a directory to scan") <= 0)
  WScript.Quit(2);
var srcDir = inputbox.InputValue;
```

Result

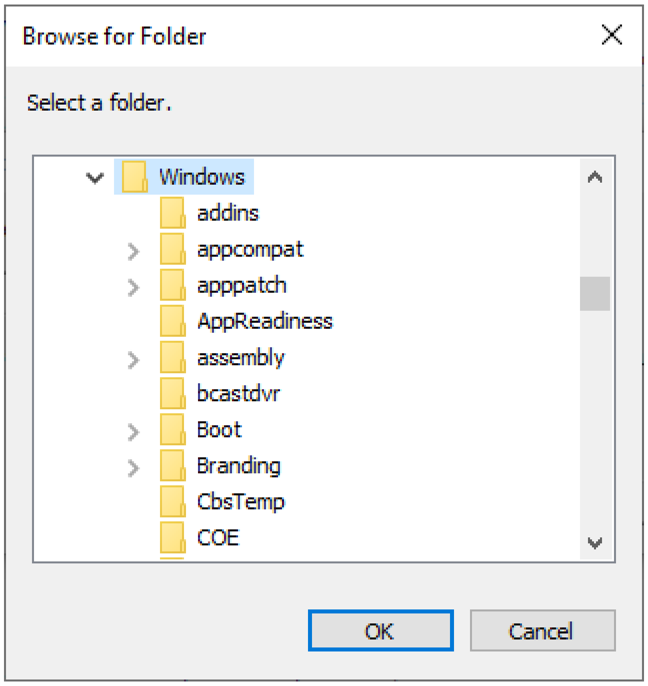

TestMaxsUtil.js

Enter path of a directory to scan

C:\WINDOWS

Note: directory you specify will be scanned for files with version resource.

Clicking the […] button runs a Browser for Folder dialog, iniitially expanding a folder at the input path of InitialValue.

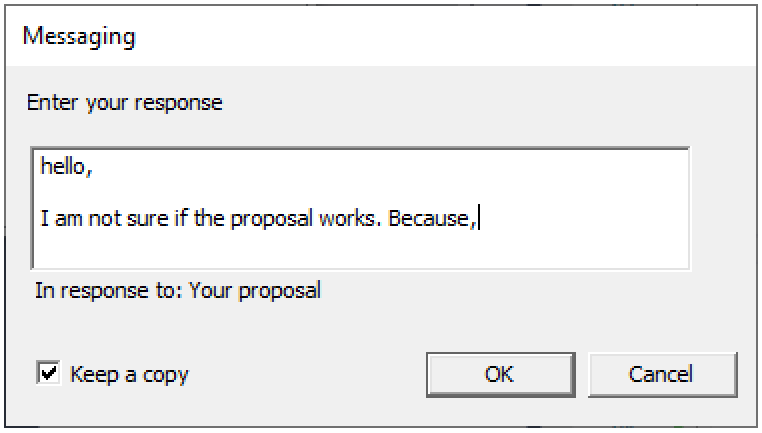


Optional Multi-line Text Entry

InputBox can be instructed to accept more than a single line of text from a user. To do that, set the Options property to the exact string of "IB_MULTILINE" before calling Show..

C#

```
var inputbox = new MaxsUtilLib.InputBox();
inputbox.Caption = "Messaging";
inputbox.Note = "In response to: " + subject;
inputbox.CheckBox = "Keep a copy";
inputbox.Checked = true;
inputbox.Options = "IB_MULTILINE";
var hwnd = (new WindowInteropHelper(this)).Handle;
if (inputbox.Show("Enter your response", hwnd) <= 0)
   return;
var msg2 = inputbox.InputValue;
var keepCopy = inputbox.Checked;
```


Messaging

Enter your response

hello,

I am not sure if the proposal works. Because,

In response to: Your proposal

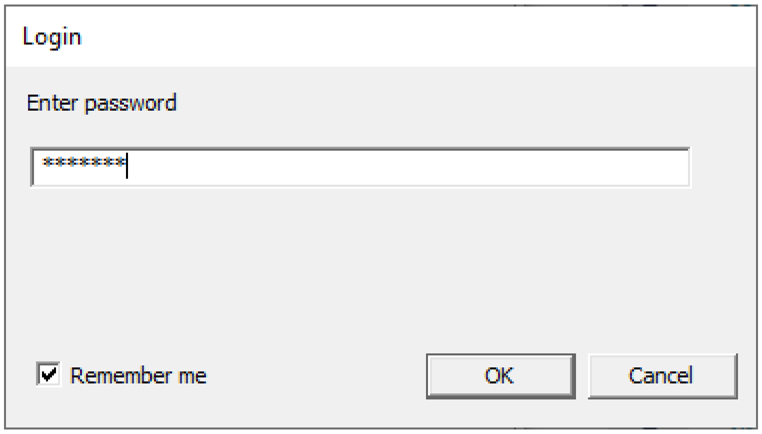
Optional Password or Number-Only Entry

InputBox's Options property can accept a couple of more settings. Set it to "IB_PASSWORD" to show typed characters as a series of '*' to hide the password text. Or, set it to "IB_NUMBER" to restrict input characters to numeric characters, 0 to 9.

JScript

