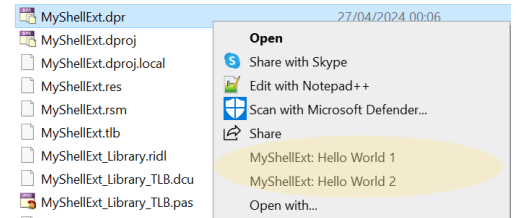


Tutorial- Delphi- Shell Extension- Context Menu- Part 1 (basic context menu)

Scope

This is a Delphi tutorial for implementation of a Windows Shell (Explorer) Extension in form of a Context Menu, which provides functionality when right-clicking on a file in the Windows Explorer.

The tutorial provides a full step-by-step guide building a Delphi project from scratch to achieve the additional context menu functionality on a Windows Explorer, as shown on the figure. The 2 menu entries *MyShellExt: Hello World 1*, and *2* are provided by the example code. On the selection we will provide simple code.



Background

A Windows Shell Extension is expanding the function of the Windows Explorer and adds additional functionality, like a context menu when right-clicking on a file or a selection of files.

This tutorial provides a step-by-step (idiot) guide with screenshots and code snippets you can copy and paste.

Prerequisite

You need a Delphi Compiler - for this project I used Delphi 10 Seattle.

You need Windows Operating System.

Feedback- Help

Friendly feedback is always welcome: delphi@ugarbe.de

What will you Learn

- Delphi
- Active-X COM Object
- Register DLL
- Shell Extension Context Menu

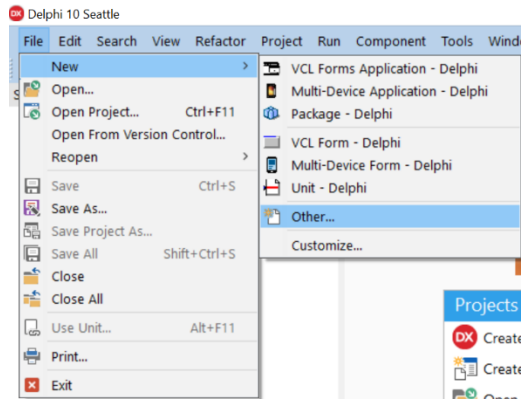
GSCL//Unclassified//

Create an Active-X COM Object

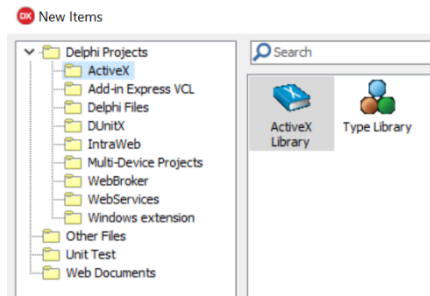
Lets go for it – start your Delphi IDE

Create an Active-X Library

File -> New -> Other

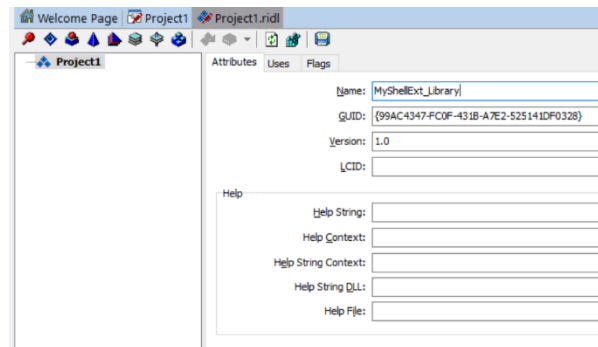


ActiveX -> ActiveX Library

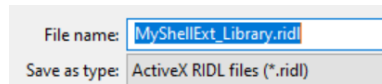


-> a project is created.

