Create an external DLL from a template

Contents

1. Introduction

This document shows the steps how to create an external DLL to use with the SVObserver External tool from a project template.

It should be possible for an C++ programmer to implement a simple DLL in less than one day's work. For this reason, template solutions are provided that already contain the framework for the SVObserver application. The developer will only have to modify one class and a “define”- file.

There is a single DLL Template available called Template_ExternalDll.

This template can be used to create external DLL with either MIL based images or HBitmap images. If no images are to be used it is best to work in HBitmap mode.

Visual Studio 2015 is used for the Template

1.1. DLL Interface versions.

The template is designed to create 64bit external DLLs vor SVObserver 7.x and higher.

The MIL differences in the SVObserver versions are:

- SVObserver 7.x: MIL 9, 64bit (SVO_INTERFACE_001)
- SVObserver 8.x MIL 10, 64bit (SVO_INTERFACE_001 / SVO_INTERFACE_820)
- SVObserver 10.x MIL 10, 64bit (SVO_INTERFACE_1000)

As most MIL functions calls are unchanged in between MIL9 and 10, it is often possible to compile the same source for both MIL9 and MIL10.

Handling of HBitmap did not change during versions. External DLLs compiled with HBitmap are compatible with all SVObserver Versions vom 7.x

1.2. SVObserver Base interface

The SVObserver Interface was stable until SVObserver 8.20.

SVO_INTERFACE_001 allows Up to 4 Input and 4 Result gray scale images as well as up to 50 Input values and 50 Result values.

This interface is needed for External DLLs running with SVObserver 64bit Version prior to 8.20.

If the template / external dll should include checks that no functionality of SVObserver 8.20 interface is used by accident uncomment in defines.h:

```
#define SVO_INTERFACE_001_ONLY
```

1.3. Changes in the SVObserver 8.20 interface

SVObserver 8.20 did introduce color images, arrays and tables.

Some dlls can be written for both grayscale and color images. If a external DLL needs 8.20 interface support, uncomment in defines.h :

```
#define SVO_INTERFACE_820_OR_HIGHER
```

For most DLLs this is the most suitable selection.

1.4. Changes in SVObserver 10.00 interface

SVObserver 10.00 did introduce names and a help text for results. The template will always expect this values, but they will only be displayed in SVOberver 10.00 and higher.

As the 8.20 interface did introduce some bugs, the full functionality is available from SVObserver 10.00 on.

```
#define SVO_INTERFACE_1000_OR_HIGHER
```

Template and SVObserver both cover all fixes for the interface. In most cases the usage of SVO_INTERFACE_820_OR_HIGHER is sufficient

2. Create a solution

- Copy this template solution.
- Choose a new name, e.g. "Transmogrifier".
- Rename the main folder to that name, e.g. "Transmogrifier".
- Replace “Template” in all filenames with the chosen name. (Currently this is only 1 File)
    - Template-ExternalDll.sln
- Open the following files with any editor and replace with the search function all occurrences of “Template” with the new name.
    - ExternalDLL.vcxproj
        - <ProjectName>Template</ProjectName>
    - <new Name>-externalDLL.sln
        - Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "Template", "ExternalDll.vcxproj", "{F18E13E9-FE92-8282-1B1F-4312951A3D6B}"
- Open the solution with VS2015.
- In "defines.h" replace
    - #define TOOLNAME "TemplateTool"
      with the new Name
- Search for “TODO” and follow the instructions in these comments. Only the following files need to be changed:
    - CDLLTool.cpp
    - CDLLTool.h
    - defines.h

2.1. Special steps for MIL template

For the MIL-template you need the MIL library and include files.

Set the path $(PATH_3RDPARTY) to the window environment variable. Then you need to place the MIL files to the path:

- MIL10 x64: $(PATH_3RDPARTY)\Matrox\MIL10_64bit\Include and $(PATH_3RDPARTY)\Matrox\MIL10_64bit\lib
- MIL9 x64: $(PATH_3RDPARTY)\Matrox\MIL9_64bit_M900B1950R2+\Include and $(PATH_3RDPARTY)\Matrox\MIL9_64bit_M900B1950R2+\lib

3. Changes in the template source code

This section contains information on what needs to be changed in the external DLL template. The corresponding areas are marked with TODO comments in the code.