```
var inputbox = new ActiveXObject("MaxsUtilLib.InputBox");
inputbox.Options = "IB_PASSWORD";
if (inputbox.Show("Enter a password") <= 0)
   return;
var pswd = inputbox.InputValue;
```

Result


Login

Enter password

****

Programming with ProgressBox

Use this automation object to display the changing status and progress of an extended operation, e.g., directory scan or file download.

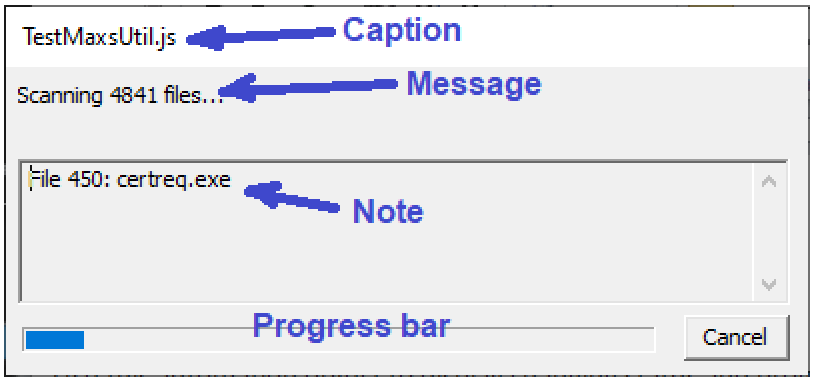
The object spawns a UI thread and runs a modeless dialog. The threaded design allows the object to make updates to the progress bar and status message fields without blocking the main thread where the client application executes.

ProgressBox uses three properties to control the progress display.

- LowerBound - the lower bound of the displayable range.
- UpperBound - the upper bound.
- ProgressPos - current progress position.

The three properties are of the data type of signed 32-bit integer. However, the maximum range possible is 0 to 65535 (0xFFFF). So, limit it to the lower 16 bits when assigning a value to any of these properties. ProgressBox does not accept real numbers in float or double.

The figure below shows correspondence between UI elements and some of the properties of ProgressBox.



Basic Technique

There are three stages of life to ProgressBox.

1) Creation -- an instance is created in a new thread, and the UI components are initialized.
2) Iteration -- a modeless dialog with a progress bar is started entering a loop where the client application iterates through items or repeats tasks and increments the progress position.
3) Termination - the dialog is closed or canceled, and the thread is terminated.

To prepare ProgressBox, at minimum, assign a status message to the Message property, and assign an upper bound to the UpperBound property. If a lower bound is not explicitly assigned, ProgressBox defaults to a lower bound of 0. Nothing is displayed at this point.

To display, call the Start method. A display framework appears with your message. A progress bar is visible there. It is empty since the progress position is at the low end of the range. Do your iterative operation. At each iteration, call the Increment method. Optionally, set the Note property to a description of the item at current iteration. The colored portion of the progress bar grows in direct proportion to the incremented amount.

Your application exits the loop as the job completes. Call the Stop method. The progress bar is stopped and removed. The properties of ProgressBox remain accessible.

Note that you can call Start more than once if you have not called Stop. There are optional settings you can pass to the Start method to change the behavior of ProgressBox. For instance, passing PROGRESSBOXSTARTOPTION_DISABLE_CANCEL to the Start method grays out the Cancel button. You can call Start with that option if you want to disallow cancelation while performing a certain task within a larger iterative process.

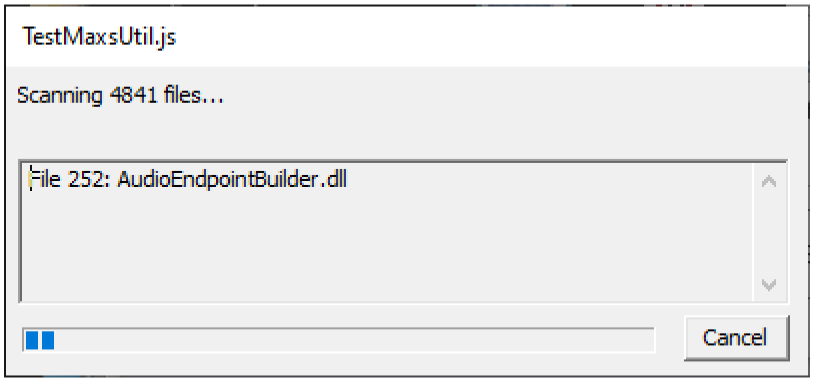
Here is a script for scanning a directory and updating the progress display after each file is processed.

JScript