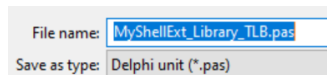
Rename the Attribute Name from Project1 to your plugin name with the _Library. For instance to: MyShellExt_Library:



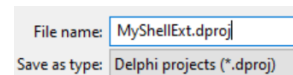
-> Save -> MyShellExt_Library.ridl



-> Save All ->



To save the unit

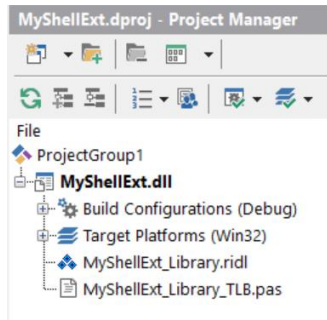


and the project.

GSCL//Unclassified//

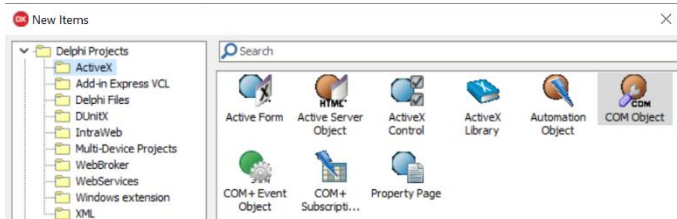
GSCL//Unclassified//

The project should look like this:



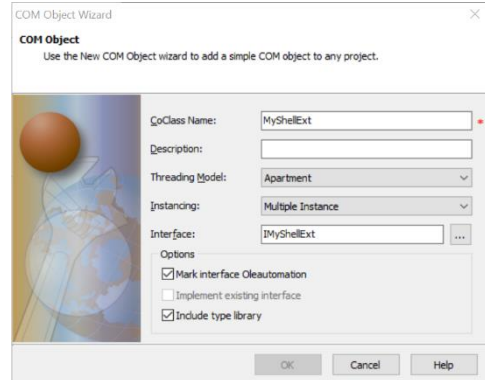
Add a COM Object

-> File -> New -> Other -> ActiveX -> COM Object

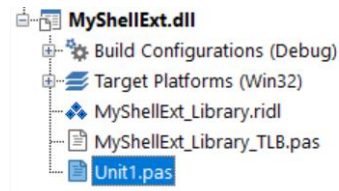


-> give the CoClass Name to: MyShellExt

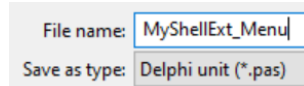
The rest should be fine



There will be a new unit: Unit1



Open Unit1 and save it as: MyShellExt_Menu



GSCL//Unclassified//

GSCL//Unclassified//

Result of COM Object Creation

The unit code looks like this:

```
unit MyShellExt_Menu;
{$WARN SYMBOL_PLATFORM OFF}
interface
uses
  Windows, ActiveX, Classes, ComObj, MyShellExt_Library_TLB, StdVcl;
type
  TMyShellExt = class(ITypedComObject, IMyShellExt)
  protected
  end;
implementation
uses ComServ;
initialization
  ITypedComObjectFactory.Create(ComServer, TMyShellExt, Class_MyShellExt,
    ciMultiInstance, tmApartment);
end.
```

GSCL//Unclassified//

Build the Code for the Shell Extension

Create an ObjectFactory for initialising the COM Object

Now we adopt the code to write the initialisation handler which must register our COM object (the Class we defined:

Class_MyShellExt) and add a procedure to registry keys in the Windows Registry to make the shell extension known to the Explorer.

For this we define a new type: TMyShellExt_Factory with an UpdateRegistry procedure and adopt the code in the initialization section to create and instance of the COM object. Lets also add a finalization section for later.

Here the code for copy pasting:

```
unit MyShellExt_Menu;

{$WARN SYMBOL_PLATFORM OFF}

interface

uses
  Windows, ActiveX, Classes, ComObj, MyShellExt_Library_TLB, StdVcl;

type
  TMyShellExt = class(TTypedComObject, IMyShellExt)
  protected
  end;

  type
  TMyShellExt_Factory = class (TComObjectFactory)
  public
    procedure UpdateRegistry (Register: Boolean); override;
  end;

implementation

uses ComServ;

initialization
  TMyShellExt_Factory.Create(ComServer, TMyShellExt, Class_MyShellExt, 'MyShellExt', 'testing',
  | ciMultiInstance, tmApartment);

finalization

end.
```

```
type
  TMyShellExt_Factory = class (TComObjectFactory)
  public
    procedure UpdateRegistry (Register: Boolean); override;
  end;

implementation

uses ComServ;

initialization
  TMyShellExt_Factory.Create(ComServer, TMyShellExt,
  Class_MyShellExt, 'MyShellExt', 'testing',
  ciMultiInstance, tmApartment);
```