For most DLLs the modification of these 3 files are sufficient:

- Defines.h
- CDllTool.h
- CDllTool.CPP

3.1. Do not change these files

The following files are template files and should not be changed:

- CMyBitMapInfo.h and CMyBitMapInfo.cpp
- ExternalDll.h, ExternalDll.cpp, ExternalDll.rc
- Guid.h
- StructDefinitions.h

3.2. Defines.h

In defines.h the common structure of the DLL is defined.

For ease of use a set of defines can be used to remove unneeded functionality. These can be uncommented if needed or used by external tools like UnIfDef.

```
#define USE_INPUT_VALUES
#define USE_INPUT_IMAGES
#define USE_RESULT_VALUES
#define USE_RESULT_ARRAYS
#define USE_RESULT_IMAGES
#define USE_RESULT_TABLES

#define EXAMPLECODE
```

E.g. if it is known no Tables will be used, the programmer can
uncomment USE_RESULT_TABLES.

All Example Code in the DLL is enclosed by #ifdef EXAMPLECODE an can be removed.

The Interface to use can be selected using the statements like:

```
#define SVO_INTERFACE_820_OR_HIGHER
```

In the next part, the version numbers and the name of the DLL will be defined. These will be automatically be part of the details of the DLL and will be seen in the SVObserver-GUI, if edited correctly here.

```
#define DLLVERSION                              0,10
#define DLLVERSIONSTRING "DLL-V 0.10 (14.02.2020)"
static const long      static_iToolVersion = 010;
```

Then there are enums to define all inputs and results. The last enum value has to have the form “NUM_....” and automatically contains the number of values (do not use assign numbers to the enum values - this will not work correctly!)

Functions return HRESULTs which can be defined. In all HRESULTs set the flag S_FALSE should be OR-ed in. Example:

```
#define ERRORCODE_INPUTVALUE_DOUBLE_ARRAY_TOOSMALL
        (S_FALSE|(HRESULT) 0x0040)
```

3.3. CDllTool.h

Here is only one TODO. Add parameter you need to the class.

The template is designed with standard member paramters for all input values, input images, result values and result images.

```
std::vector<_variant_t> m_aInputValues;
std::vector<_variant_t> m_aResultValues;
std::vector<_variant_t> m_ResultTables;
std::vector<long> m_aMaxResultArraySizes;
std::vector<long> m_aMaxResultTablesRowSizes;

// for HBitmap Images
std::vector<HBITMAP> m_aInputImageIds;
std::vector<std::unique_ptr<CMyBitMapInfo>> m_aResultImageInfo;
std::vector<ImageDefinitionStruct> m_aResultImageDefs;

// for MIL Images
std::vector<MIL_ID> m_aInputImageIds;
std::vector<MIL_ID> m_aResultImageIds;
```

```
std::vector<ImageDefinitionStruct> m_aResultImageDefs;

// for Tables
std::vector<_variant_t> m_ResultTables;
std::vector<long> m_aMaxResultTablesRowSizes;

// For Arrays
std::vector<long> m_aMaxResultArraySizes;
```

3.4. CDllTool.cpp

3.4.1. Constructor

Add the code you need to initialize your parameter and modules are needed.

3.4.2. getInputImageInformation

```
static void getInputImageInformation(std::array<InputImageInformationStruct,
NUM_INPUT_IMAGES> &rInputImageInfoDef);
```

This method gets information on the input images accepted by the DLL (via the parameter ppInputImageInformation). This parameter is preallocated according to the enum value NUM_INPUT_IMAGES. The method must fill the structure with the required InputImageInformationStructs.

Important: This function will only be called by SVObserver versions from 8.20 on!

3.4.3. getInputValuesDefinition

```
static void getInputValuesDefinition(
std::array<InputValueDefinitionStructEx, NUM_INPUT_VALUES >  &rInputDef);
```

This method gets information on the input values accepted by the DLL (via the parameter inputDef). This parameter is preallocated according to the enum value NUM_INPUT_VALUES. The method must fill the structure with the required values.

Input Values can be of one of the following types:

- VT_I4
- VT_R8
- VT_BSTR
- VT_BOOL (only from SVObserver 8.20 onwards)
- VT_ARRAY | VT_R8 (1 or 2 dimensional) (only from SVObserver 8.20 onwards)
- VT_ARRAY | VT_I4 (1 or 2 dimensional) (only from SVObserver 8.20 onwards)

Samples on how to fill this struct are provided in the template.