```
var srcDir = "C:\\Windows\\System32";
var progress = new ActiveXObject("MaxsUtilLib.ProgressBox");
var fld = fso.GetFolder(srcDir);
var fc = fld.Files;
progress.Caption = WScript.ScriptName;
progress.Message = "Scanning "+fc.Count+" files...";
progress.UpperBound = fc.Count;
progress.Start();
var e = new Enumerator(fc);
for (; !e.atEnd(); e.moveNext()) {
 var f = e.item();
 progress.Note = "File "+(progress.ProgressPos+1)+": "+f.Name;
 // do some processing.
 progress.Increment();
 if (progress.Canceled)
 {
  progress.Message = "Scanning canceled.";
  break;
 }
}
progress.Stop();
```

In the above, cancelation by a user is watched by reading the value of the Canceled property. ProgressBox sets the property to true after the Cancel button is clicked.

Result



```
Scanning 4841 files...
File 252: AudioEndpointBuilder.dll
```

Subscribing to the Cancel Notification

In the above JScript example, the Canceled property of ProgressBox was actively read to see if the Cancel button has been clicked. ProgressBox supports event notification for the cancel event. By subscribing to the notification service, your program will automatically receive a call as soon as the event becomes available. This is useful, for instance, if you engage in a tight processing where a subroutine crunches number, and may not have access to the ProgressBox instance, but can check an internal flag that's updated by a cancel listener method.

The example below is from a C# program. It scans a directory for files and add each file and its attribute info to a list for later display. Method ScanDirectory creates a ProgressBox instance. It adds the event handler, Progress_Cancel, to the Cancel event of ProgressBox. It starts a scan loop. If the Cancel button is clicked, Progress_Cancel is called immediately which raises the internal flag, scanCanceled. On seeing the raised flag, ScanDirectory quits scanning and exits the loop.

C#

```
// an internal flag indicating a cancelation from ProgressBox.
bool scanCanceled;

private void ScanDirectory(string dirPath)
{
  int fileCount = Directory.GetFiles(dirPath).Length;

  var progress = new MaxsUtilLib.ProgressBox();
  // subscribe to the notification service. ProgressBox will call our
  listener method as soon as the cancel button is clicked.
  progress.Cancel += Progress_Cancel;
  scanCanceled = false;

  // configure the progress box with a status message and progress range
  progress.Message = "Scanning "+fileCount+"files...";
  progress.UpperBound = fileCount;
  // start the progress display.
  progress.Start();
  // this will collect file info queried from each file in the directory.
  var c = new ObservableCollection<FileAttribList>();

  // run a directory scan and make a query on each file found.
  var files = Directory.EnumerateFiles(dirPath);
  foreach(var f in files)
  {
   progress.Note = Path.GetFileName(f);
   var attribs = QueryFileInfo(f);
   if(attribs == null)
    break; // quit. the cancel button has been hit.
   progress.Increment();
  }
  // show the list of files and collected attribute data.
  FileVersionList.ItemsSource = c;
  // we're done. remove the progress display.
  progress.Stop();
 }

 // responds to a cancelation event from ProgressBox.
 private void Progress_Cancel(ref bool CanCancel)
 {
  // set CanCancel to false to disapprove the cancelation.
  if (ignoreCancel)
   CanCancel = false;
  scanCanceled = CanCancel;
 }

 private FileAttribList QueryFileInfo(string f)
 {
   var attribs = new FileAttribList();
   // build a list of attributes on f.
   …
   if (scanCanceled)
    return null;
   return attribs;
 }
```

Hiding of the Progress Bar

If you do not define a progress range, i.e., by assigning an upper bound greater than the lower bound, the progress bar is not displayed. To unhide the progress bar after the progress dialog is started, do the following:

```
 pb.UpperBound = …;
 pb.ShowProgressBar();
```

where pb is an instance of ProgressBar.

On the other hand, if the progress bar should no longer be visible, you can hide it by doing this.

```
 pb.HideProgressBar();
```

Programming with VersionInfo

Use VersionInfo to retrieve version information, e.g., file version, from a Win32 executable or library file. EXE or DLL may contain a block of Win32 metadata called the VERSIONINFO resource in its file structure. VersionInfo searches the VERSIONINFO for requested pieces of information.

VERSIONINFO Resource

The VERSIONINFO structure has a header section, and a body section of two blocks of data. The header consists of fixed-length records of numeric versioning attributes. The first block is named StringFileInfo. It has one or more language-specific sections each with variable-length records of descriptive textual attributes. The second block is named VarFileInfo. It is an array of supported languages and encodings.

For more details on the data format of VERSIONINFO, refer to the Microsoft documentation at https://docs.microsoft.com/en-us/windows/win32/menurc/versioninfo-resource.

Here is an example of VERSIONINFO with multiple language sections.