Declare the Shell Extension Interfaces

Now we tackle the type definition of TMyShellExt to provide the interfaces required for a shell extension. Pending which kind of extension you wish to implement different interfaces need to be provided. Lets start easy with a Context Menu, which requires:

```
class(TComObject, IUnknown, IContextMenu, IShellExtInit)
```

The IContextMenu and IShellExtInit require specific procedures being provided by the object. For more info on there query the Microsoft webpages.

The initialisation function is casted to InitShellExt, just for better reading and the IContextMenu requires 3 functions as shown.

Also **add** in the uses clause the **shlObj** library to make the objects know to Delphi. We also define a variable in the private section which can be used by our code later on.

```
uses
  Windows, ActiveX, Classes, ComObj, MyShellExt_Library_TLB, StdVcl, shlObj;

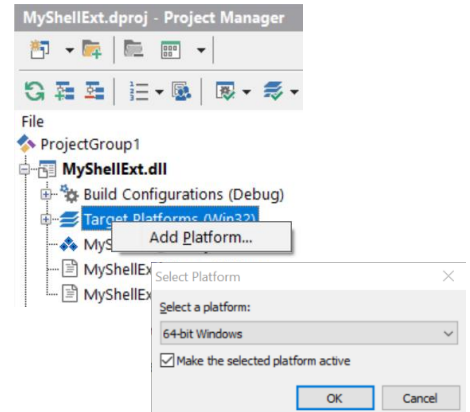
type
  TMyShellExt = class(TComObject, IUnknown, IContextMenu, IShellExtInit)
  private
    FileName: string;
  protected
    {Declare IContextMenu methods here}
    function QueryContextMenu(Menu: HMENU; indexMenu, idCmdFirst, idCmdLast,
      uFlags: UINT): HRESULT; stdcall;
    function InvokeCommand(var lpici: TCMInvokeCommandInfo): HRESULT; stdcall;
    function GetCommandString(idCmd: UINT_PTR; uFlags: UINT; pwReserved: PUINT;
      pszName: LPSTR; cchMax: UINT): HRESULT; stdcall;
    {Declare IShellExtInit methods here}
    function IShellExtInit.Initialize = InitShellExt;
    function InitShellExt (pidlFolder: PItemIDList; lpobj: IDataObject;
      hKeyProgID: HKEY): HRESULT; stdcall;
  end;
```

Adjust Delphi Compiler Options

Before compiling the code adjust the Target Platform to Win64 which is the 64-bit code which most of the Windows installations require today.

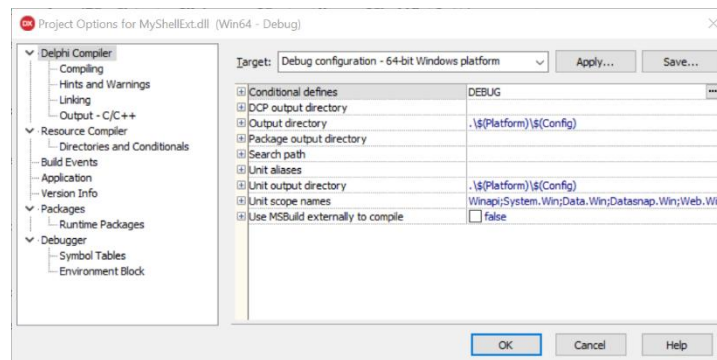
-> right click on Target Platforms (Win32) -> Add Platform