3.4.4. getResultValueDefinition

```
static void
getResultValueDefinition(std::array<ResultValueDefinitionStructEx,
NUM_RESULT_VALUES>  &rResultDef);
```

This method must return the definition of the result values in the parameter rResultDef. This parameter is preallocated with the length of the define NUM_RESULT_VALUES. The method must fill the structure with the required values.

Results can be of one of the following types:

- VT_I4
- VT_R8
- VT_BSTR
- VT_BOOL (only from SVObserver 8.20 onwards)
- VT_ARRAY | VT_R8 (1 or 2 dimensional) (only from SVObserver 8.20 onwards)
- VT_ARRAY | VT_I4 (1 or 2 dimensional) (only from SVObserver 8.20 onwards)

Samples on how to fill this struct are provided in the template.

3.4.5. getResultTableDefinitions

```
static void
getResultTableDefinition(std::array<ResultTableDefinitionStructEx,
NUM_RESULT_TABLES>  &rResultTables);
```

This method must return the definition of the result tables

3.4.6. validateInputValue

```
static HRESULT validateInputValue(long lParameterNumber, const VARIANT
vParameterValue);
```

This method should check if the parameter has a valid value. It will be called by SVObserver if the customer changes the value by the GUI. “paramNumber” is the index of the parameter (0 <= paramNumber < NUM_INPUT_VALUES). If "parameter" is valid the method should return S_OK, else it returns an error code, which will be used in the method getErrorMessageString to get an error text.

Info!

This function will only be called when a value is changed in the GUI. And only for the changed value.

3.4.7. getErrorMessageString

```
HRESULT getErrorMessageString(unsigned long ulErrorCode, BSTR*
pbstrErrorMessage);
```

This method will be called to inquire for an error message text corresponding to a given ulErrorNumber. pErrMessage will be allocated by this method and will be freed by the caller. If the return value is not S_OK, nothing has been allocated.

3.4.8. initRun

```
HRESULT initRun(const std::array<ImageDefinitionStruct, NUM_INPUT_IMAGES>
&rImageDefinitions, const std::array<VARIANT, NUM_INPUT_VALUES>
&rInputValues);
```

This method initializes the class and prepares it for going online. It resets all inputs and outputs and should be used for initialization. All image definitions will be set to to those contained in pImageDefinitions, all input values will be set to those contained in pInputValues. Everything which can be done at initialization time should be done here to save time in run mode.

In this method, there are two general TODO-parts and a third part for image templates:

- Define the m_aResultValues, code sample:

```
m_aResultValues[ResultValue_LONG].vt = VT_I4;
m_aResultValues[ResultValue_LONG].lVal = 2;

m_aResultValues[ResultValue_DOUBLE].vt = VT_R8;
m_aResultValues[ResultValue_DOUBLE].dblVal = 1.5;

m_aResultValues[ResultValue_BSTR] = L"Result Value";
```

- In the end, the user should add the DLL special initialization part if required. It can use the input values m_aInputValues and - if an image-template - also the input images m_aInputImages.
- Only for image templates: Define the result image definition (m_aResultImageDefs). You can use the input values (m_aInputValues), because they will have been set in this method before.

3.4.9. run

```
HRESULT run();
```

This method runs this DLL. The input parameters and images are set before and can be used with the member parameter m_aInputValues and m_aInputImages. The results should be set to m_aResultValues and m_aResultImageInfo (for HBITMAP) or m_aResultImages (for Mil) (just use the image buffer and do not create or delete them. The creation and deletion will be handled from SVObserver or from the code of the template which should not be changed.)

A sample of setting the result values:

```
m_aResultValues[ResultValue_LONG].lVal = 2;

m_aResultValues[ResultValue_DOUBLE].dblVal = 1.5;

::SysFreeString(m_aResultValues[ResultValue_BSTR].bstrVal);
int i = 1;
std::wstring strTemp = L"Result Value " + std::to_wstring(i) + L" from run
äöü@†";
m_aResultValues[ResultValue_BSTR].bstrVal =
::SysAllocString(strTemp.c_str());
```

3.4.10. cleanup

```
void cleanup();
```

Here all allocated memory should be freed. If necessary, the developer has to add code here.