```
VS_VERSION_INFO VERSIONINFO
 FILEVERSION 1,2,3,4
 PRODUCTVERSION 2,0,0,1
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
 FILEOS 0x40004L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
 BLOCK "StringFileInfo"
 BEGIN
  BLOCK "040904b0"
  BEGIN
   VALUE "CompanyName", "Maximilian the Cat"
   VALUE "FileDescription", "Test Program"
   VALUE "FileVersion", "1.2.3.4"
   VALUE "InternalName", "TestUtil.exe"
   VALUE "LegalCopyright", "Copyright (C) 2022 maximilian"
   VALUE "OriginalFilename", "TestUtil.exe"
   VALUE "ProductName", "Utility Library"
   VALUE "ProductVersion", "2.0.0.1"
  END
  BLOCK "040704b0"
  BEGIN
    VALUE "CompanyName", "Maximilian die Katze"
    VALUE "FileDescription", "Testprogramm"
    VALUE "FileVersion", "1.2.3.4"
    VALUE "InternalName", "TestUtil.exe"
    VALUE "LegalCopyright", "Urheberrechte © 2022 maximilian"
```

```
      VALUE "OriginalFilename", "TestUtil.exe"
      VALUE "ProductName", "Utility-Bibliothek"
      VALUE "ProductVersion", "2.0.0.1"
    END
    BLOCK "040c04b0"
    BEGIN
      VALUE "CompanyName", "Maximilian le chat"
      VALUE "FileDescription", "Programme d'essai"
      VALUE "FileVersion", "1.2.3.4"
      VALUE "InternalName", "TestUtil.exe"
      VALUE "LegalCopyright", "droits d'auteur © 2022 maximilian"
      VALUE "OriginalFilename", "TestUtil.exe"
      VALUE "ProductName", "Bibliothèque d'utilitaires"
      VALUE "ProductVersion", "2.0.0.1"
    END
  END
  BLOCK "VarFileInfo"
  BEGIN
    VALUE "Translation", 0x409, 0x4b0, 0x407, 0x4b0, 0x40c, 0x4b0
  END
END
```

The above VERSIONINFO has string data in English, German, and French in that order.

The VersionInfo automation object lets you read numeric fields in the header section as integer-typed values, and lets you read descriptive fields in the StringFileFields block as character strings. If there are more than one language section, VersionInfo lets you pick a language and read string data in the language. In a language-dependent query, VersionInfo checks to see if the requested language is in the Translation array, and if so, it reads the encoding ID and forms a key for a section lookup in the StringFileInfo block.

If no language is specified by the client, VersionInfo uses the first language (and code page) found in the Translation section. Use the Lanuage and CodePage properties of VersionInfo to target a secondary language.

Limitation

VersionInfo does not support files with multiple VERSIONINFO resources. It can still read a primary VERSIONINFO. But, it cannot read any of the others.

Available Version Attributes

The attributes you can query from the VERSIONINFO header are shown below.

Attribute name for QueryAttribute	Data type	Remarks
FileVersion	String	Constituent version numbers are combined in the format of <major>.<minor>.<revision>.<build>
ProductVersion	String	<major>.<minor>.<revision>.<build>
Signature	Integer	A magic number signifying the header structure is of Win32 VS_FIXEDFILEINFO.

StrucVersion	Integer	
FileVersionMS	Integer	HIWORD=major version number, LOWORD=minor version number
FileVersionLS	Integer	HIWORD=revision number, LOWORD=build number
ProductVersionMS	Integer	HIWORD=major version number, LOWORD=minor version number
ProductVersionLS	Integer	HIWORD=major version number, LOWORD=minor version number
FileFlagsMask	Integer	A mask to validate FlieFlags
FileFlags	Integer	Bitfield flags, e.g., VFF_DEBUG
FileOS	Integer	VOS_*
FileType	Integer	VFT_*
FileSubtype	Integer	VFT2_*
FileDate	Date	Time stamp

Use a statement of this form to make a query.

```
var attrib = VersionInfo.QueryAttribute( "<attribue name>" );
```

The data type of attrib depends on the attribute name.

The well-known, string-typed attributes you can query from the StringFileInfo section of VERSIONINFO are shown below. There can be other attribute names in use that are unique to the particular VERSIONINFO opened.

Attribute name for QueryAttribute	Remarks
Comments	
CompanyName	
FileDescription	
FileVersion	
InternalName	
LegalCopyright	
LegalTrademarks	
OriginalFilename	
ProductName	
ProductVersion	
PrivateBuild	
SpecialBuild	

Use a statement of this form to make a query.

