

# Demo Script: DevCon 2024

## Download and Save Sample Code

1. Download [twinBASIC BETA 504](#)
2. Extract from Zip folder
3. Open twinBASIC > **Sample 4**
4. Enter Project Name: Demo2024 (*Do not put spaces in the name*)
5. Save as: %tmp%\Demo2024\Demo2024.twinproj
  1. Navigate to: %tmp%
  2. Create folder named: Demo2024
  3. Filename: Demo2024.twinproj

## Customize the Addin Name and Description

Next, let's customize the friendly name and description of our addin. This is the info that appears in the VBA Add-in Manager dialog box.

1. Go to **dllRegistration.twin > DllRegisterServer**
2. In the "FriendlyName" line, replace *AddinProjectName* with "DevCon 2024 Demo"
3. In the "Description" line, replace *AddinProjectName* with "Create a strongly-typed collection class from an existing VBA class object."
4. Save the project
5. Build the project
6. Launch M: \Repos\NLS\DevCon2024\DevCon2024.accdb
7. Switch to VBA: **Ctrl + G**
8. Dock the add-in window
9. Go to "Add-Ins" > "Toggle myToolWindow Visibility"
10. Go to "Add-Ins" > "Add-in Manager"
  - Point out the "DevCon 2024 Demo" item with description below
11. Close "Add-In Manager" window

## Create the Tool Window Controls

1. Open myToolWindow.tbcontrol
2. Select all controls and delete them
3. Select form and set the following properties:
  - Height: 1700
  - Width: 2550
4. Create a text box and set the following properties:
  - Name: tbObjName
  - Anchors > Right:  True

- Height: 300
  - Left: 150
  - Text: *{blank}*
  - TextHint: Object Class Name
  - Top: 150
  - Width: 2250
5. Create a text box and set the following properties:
- Name: tbCollName
  - Anchors > Right:  True
  - Height: 300
  - Left: 150
  - Text: *{blank}*
  - TextHint: Collection Class Name
  - Top: 600
  - Width: 2250
6. Create a button and set the following properties:
- Name: btnCreateClass
  - Anchors > Right:  True
  - Caption: Create Collection Class
  - Height: 450
  - Left: 150
  - Top: 1050
  - Width: 2250

## Edit the Code in myToolWindow.twin

1. Delete the Timer1\_Timer() and HelloWorld\_Click() subroutines
2. Add a Click event handler for btnCreateClass using the code below

```
Private Sub btnCreateClass_Click()  
    MsgBox "Object class name: " & Me.tbObjName.Text & vbNewLine & _  
        "Collection class name: " & Me.tbCollName.Text, vbInformation,  
    "Create Class"  
End Sub
```

## Build and Test the Addin on Same Machine

The following instructions assume a machine with 32-bit Office (e.g., mjw20):

1. Ensure "**win32**" is selected in dropdown
2. **File > Build**
  - Creates and registers this file:  
M:\Repos\NLS\DevCon2024\Build\DevCon2024\_win32.dll
  - As part of registration, the following registry key and values are created:
    - HKEY\_CURRENT\_USER\SOFTWARE\Microsoft\VBA\VBE\6.0\Addins\DevCon202

#### 4. myAddIn\

- Description: "DevCon2024"
- FriendlyName: "DevCon2024"
- LoadBehavior: 3 (3 => [Loaded/Load at startup](#))
- HKEY\_CLASSES\_ROOT\DevCon2024.myAddIn\CLSID
  - (Default): {9B80DA6E-8B20-4D53-AE54-430ACFAE987B} *(this matches the [ClassID()] attribute value above the myAddIn class in myAddIn.twin)*
  - ✖
- HKEY\_CLASSES\_ROOT\DevCon2024.myToolWindow\CLSID
  - (Default): {D531346A-90B8-470D-AA33-FB009F19CEFD} *(this matches the [ClassID()] attribute value above the myToolWindow class in myToolWindow.twin)*
  - ✖
- HKEY\_CLASSES\_ROOT\CLSID\{9B80DA6E-8B20-4D53-AE54-430ACFAE987B}
  - (Default): myAddIn
  - \InProcServer32
    - (Default):
      - M:\Repos\NLS\DevCon2024\Build\DevCon2024\_win64.dll (**NOTE:** *The presence of win64.dll here is likely a result of running the win64 build as shown in the next section*)
    - ThreadingModel: Both
  - \ProgID
    - (Default): DevCon2024.myAddIn
    - ✖
- DEBUG CONSOLE should show this:
  - [LINKER] SUCCESS created output file 'M:\Repos\NLS\DevCon2024\Build\DevCon2024\_win32.dll'
  - [LINKER] → Open Folder (**NOTE:** *this is a clickable link*)
  - [REGISTER] type-library registration completed. DllRegisterServer() returned OK