4. Overview Methods from CDllTool

For the external DLL there is a list of methods in CDllTool which are used indirectly from the SVObserver. The methods the DLL writer has to change to make the behavior of the DLL as it was described in the section before. In this capital the order of the calls (by SVObserver) are displayed. Some methods are only called if it is written for Mil or HBitmap-DLL. Two use cases are here described. (The order is from SVObserver 8.10, but it should be the same for earlier version)

4.1. Order of methods in initialization case

This happens when a configuration was loaded, a value was changed in edit mode or the run mode is entered:

MIL, HBITMAP: getInputImageInformation

getInputValuesDefinition

getResultValueDefinition

initRun

MIL: setMILInputImages

MIL, HBITMAP: getResultImageDefs

MIL: setMILResultImages

getResultValues

4.2. Order of methods for run

This happens in the run mode, but also in case of a RunOnce after changes were effectuated in edit mode:

setInputValues

MIL: setMILInputImages

HBITMAP: setHBITMAPInputImages

MIL: setMILResultImages

run

getResultValues

getResultTables

HBITMAP: getHBITMAPResultImages

5. Other method in CDllTool

Here is a list of methods that are parts of CDllTool, but do not normally have to be changed by the developer.

5.1. setGUID

```
void setGUID(const GUID guid);
```

Sets the guid of the tool. The GUID will be used to tell apart the instances of this DLL.

5.2. setInputValues

```
HRESULT setInputValues(const std::array<VARIANT, NUM_INPUT_VALUES>
&pInputValues);
```

Sets the input values for the next run to m_aInputValues.

5.3. getResultValues

```
HRESULT getResultValues(VARIANT* paResultValues) const;
```

Copies the result values of the last run from m_aResultValues to paResultValues.

5.4. getResultTables

```
HRESULT getResultTables(VARIANT* pResultTables) const;
```

Copies the result tables of the last run from m_ResultTables to pResultTables

5.5. getResultImageDefs

This method is only available for image templates.

This method should return the result image definition. It creates the structure and fills it with the data from m_aResultImageDefs (which should already have been set by the developer in initRun()). The structure will be have to be freed by the caller. The array ppaStructs must be allocated with the length of NUM_RESULT_IMAGES and the size must be set in plArraySize.

5.6. setHBITMAPInputImages

```
HRESULT setHBITMAPInputImages(const std::array<HBITMAP, NUM_INPUT_IMAGES>
&rHandles);
```

This method is only available for HBITMAP-templates.

This method is get the input images from SVObserver and set it to m_aInputImages.

5.7. getHBITMAPResultImages

```
HRESULT getHBITMAPResultImages(std::array<HBITMAP, NUM_RESULT_IMAGES>
&rHandles) const;
```

This method is only available for HBITMAP-templates.

This method returns the pointers to the result images. It will be set it from m_aResultImageInfo.

5.8. setMILInputImages

```
HRESULT setMILInputImages(const std::array<MIL_ID, NUM_INPUT_IMAGES>
&rMILhandles);
```

This method is only available for MIL-templates.

This method gets the input images from SVObserver and set it to m_aInputImages.

5.9. setMILResultImages

```
HRESULT setMILResultImages(std::array<MIL_ID, NUM_RESULT_IMAGES>
&rMILhandles);
```

This method is only available for MIL-templates.

This method sets the MIL-handle for the result images to the parameter m_aResultImages.

5.10. getResultTablesMaxRowSizes

```
static void
getResultTableDefinition(std::array<ResultTableDefinitionStructEx,
NUM_RESULT_TABLES>  &rResultTables);
```

The Values are set using *m_aMaxResultTablesRowSizes* These values are set during initRun.

5.11. getResultValuesMaxArraySizes

```
HRESULT getResultValuesMaxArraySize(std::array<int, NUM_RESULT_VALUES>
&rArraySizes) const;
```

This method gets the maximum size for all arrays. The input parameter holds NUM_RESULT_VALUES. Values should be set for all array results.

The Values are set using *m_aMaxResultArraySizes* These values are set during *initRun*.

6. Debug Mode

Debug configurations use code provided by DebugUtilities.h.

During debug a debugfile is written. The path and name is defined in defines.h as

```
#define DEBUGFILE "C:\\SVIMDLLDebug_" TOOLNAME ".txt"
```

This Debugfile does have entries for each method called from SVObserver defined in ExternalDll.