-> select 64-bit Windows



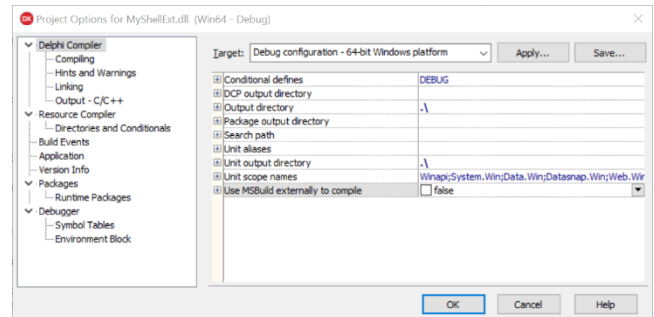
A proposed adjustment is the location where Delphi stores the compiled code, the MyShellExt.dll in our case. This step is not important and is purely to support my style of working 😊

-> Project -> Options



Then you can change the Output Directory and Unit Output Directory entries to .\

After this the .dll will be created in the same directory as where the project is saved. (otherwise the .dll will be in sub-directories – this).



GSCL//Unclassified//

Prepare the Interface Functions Implementation for Compiler Test

Now we prepare the implementation of all functions which have been defined for the defined 2 types: TMyShellExt and TMyShellExt_Factory.

After this step we should be able to compile the code and receive the .dll in the code directory.

Please ensure this works. There will be warnings but there must be no errors.

```

implementation

uses ComServ, Messages, SysUtils, Registry, vcl.dialogs;

function TMyShellExt.InitShellExt (pidlFolder: PItemIDList; lpobj: IDataObject;
  hKeyProgID: HKEY): HRESULT; stdcall;
begin
end;

function TMyShellExt.QueryContextMenu(Menu: HMENU; indexMenu, idCmdFirst, idCmdLast,
  uFlags: UINT): HRESULT; stdcall;
begin
end;

function TMyShellExt.InvokeCommand(var lpici: TCMInvokeCommandInfo): HRESULT; stdcall;
begin
end;

function TMyShellExt.GetCommandString(idCmd: UINT_PTR; uFlags: UINT; pwReserved: PUINT;
  pszName: LPSTR; cchMax: UINT): HRESULT; stdcall;
begin
end;

procedure TMyShellExt_Factory.UpdateRegistry (Register: Boolean);
begin
end;

```

Add the functional code

To see that we are on the right track we need to provide code to the following 2 functions: we need to update the registry, so the Explorer knows which object is serving an Explorer event – in our case a right mouse click.

Registry Entries for Context Menu Handlers

First we link our shell extension class (Class_MyShellExt) to the context menu handlers. This is done through the registry with this code. When later we will register our .dll this procedure is called with the value True, if we unregister it will be called with the value False.

If Register the registry key is created and under the default value the GUID (Globally Unique Identifier) to our object is provided. If Register is false the key will be deleted from the Registry.

```

procedure TMyShellExt_Factory.UpdateRegistry (Register: Boolean);
var
  Reg: TRegistry;
begin
  inherited UpdateRegistry (Register);
  Reg := TRegistry.Create;
  Reg.RootKey := HKEY_CLASSES_ROOT;
  try
    if Register then
      if Reg.OpenKey('*\ShellEx\ContextMenuHandlers\MyShellExt', True) then
        Reg.WriteString('', GUIDToString(Class_MyShellExt));
      if not Register then
        if Reg.OpenKey('*\ShellEx\ContextMenuHandlers\MyShellExt', False) then
          Reg.DeleteKey('*\ShellEx\ContextMenuHandlers\MyShellExt');
    finally
      Reg.CloseKey;
      Reg.Free;
    end;
  end;

```

GSCL//Unclassified//

Display a Menu Item in the Context Menu

Now we can add menu items to the context menu. The function QueryContextMenu provides this feature. InsertMenu is one way of doing it as shown here. We add 2 menu items and need to return the number of added items.

```
function TMyShellExt.QueryContextMenu(Menu: HMENU; indexMenu, idCmdFirst,
idCmdLast,
    uFlags: UINT): HRESULT; stdcall;
begin
    // If the flags include CMF_DEFAULTONLY then we shouldn't do anything
    if (uFlags and CMF_DEFAULTONLY) = CMF_DEFAULTONLY then Result := 0
    else begin
        // add a new item to context menu
        InsertMenu (Menu, indexMenu,
            MF_STRING or MF_BYPOSITION, idCmdFirst,
            'MyShellExt: Hello World 1');
        InsertMenu (Menu, indexMenu+1,
            MF_STRING or MF_BYPOSITION, idCmdFirst+1,
            'MyShellExt: Hello World 2');
        // Return number of menu items added
        Result := 2;
    end;
end;
```


GSC//Unclassified//

First testing of code

Now is the first step where we can see if our code works 😊

Just 2 steps.