3. Open Excel or Access

4. Press [Alt] + [F11] to go to VBA IDE

5. Tool window will likely be floating; click and drag to dock it somewhere:



6. Enter Obj Name in the first text box, Coll Name in the second text box, then click [Create Collection Class]



## Build and Test the Addin on a Different Machine and Bitness

The following instructions assume you are building on a machine with 32-bit Office (mjw20), but installing on a machine with 64-bit Office (e.g., gbm18):

1. Ensure "**win64**" is selected in dropdown
2. **File > Build**
3. I copied M:\Repos\NLS\DevCon2024\Build\DevCon2024\_win64.dll to %fb%\12114\DevCon2024\_win64.dll (I will test registering it tomorrow on gbm18)
4. Open a non-admin cmd prompt

5. Run: `regsvr32 DevCon2024_win64.dll`
  - Receive message: "DllRegisterServer in DevCon2024\_win64.dll succeeded."
6. Open Word (or Excel) - The add-in appears.

## Copy and Paste Working VBA Code into twinBASIC

1. Add a standard code module named "MyModule":
  1. Right-click Sources > **Add > Add Module (.TWIN supporting Unicode)**
2. Go to [Strongly-Typed Collections: The Easy Way](#)
  1. Copy and paste the [GetGuidBasedTempPath code](#)
  2. Copy and paste the [FileWrite code](#)
3. Handle "Unrecognized datatype symbol 'Scripting'" error in DIAGNOSTICS pane:
  1. Go to **Project > References**
  2. Switch to "Available COM References" tab
  3. Search for "script" and then click the "Microsoft Scripting Runtime" reference
  4. Click [Save Changes]

## Add fafalone's WinDevLib Package for API Calls

1. **Project > References...**
2. Switch to "Available Packages" tab
3. Search for "windows"
4. Check box next to " Windows Development Library for twinBASIC vX.Y.ZZZ"
  - The package will immediately begin downloading in the background
  - When the download finishes, the name will change to " [IMPORTED] Windows Development Library for twinBASIC vX.Y.ZZZ"
  - NOTE: "[WinDevLib for Implements](#)" is a different package
5. Click [Save Changes]
6. Comment out (or delete) API ``Declare`` lines throughout the project
  - Be aware that if you used non-standard ``Alias`` names, you may need to adjust your API calls to match the standard versions used in WinDevLib
  - **myAddIn.twin:**
    - Delete Private Type RECT structure
    - Delete GetClientRect () function declare
  - **InterProcess.twin:**
    - Delete GetCurrentProcessId () function declare line...
    - ...through Type UUID structure
  - **MyModule.twin:**
    - Delete Sleep sub declare
    - Comment out CoCreateGuid function declare and highlight the failure to compile due to the stricter typing of `id As UUID` in WinDevLib versus `id As Any` in my code
    - Uncomment the CoCreateGuid function to show that explicit API declares override the WinDevLib versions
7. Pass Unicode strings directly to API declare functions
  - Most string-related API functions have ANSI and Unicode versions ("A" and "W" for "ANSI" and

"Wide", respectively)

- Lots of legacy VB6/VBA code use the ANSI version of API functions
- WinDevLib [encourages the use of Unicode versions](#) by default
- This means that code that passes input strings to API functions may require wrapping the string in ``StrPtr()` (or removing ``StrPtr()`) from your existing code
- Remove `StrPtr()` from calls to `FindWindowEx()` in `InterProcess.callerApplicationObject`
- Convert final argument from `0&` to `vbNullString` for calls to `FindWindowEx()` in `InterProcess.callerApplicationObject`

From:

<https://grandjean.net/wiki/> - **Grandjean & Braverman, Inc**

Permanent link:

<https://grandjean.net/wiki/12114/demo?rev=1713155819>

Last update: **2024/04/15 04:36 UTC**