```
var attrib = VersionInfo.QueryAttribute( "<attribue name>" );
```

The data type of attrib is string.

Automating a Script Program

To use VersionInfo, you assign a pathname of a file to the File property of VersionInfo. Read the VersionString property to get the version number as a character string. A version number string is typically of the form <major version>.<minor version>.<revision number>.<build number>. If you need an integer form of the version number, use the MajorVersion and MinorVersion properties.

The metadata that the VERSIONINFO resource structure includes the name and a description of a product that the file is a part of as well as information on who makes the product.

Here is a sample script. It queries and displays version information on a system file, namely, the EXE of Windows Explorer, by assigning the EXE pathame to VersionInfo and reading VersionString and querying version resource attributes by attribute name.

JScript

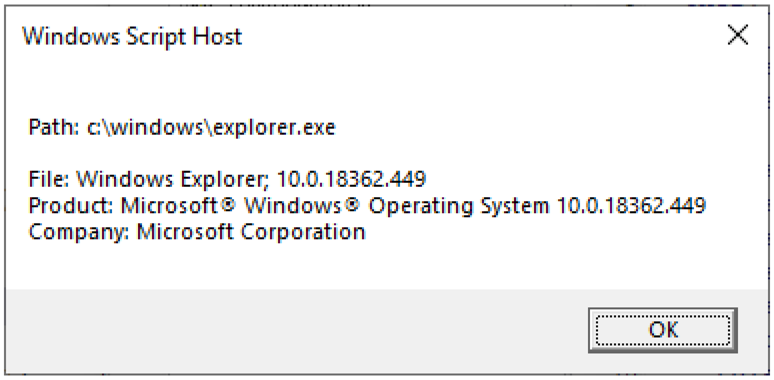
```
var pathname = "c:\\windows\\explorer.exe";

var fso = new ActiveXObject("Scripting.FileSystemObject");
var f = fso.GetFile(pathname);
var fv = new fileVersionInfo(f);
if(fv.errCode == 0)
  WScript.Echo("Path: "+pathname+"\n\nFile: "+fv.fileDescription+";
 "+fv.fileVersion+"\nProduct: "+fv.productName+"
 "+fv.productVersion+"\nCompany: "+fv.companyName);
else
  WScript.Echo(fv.errDesc+"; Code="+fv.errCode);

function fileVersionInfo(f) {
 this._vi = new ActiveXObject("MaxsUtilLib.VersionInfo");
 this._vi.File = f.Path;
 this.errCode = 0;
 this.errDesc = "";
 try {
  this.fileVersion = this._vi.VersionString;
  this.fileDescription = this._vi.QueryAttribute("FileDescription");
  this.productName = this._vi.QueryAttribute("ProductName");
  this.productVersion = this._vi.QueryAttribute("ProductVersion");
  this.companyName = this._vi.QueryAttribute("CompanyName");
 } catch(e) {
  this.errCode = e.number & 0xFFFF;
  if (this.errCode == 0x714)
   this.errDesc = "ERROR_RESOURCE_DATA_NOT_FOUND";
  else if (this.errCode == 0x715)
   this.errDesc = "ERROR_RESOURCE_TYPE_NOT_FOUND";
  else if (this.errCode == 0x716)
   this.errDesc = "ERROR_RESOURCE_NAME_NOT_FOUND";
  else if (this.errCode == 0x717)
   this.errDesc = "ERROR_RESOURCE_LANG_NOT_FOUND";
  else
   this.errDesc = "ERROR "+e.number.toString();
 }
}
```

The above script defines a JScript object called fileVersionInfo to wrap VersionInfo and metadata read from the VERSIONINFO of an input file. In case the file has no VERSIONINFO resource, the object uses a try..catch block. VersionInfo returns an interface error code if it is unable to find the queried metadata. In response, the script host throws an exception. The scripts catches the exception and keeps the error code and generates a description for it so that it can alert the user.

Result



```
Windows Script Host

Path: c:\windows\explorer.exe

File: Windows Explorer; 10.0.18362.449
Product: Microsoft® Windows® Operating System 10.0.18362.449
Company: Microsoft Corporation
```

Automating a WPF Program

The sample code below scans a directory, and builds a list of file names and version attributes. It uses a C# structure to wrap a VersionInfo instance, and expose the queried attributes to a list view.

C#

```
public partial class MainWindow : Window
{
 private void Button_Click(object sender, RoutedEventArgs e)
 {
  String dirPath=null;
  using (var dlg = new FolderBrowserDialog())
  {
   var res = dlg.ShowDialog();
   if (res == System.Windows.Forms.DialogResult.OK)
    dirPath = dlg.SelectedPath;
  }
  if (dirPath == null)
   return;
  int fileCount = Directory.GetFiles(dirPath).Length;
  var c = new ObservableCollection<VersionAttribList>();
  // this will collect version info queried from each file in the source
 directory.
  var c = new ObservableCollection<VersionAttribList>();
  // run a directory scan and make a version query on each file found.
  var files = Directory.EnumerateFiles(dirPath);
  foreach(var f in files)
  {
   var val = new VersionAttribList(f);
   c.Add(val);
```