Additional Text can be send to this debug file in 2 ways:

```
#if _DEBUG
        fdb(1, "TEXT - lValue1=%ld lValue2=%ld\n",lValue1,lValue2);
#endif
```

or without #if #endif as:

```
 FDB(1, "TEXT - lValue1=%ld lValue2=%ld\n",lValue1,lValue2);
```

The first parameter is the nesting depth. This can be used to make the debug file easier to read.

7. Color Formats

SVObserver will always send color image data in a buffer format BGRX (32 bit), Packed. The first band is Blue, then Green then Red.

All Images do have 3 bands.

7.1. MIL Pixel Access

7.1.1. Buffer image

To create a buffer image, this is a correct way to allocate:

```
m_bufferImage = MbufAllocColor(M_DEFAULT_HOST, 3, mywidth, myheight, 8 +
M_UNSIGNED, M_IMAGE, M_NULL);
```

7.1.2. Pixel Access Example

```
        MIL_INT source = MbufInquire(milImage, M_HOST_ADDRESS, M_NULL);
        MIL_INT pitchByte = MbufInquire(milImage, M_PITCH_BYTE, M_NULL);
        MIL_INT imageHeight = MbufInquire(milImage, M_SIZE_Y, M_NULL);
        MIL_INT imageWidth = MbufInquire(milImage, M_SIZE_X, M_NULL);
        MIL_INT sizeBand = MbufInquire(milImage, M_SIZE_BAND, M_NULL);

        BYTE *lineptr;
        for (int y = 0; y < 3; y++)
        {
                lineptr = &((BYTE*)source)[pitchByte *  y];
                for (int x = 0; x < 3; x++)
                {
                        FDB(1, "P(%03d,%03d): RGB (%03d, %03d, %03d) ", x, y,
lineptr[2], lineptr[1], lineptr[0]);
                        lineptr += 4;
                }
        }
```

7.2. HBitmap

7.2.1. Pixel Access Example

```
        DIBSECTION dibSource;
        //set result image1
        if (::GetObject(m_aInputImageIds[InputImage_Color], sizeof(dibSource),
&dibSource) != 0)
        {
                BYTE *source = (BYTE*)dibSource.dsBm.bmBits;
                BYTE *lineptr;

                LONG pitchByte = dibSource.dsBm.bmWidthBytes;
                LONG imageHeight = dibSource.dsBm.bmHeight;

                // Debug print first 3 columns and row
                for (LONG y = 0; y < 3; y++)
                {
                        lineptr = &(source)[pitchByte *  y];
                        for (int x = 0; x < 3; x++)
                        {
                                FDB(1, "P(%03d,%03d): BGRX (%03d, %03d, %03d)
", x, y, lineptr[2], lineptr[1], lineptr[0]);
                                lineptr += 4;
                        }
                }
        }
```

8. Defines for control structures

8.1. Distinguish in between MIL9 and MIL10

To distinguish in between MIL9 and MIL10, all MIL9 configurations have the preprocessor definition MIL9.

```
#ifdef MIL9
        //MIL 9 Code
#else
        // MIL 10 Code
#endif
```

8.2. Distinguish in between MIL and HBitmap

To distinguish in between Hbitmap and MIL, all HBitmap configurations have the preprocessor definition HBMP.

```
#ifdef HBMP
        // HBMP code
```

```
#else
        // MIL Code
#endif


#ifndef HBMP
        // MIL Code
#endif
```

8.3. Examples in the Template

In the file defines.h an define can be enabled:

```
//#define EXAMPLECODE
```

If this is enabled a loat of example code is enabled for reference.

9. 3rd Party (vision) Libraries (licensing)

Different third party libraries may impose different complications. The reason for most issues will probably be tied to their licensing schemes.
Whoever installs the DLL is responsible for making sure the appropriate licensing has been taken care of.

3rd Party Libraries should always be placed in the $(PATH_3RDPARTY) folder.

9.1. Boost as 3rd Party Library

Boost is often used. It should be placed direct into the $(PATH_3RDPARTY) folder.

- $(PATH_3RDPARTY)\boost_1_72_0 is part of the file **SVGlobalProperties.props**

```
<AdditionalIncludeDirectories>$(PATH_3RDPARTY)\Matrox\MIL9_64bit_M900B1950R2+
\Include;$(PATH_3RDPARTY)\boost_1_72_0</AdditionalIncludeDirectories>
```