-> compile the code – there must be no errors!

Then we need to register the dll to the Windows operating system. You can do this with the Delphi IDE, or through the command line interface. I prefer the hard way through the command line interface.

-> open CMD with administrator privileges! ... and go to the directory of your project.

When you list the directory (dir) you will find our MyShellExt.dll.

Ensure the command prompt runs in Administrator mode (top of screenshot).

```
Administrator: Command Prompt
C:\Users\uli\Documents\Embarcadero\Studio\Projects\Test\ShellExt\06 - MyShellExt>dir
Volume in drive C has no label.
Volume Serial Number is 588A-DB71

Directory of C:\Users\uli\Documents\Embarcadero\Studio\Projects\Test\ShellExt\06 - MyShellExt

27/04/2024 14:30 <DIR>          .
27/04/2024 14:30 <DIR>          ..
27/04/2024 15:02             12.928.976 MyShellExt.dll
27/04/2024 00:06              324 MyShellExt.dpr
27/04/2024 12:01             28.173 MyShellExt.dproj
27/04/2024 12:01             1.893 MyShellExt.dproj.local
27/04/2024 12:02              448 MyShellExt.res
27/04/2024 15:02          24.148.061 MyShellExt.rsm
27/04/2024 11:57              1.480 MyShellExt.tlb
26/04/2024 23:59              1.083 MyShellExt_Library.ridl
27/04/2024 11:57              4.199 MyShellExt_Library_TLB.dcu
26/04/2024 23:58              4.091 MyShellExt_Library_TLB.pas
27/04/2024 15:02              8.717 MyShellExt_Menu.dcu
27/04/2024 14:30              1.857 MyShellExt_Menu.pas

               12 File(s)      37.129.296 bytes
                 2 Dir(s)      986.361.856 bytes free

C:\Users\uli\Documents\Embarcadero\Studio\Projects\Test\ShellExt\06 - MyShellExt>
```

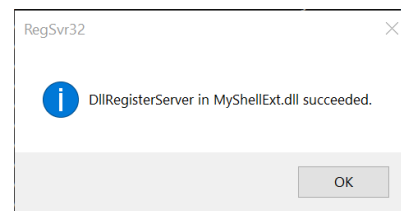
Now you can register the dll with the following command:

```
regsvr32 MyShellExt.dll
```

The registration will be confirmed of being successful.

To unregister use:

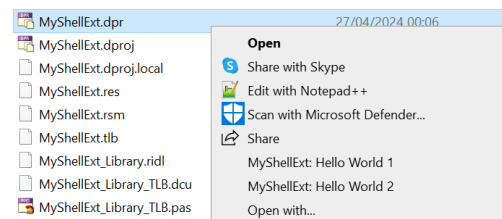
```
regsvr32 -u MyShellExt.dll
```



Now in Explorer, right click any of the files and you should see the 2 menu items 😊

Congratulations

The next steps will add actions to the interface.



GSC//Unclassified//

GSCL//Unclassified//

ONE MORE THING WITH REGISTERED DLLs

As seen above there is a way to register and un-register dlls. Once a dll is registered, you cannot change or delete it. This means you can also not recompile the code, you'll receive an error message from Delphi !!!

After un-registering the dll is still bound to the Windows Explorer – at least for a certain time which I don't know (if you know this, feel free to provide feedback).

The solution: close the instance of Explorer(s) where you tested the functionality. Then reopen Explorer in that directory. I use the right click on the Taskbar, which shows me the last directories used by Explorer, so I don't need to browse back to the folder where our code is stored.

Add Code to the Menu Items

Now we will to provide functionality to the 2 menu items. (remember, we published 2 menu items).