```
    }
    // show the collected list of file names with their version info.
    FileVersionList.ItemsSource = c;
   }

  public struct VersionAttribList
  {
    String _fpath;
    String _fname;
    String _fversion;
    String _pname;
    String _pversion;
    String _fdesc;

    public VersionAttribList(String filePath)
    {
      var vi = new MaxsUtilLib.VersionInfo();
      vi.File = filePath;
      _fpath = filePath;
      _fname = Path.GetFileName(filePath);
      try
      {
        _fversion = vi.VersionString;
        _pname = vi.QueryAttribute("ProductName");
        _pversion = vi.QueryAttribute("ProductVersion");
        _fdesc = vi.QueryAttribute("FileDescription");
      }
      catch (Exception e)
      {
        _fdesc = "[Error 0x"+Convert.ToString(e.HResult, 16)+";
  "+e.Message+"]";
        _fversion = "(na)";
        _pname = "(na)";
        _pversion = "(na)";
      }
    }

    public String FileName { get { return _fname; } set { _fname = value; } }
    public String FileVersion { get { return _fversion; } set { _fversion =
  value; } }
    public String ProductName { get { return _pname; } set { _pname =
  value; } }
    public String ProductVersion { get { return _pversion; } set { _pversion =
  value; } }
    public String FileDescription { get { return _fdesc; } set { _fdesc =
  value; } }
  }
 }
```

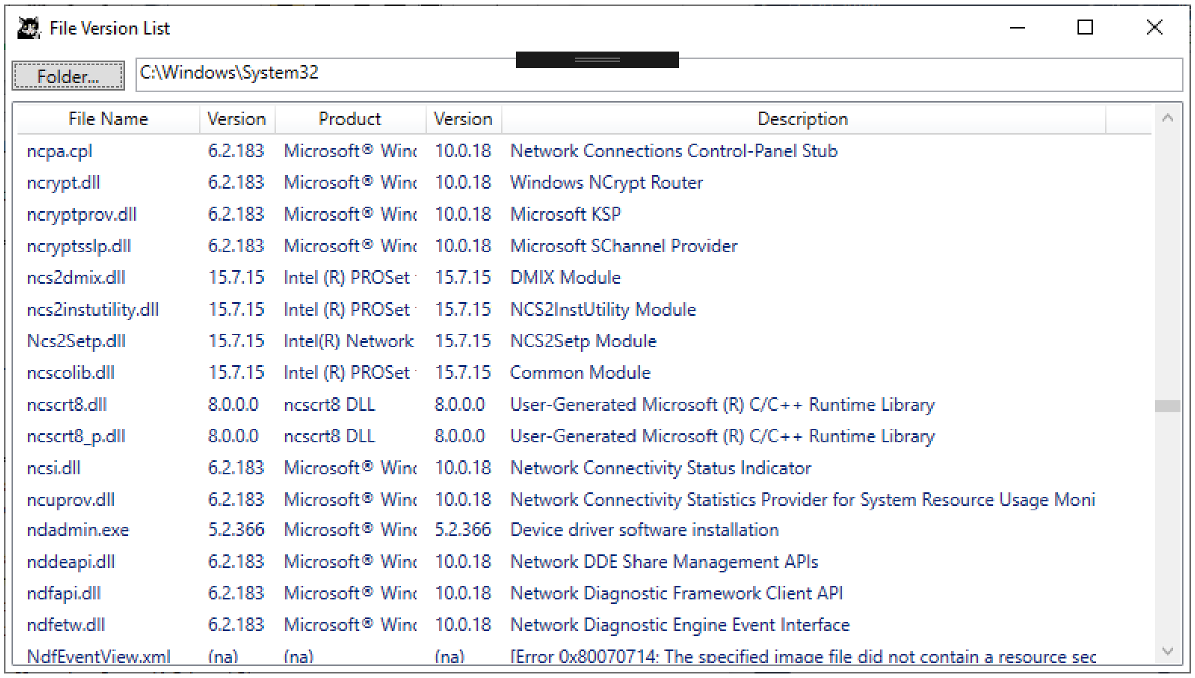
The above code-behind backs this XAML definition.

```
<Window x:Class="ListFileVersions.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

```
       xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
       xmlns:local="clr-namespace:ListFileVersions"
       mc:Ignorable="d"
       Title="File Version List" Height="450" Width="800" Icon="app.ico"
SizeChanged="Window_SizeChanged">
   <Grid>
  <Button x:Name="FolderOpenButton" Content="Folder..." Margin="4,6,0,0"
HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
Click="Button_Click"/>
  <TextBox x:Name="FolderPath" Margin="0,4,4,0" HorizontalAlignment="Right"
VerticalAlignment="Top" Height="23" TextWrapping="Wrap" Width="694"
IsReadOnly="True"/>
  <ListView x:Name="FileVersionList" Margin="4,0,4,4"
HorizontalAlignment="Stretch" Height="374"  VerticalAlignment="Bottom">
   <ListView.View>
    <GridView>
     <GridViewColumn Width="120"
                          DisplayMemberBinding="{Binding FileName}"
                          Header="File Name" />
     <GridViewColumn Width="50"
                          DisplayMemberBinding="{Binding FileVersion}"
                          Header="Version" />
     <GridViewColumn Width="100"
                          DisplayMemberBinding="{Binding ProductName}"
                          Header="Product" />
     <GridViewColumn Width="50"
                          DisplayMemberBinding="{Binding ProductVersion}"
                          Header="Version" />
     <GridViewColumn Width="400"
                          DisplayMemberBinding="{Binding FileDescription}"
                          Header="Description" />
    </GridView>
   </ListView.View>
  </ListView>

 </Grid>
</Window>
```

