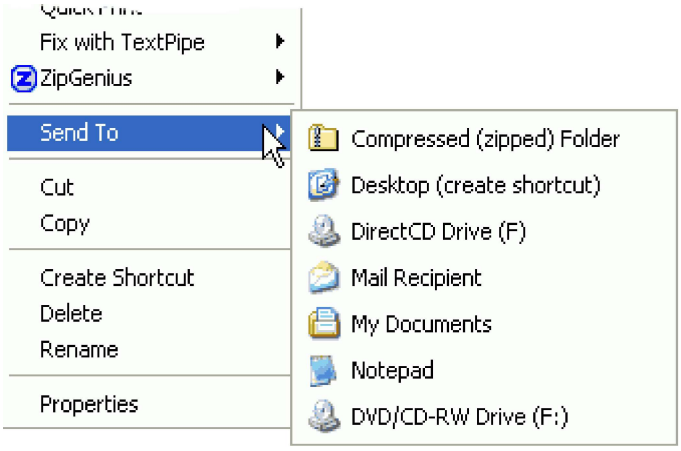
How to add a program to the Explorer Send To menu

Why do it

If you are writing a general purpose file-related program it can be useful to add it to the Windows Explorer "Send To" menu. For example I place Windows Notepad on the "Send To" menu so I can easily view any file in the text editor:



How it's done

There are two stages achieving our goal:

1. Ensuring our program can receive files sent from the "Send To" menu.
2. Creating an entry for our program in the "Send To" menu.

Although you can create a special handler for the "Send To" menu, we will take the simple approach of simply storing a shortcut to our program in the menu.

Receiving files from the "Send To" menu

This is very simple. If the user selects one or more files, right clicks and selects *Send To* | *Our program*, Windows will simply start our program and pass the names of the selected files on the command line. So, to read the files sent from the "Send To" menu, we simply need to check our command line parameters.

We can do this in our application's *FormCreate* event handler as follows:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 1 to ParamCount do
    ProcessFile(ParamStr(I));
end;
```

Listing 1

where *ProcessFile* processes a given file in an application defined way. (In the *demo program* that accompanies this article we simply display the file names in a memo control).

That's all there is to handling the files. Now let's look at how we create the "Send To" menu entry.

Creating the "Send To" menu item

Windows stores the contents of a user's "Send To" menu in a special folder. We will need to add a shortcut to our program in that folder.

We must first find the path to the "Send To" folder. Each user has their own copy of this folder. We find its location by using the *SHGetSpecialFolderLocation* and *SHGetPathFromIDList* API calls (defined in the `ShlObj` unit) as follows:

```
function GetSendToFolder: string;
var
  pidl: PItemIDList;
  PPath: array[0..MAX_PATH] of AnsiChar;
begin
  Result := '';
  if SHGetSpecialFolderLocation(0, CSIDL_SENDTO, pidl) = NOERROR then
  begin
    try
      if SHGetPathFromIDList(pidl, PPath) then
        Result := PPath;
    finally
      CoTaskMemFree(pidl);
    end;
  end;
end;
```

Listing 2

We first use *SHGetSpecialFolderLocation* to get a PIDL representing the required folder. We pass *CSIDL_SENDTO* to the function to get the PIDL for the current user's "Send To" folder. If we succeed in getting the PIDL we then get the path to the folder by passing the PIDL to *SHGetPathFromIDList* and returning a string representation of the PChar buffer we passed to the function. Windows allocated memory for the PIDL using its task allocator. It is our responsibility to free the PIDL, using the *CoTaskMemFree* function (defined in the `ActiveX` unit).

Having found the path to the user's "Send To" folder we can now create a shortcut to our application in the folder. We may choose to do this in a set up program or by making this an option in our application. Whatever our choice, here is the code we use:

```
...
CreateShellLink(
  GetSendToFolder + '\' + 'My SendTo Item.lnk',
  ParamStr(0),
  'My SendTo Sample Program',
  ExtractFileDir(ParamStr(0)),
  '',
  '',
  -1
);
...
```

Listing 3

We are making use of the *CreateShellLink* function from the *Code Snippets Database* to create the shortcut to our program. The parameters passed to *CreateShellLink* are:

1. The first parameter to *CreateShellLink* is the full path to the shortcut file – here we use the name of the "Send To" folder returned by *GetSendToFolder*, followed by the name of the shortcut file. Note that the "Send To" menu displays the name of this file, stripped of it's extension, so give the file a descriptive name.
2. The second parameter is the name of the file referenced by the shortcut – our program in this case. *ParamStr(0)* stores the name of the program.
3. The third parameter is a description of the shortcut (this is *not* displayed in the "Send To" menu).
4. The fourth parameter is our program's working directory. This may be omitted if not required.
5. The fifth parameter stores any command line that is to be passed to the program. We do not use this parameter here. Remember that the "Send To" menu also passes file names on the command line.

6. The sixth parameter is not used here: when provided it specifies the file containing the shortcut's icon. Since we don't specify a file, the icon is assumed to be in the program file.
7. The final parameter also related to the shortcut's icon. Specifying -1 indicates that the program's default icon should be used.

Since this article is not about how to create shell links, we will not go into detail of how this routine works or review its source code here. (The source code *is* included in the accompanying *demo program*).

That's all folks

We have now covered the basics of adding an item to the "Send To" menu and in handling files "sent" to our program from the menu. The code presented can be used as a basis for implementing "Send To" menu support in your programs.

Demo program

The source code of a demo program that illustrates the points covered in this article is available for *download*. The code has been tested with Delphi 4 and Delphi 7.