When one of the items is selected the InvokeCommand function is executed. The lpici variable, a 16 bit variable, holds the index of the menu item and some additional context information. In the second half of the code we check the lower byte of the variable, which holds the index. Based on the index we show a message if the 1st menu item has been selected or the 2nd menu item.

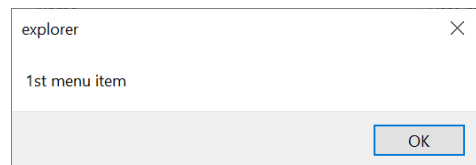
Compile the code and test the DLL.

```
function TMyShellExt.InvokeCommand(var lpici:
TCMInvokeCommandInfo): HRESULT; stdcall;
begin
  Result := NOERROR;
  // Make sure we are not being called by an application
  if HiWord(Integer(lpici.lpVerb)) <> 0 then
  begin
    Result := E_FAIL;
    Exit;
  end;
  // Make sure we aren't being passed an invalid argument number
  if LoWord(lpici.lpVerb) > 1 then
  begin
    Result := E_INVALIDARG;
    Exit;
  end;
  // execute the command specified by lpici.lpVerb.
  if LoWord(lpici.lpVerb) = 0 then
  begin
    showMessage('1st menu item');
  end;
  if LoWord(lpici.lpVerb) = 1 then
  begin
    showMessage('2nd menu item');
  end;
end;
```

HINT: Don't forget to first unregister the DLL, close the Explorer where you tested the code, open a new Explorer at the folder, compile, register ... now you can test.

Here the reward of all the work we have done so far:

Impressive? Almost. Considering that we generate code which integrates with the Windows Operating System and adds new functionality to the Explorer, is impressive, but up to now this is very static and we need one more thing to develop real cool code. We need to know the file(s) which was selected. Then we can apply code to that file.

**GSCL//Unclassified//**

Identify the File(s) Selected

Lets tackle the init function which we casted to InitShellExt. The function is called when a one of our menus is selected. The lpdojb variable links to a data structure which includes the filenames selected and applied to one of our menu items.

The function DragQueryFile (medium.hGlobal, \$FFFFFFFF, nil, 0) provides the nth selected filename, selected with the second parameter of the function. If this parameter is \$FFFFFFFF then it provides the number of selected files.

To keep the code most easy we only accept one file being selected and store the result in the fFileName variable.

To demonstrate that we receive the correct filename, adopt the code in the InvokeCommand function in the 2 showMessage functions to show the fFileName variable, which holds the selected filename.

```
function TMyShellExt.InitShellExt (pidlFolder: PItemIDList; lpdojb:
IDataObject; hKeyProgID: HKEY): HRESULT; stdcall;
var
medium: TStgMedium;
fe: TFormatEtc;
begin
Result := E_FAIL;
// check if the lpdojb pointer is nil
if Assigned (lpdojb) then begin
with fe do begin
cfFormat := CF_HDROP;
ptd := nil;
dwAspect := DVASPECT_CONTENT;
lindex := -1;
tymed := TYMED_HGLOBAL;
end;
// transform the lpdojb data to a storage medium structure
Result := lpdojb.GetData(fe, medium);
if not Failed (Result) then begin
// check if only one file is selected
if DragQueryFile (medium.hGlobal, $FFFFFFFF, nil, 0) = 1 then
begin
SetLength (fFileName, 1000);
DragQueryFile (medium.hGlobal, 0, PChar (fFileName), 1000);
// realign string
fFileName := PChar (fFileName);
Result := NOERROR;
end else
Result := E_FAIL;
end;
ReleaseStgMedium (medium);
end;
end;
```

```
if LoWord(lpici.lpVerb) = 0 then
begin
showMessage('1st menu item selected on file: ' + fFileName);
end;

if LoWord(lpici.lpVerb) = 1 then
begin
showMessage('2nd menu item selected on file: ' + fFileName);
end;
```

Don't forget to: Unregister, close Explorer window, reopen Explorer, compile, register, right click.

Congratulations you made it 😊

This completes Part 1 of this tutorial.