Result


File Version List

Folder... C:\Windows\System32

File Name	Version	Product	Version	Description
ncpa.cpl	6.2.183	Microsoft® Winc	10.0.18	Network Connections Control-Panel Stub
ncrypt.dll	6.2.183	Microsoft® Winc	10.0.18	Windows NCrypt Router
ncryptprov.dll	6.2.183	Microsoft® Winc	10.0.18	Microsoft KSP
ncryptsslp.dll	6.2.183	Microsoft® Winc	10.0.18	Microsoft SChannel Provider
ncs2dmix.dll	15.7.15	Intel (R) PROSet	15.7.15	DMIX Module
ncs2instutility.dll	15.7.15	Intel (R) PROSet	15.7.15	NCS2InstUtility Module
Ncs2Setp.dll	15.7.15	Intel(R) Network	15.7.15	NCS2Setp Module
ncscolib.dll	15.7.15	Intel (R) PROSet	15.7.15	Common Module
ncscrt8.dll	8.0.0.0	ncscrt8 DLL	8.0.0.0	User-Generated Microsoft (R) C/C++ Runtime Library
ncscrt8_p.dll	8.0.0.0	ncscrt8 DLL	8.0.0.0	User-Generated Microsoft (R) C/C++ Runtime Library
ncsi.dll	6.2.183	Microsoft® Winc	10.0.18	Network Connectivity Status Indicator
ncuprov.dll	6.2.183	Microsoft® Winc	10.0.18	Network Connectivity Statistics Provider for System Resource Usage Moni
ndadmin.exe	5.2.366	Microsoft® Winc	5.2.366	Device driver software installation
nddeapi.dll	6.2.183	Microsoft® Winc	10.0.18	Network DDE Share Management APIs
ndfapi.dll	6.2.183	Microsoft® Winc	10.0.18	Network Diagnostic Framework Client API
ndfetw.dll	6.2.183	Microsoft® Winc	10.0.18	Network Diagnostic Engine Event Interface
NdfEventView.xml	(na)	(na)	(na)	[Error 0x80070714: The specified image file did not contain a resource sec

Finding Supported Languages

Use the QueryTranslation method. The method accepts an index into the Translation array. To find all available languages, repeat calling QueryTranslation by incrementing the index number starting at 1 until the method raises an interface error of E_BOUNDS (code 0x8000000BL). The method returns a combination of a language ID and a code page ID. The low word part is the language ID. The high word part the code page ID. If the language ID is 1033 (or 0x409 in hexadecimal notation), the language the string attributes are in is English. If the code page ID is 1200 (or 0x4b0), the attributes are encoded in Unicode (Little Endian).

After you find out that your preferred language is supported, set the Language and CodePage properties of VersionInfo to the language ID and code page ID. All subsequent calls to the QueryAttribute method search a StringFileInfo section belonging to that language.

If you just want to know about the primary language or primary code page, and if you have not touched the properties, simply read the Language property or the CodePage property, respectively. You could also always call QueryTranslation passing 0 as the index argument.

MaxsUtilLib Reference

This section describes the properties and methods of the MaxsUtilLib automation objects.

InputBox Object

InputBox runs a dialog for collecting a text entry from a user. The object provides access to all of the properties of the dialog box and provides a method to drive the dialog.

The object supports single- and multi-line text entries in normal, password hiding, or digit-only mode. It provides a button to open a file or folder browser button so that the user can pick an item from the list and automatically fills the input field with the pathname. It also provide a check box so that the client application can indicate availability of an optional setting to the user.

Properties

The properties exposed by InputBox are described below.

Properties	Data type	Description
Caption	Text string	Sets or returns the caption text of the InputBox dialog box.
Note	Text string	Sets or returns the text in the note field located below the input field.
CheckBox	Text string	Sets or returns the text of the label assigned to the check box. If this is left empty, the check box is hidden.
Options	Variant (string or integer)	Sets or returns the option flags of IB_MULTILINE (0x4), IB_PASSWORD (0x20) or IB_NUMBER (0x2000).
InitialValue	Text string	Sets or returns the text that initializes the input field.
InputValue	Text string	Returns the text entered in the input field by a user.
Checked	Boolean	Sets or returns the checked state of the check box.

BrowserButtonType	Integer	Sets or returns the browser type ID assigned to the browse button. The following IDs are valid:
		IB_BROWSERBUTTONTYPE_NONE = 0
		IB_BROWSERBUTTONTYPE_FILE_OPEN = 1
		IB_BROWSERBUTTONTYPE_FILE_SAVEAS = 2
		IB_BROWSERBUTTONTYPE_FOLDER = 3

FileFilter	Text string	Sets or returns the file filter string to be passed to the file open or file save dialog on activation of the browser button. The filter string is formatted based on the following template: <File type description> | <One or more extensions> [| <2nd description> | <2nd extensions> [| <3rd...> ...]]

Regarding the setting of a value to the Options property, a string literal of an *IB_OPTION* flag can be passed instead of the integer *IB_OPTION* flag when assigning a value to Options. For example, the string "IB_MULTILINE" can be passed instead of the integer *IB_MULTILINE*.

Methods

InputBox exposes a single method.

Methods	Returned value	Description
Show(PromptMessage, [optional] ParentWindow)	Number of characters of the input text, or -1 if the dialog is canceled.	Starts the text input dialog. The method returns when a user selects the OK or Cancel button. The text contained in PromptMessage is shown above the input field. The optional ParentWindow contains a handle (HWND) of a window that parents the input dialog.

ProgressBox Object

ProgressBox displays the changing status and progress of a task performed by a client application in a compact dialog format with a cancel button allowing the user to cancel the task. It provides access to the properties of the dialog. It provides methods client applications can use to start, stop and move the dialog and manage the progress display.

Properties

These are the properties of ProgressBox.

Properties	Data type	Description
Caption	Text string	Sets or returns the caption text of the ProgressBox dialog.
Message	Text string	Sets or returns the text shown in the primary messaging area.
Note	Text string	Sets or returns the text shown in the secondary messaging area.
LowerBound	Integer	Sets or returns the lower bound of the progress range. The value should be in the range of 0 to UpperBound.
UpperBound	Integer	Sets or returns the upper bound of the progress range. If UpperBound equals LowerBound, the progress bar is not shown when the progress dialog starts. The largest possible value is 65535 (or 0xffff).
ProgressPos	Integer	Sets or returns the current progress position.
Canceled	Boolean	Returns a True if a user selects the Cancel button.
Visible	Boolean	Sets or returns the visibility state of the progress dialog box.
BarColor	Integer	Sets or returns the RGB color value of the progress bar. Returns *CLR_DEFAULT* (0xff000000) if no custom value has been assigned.

WindowHandle	IntPtr (HWND)	Returns the Win32 handle of the dialog window.

Methods

ProgressBox provides these methods.

Methods	Returned value	Description
ShowProgressBar()	None	Unhides the progress bar.
HideProgressBar()	None	Hides the progress bar.

Start([optional] Options, [optional] Params)	None	Starts the progress display.
		Options can be set to one or more of the following option flags:
		PROGRESSBOXSTARTOPTION_DISABLE_CANCEL = 1
		PROGRESSBOXSTARTOPTION_SHOW_PROGRESSBAR = 2
		PROGRESSBOXSTARTOPTION_APPEND_TO_NOTE = 4

Stop()	None	Stops the progress display and closes the dialog.
Increment([optional] Step)	None	Increases the progress position by 1 or the optional amount in Step.

Move(LocationFlag, [optional] X, [optional] Y)	None	Moves the progress window to a screen location specified by LocationFlag or by the coordinates of X and Y.
		LocationFlag can be set to one of the following:
		PROGRESSBOXMOVEFLAG_NONE = -1
		PROGRESSBOXMOVEFLAG_X_Y = 0
		PROGRESSBOXMOVEFLAG_CENTER = 1
		PROGRESSBOXMOVEFLAG_LEFTTOP = 2
		PROGRESSBOXMOVEFLAG_RIGHTTOP = 3
		PROGRESSBOXMOVEFLAG_LEFTBOTTOM = 4
		PROGRESSBOXMOVEFLAG_RIGHTBOTTOM = 5

If method Stop() is called, the ProgressBox instance destroys the display window, and subsequent calls to Start()or other methods or read or write to properties fail. To again enage progress display, the client application can create another ProgressBox instance and start it. Or, it can take this step. Write something to the Caption property. Assigning the same caption text will do. That forces InputBox to open a new display window. Follow it by setting up the progress range and messaging fields for the client's new iterative task. Then, call Start().

VersionInfo Object

VersionInfo provides methods for finding file version and product information in a program file. It provides properties for specifying a program file, and finding the language the version information is expressed in.

Properties

VersionInfo provides these properties.

Properties	Data type	Description
File	String	Sets or returns the pathname of a file for which version info is queried.
Language	Integer	Sets or returns the language ID used by QueryAttribute() to limit the search to a language- and code page-specific section of the VERSIONINFO data of the file.
CodePage	Integer	Sets or returns the code page ID used by QueryAttribute().
MajorVersion	Integer	Returns the major version number of the file.
MinorVersion	Integer	Returns the minor version number of the file.
VersionString	String	Returns the complete version string of the file.

Methods

VesionInfo provides these methods.

Methods	Returned value	Description
QueryAttribute(Name)	Attribute value as string, integer or date	Finds a version attribute of Name in the VERSIONINFO metadata of the file. Raises an interface error if the attribute is not found. Refer to the table in the *Available Version Attributes* section for more information.
QueryTranslation(Index)	LangCode as integer	Finds an entry in the translation array at the specified one-based Index position. Returns a combination of language and code page IDs as a 32-bit integer. Raises an interface error if no entry exists. Set Index to 0 to find the primary lang code.

End of Document