WindowComponent C++ Library.

Author: alwyn.j.dippenaar@gmail.com

This library produces the WindowComponent_64.dll component, and is used in most of the C++ applications and libraries as it provides common, re-usable functionality to performs a very wide range of features.

Build:

This project requires Visual Studio 2017 to build.

Dependencies:

This library depends on the following C++ components:

- AlCLogger
- Commctrl.h *Common controls32 header and lib
- Comctl32.lib
- Ws2_32.lib

Usage:

- Include the <***windowComponent_64.h***> C++ Header.
- Link to the ***WindowComponent_64.lib*** library.
- Ensure all dependencies are met, including the dependencies of the dependents themselves.

Common Utilities

All of these functions are exposed and available via the DLL interface, most of them are self-explanatory, and some may have no relevance for applications that are not interested in Win32 GUI.

```
unsigned long long DLLEXPORT FileTimeToInt64(const FILETIME & ft);            // Does stuff.


__int64 DLLEXPORT milliseconds_now();                                          // Milliseconds time now.
char DLLEXPORT *increment_address(char* address_string);              // Increment IP address.
bool DLLEXPORT isnumeric(char* valtocheck);                                    // Checks if a string is numeric.

unsigned long long DLLEXPORT getTotalPhysMemory();                             // Get total physical memory.
unsigned long long DLLEXPORT getProcessPhysicalMemory();              // Get process physical memory.

float DLLEXPORT CalculateCPULoad(unsigned long long idleTicks, unsigned long long totalTicks);         // CalculateCPULoad.
float DLLEXPORT GetCPULoad();
                                          // Get CPU Load.

long DLLEXPORT bintolong32(char* str);                                                                 // Binary to long! (32
bit)
long long DLLEXPORT bintolong64(char* str);                                                            // Binary to long! (64
bit)


void DLLEXPORT splitFileNameAndPath(char* fullPath, char* pathOut, char* fileNameOut);  // Splits a full path into filename and path.

//Stamp the diff in memory
void DLLEXPORT diffMemoryStatus(LPMEMORYSTATUS pMemstatFirst, LPMEMORYSTATUS pMemstatLast, LPWSTR pBufferOut, int pMaxSize);

//Get last error, and format message into string.
void DLLEXPORT getLastErrorMsg(DWORD le, LPWSTR msgBuffer);

//Logs the last error.
void DLLEXPORT logLastError(AlCLogger *vLogger, LPWSTR msgBuffer, bool showMsg);

//Destroys all containers.
void DLLEXPORT destroyAllContainers();

//Converts char to wchar_t
void DLLEXPORT copyCharToWChar(char *src, wchar_t *dest);

//Converts wchar_t to char
void DLLEXPORT copyWCharToChar(wchar_t *src, char *dest);

/** Seeds Random. */
void DLLEXPORT seedRandom();

/** Generate a random float between 2 floats (incl). */
float DLLEXPORT random(float min, float max);

//Gets all registered HWNDS.
HWNDContainer **getHWNDS();

//Checks if a file exists.
bool DLLEXPORT fileexists(LPWSTR fileName);

//Checks if a file exists.
bool DLLEXPORT fileexists(LPSTR fileName);

//Formats bytes.
void DLLEXPORT formatBytes(__int64 bytes, LPWSTR out, int maxSize);
void DLLEXPORT formatBytes(__int64 bytes, LPSTR out, int maxSize);

//Adds an hwnd and container to the list.
void addHWNDComponent(HWND hwnd, ComponentContainer *container);

//Formats time in ms.
void DLLEXPORT formatTimeFromMs(long long timeInMs, LPWSTR out, int size);

//Formats time in ms.
void DLLEXPORT formatTimeFromMs(long long timeInMs, LPSTR out, int size);

//Pads a string.
LPWSTR DLLEXPORT padString(LPWSTR strToPad, wchar_t charToPad, int length, bool leftPad);
//Pads a string.
LPSTR DLLEXPORT padString(LPSTR strToPad, char charToPad, int length, bool leftPad);
```

```
//Pads a string.
LPWSTR DLLEXPORT padString(int number, wchar_t charToPad, int length, bool leftPad);
//Pads a string.
LPWSTR DLLEXPORT padString(long number, wchar_t charToPad, int length, bool leftPad);
//Pads a string.
LPWSTR DLLEXPORT padString(double number, wchar_t charToPad, int length, bool leftPad);
//Pads a string.
LPWSTR DLLEXPORT padString(float number, wchar_t charToPad, int length, bool leftPad);
//Pads a string.
LPWSTR DLLEXPORT padString(__int64 number, wchar_t charToPad, int length, bool leftPad);

/**
	Determines the file size.
**/
INT64 DLLEXPORT determineFileSize(LPWSTR fileName);

INT64 DLLEXPORT determineFileSize(LPSTR fileName);
```

General Classes

These classes may be derived, or used as is, to perform various functions including Win32 GUI operations.

```
/** Used to stream from files. **/
class DLLEXPORT FileStream
{
public:
	FileStream();			//Construtor.
	~FileStream();			//Destructor.

	INT64 fileSize;			//The file size in bytes, indicates the end of the file as well.

	LPWSTR m_FileName;		//The filename being read.
	INT64 readLen;			//used to read the file.

	wifstream* file;		//The file object.

	/** Opens the file. */
	bool openFile(LPWSTR p_FileName);

	/** Closes the file. */
	bool closeFile();

	/**
		Reads from start to end into buffer.
		The buffer is expected to be of correct size.
	**/
	bool readFile(LPWSTR buffer, INT64 start, INT64 end);

};


/**
	This class must be overridden to enable easy testing of any function(s).
**/
class DLLEXPORT TestSuite
{
public:
	TestSuite(AlCLogger* p_Logger);		//Constructor.
	virtual ~TestSuite() {};		//Destructor.

	AlCLogger* logger;

	/** This function must be invoked to start running tests. */
	bool startTests();

	/** This must be implemented to run the actual tests.*/
	virtual bool runTests() { return false; };

	wchar_t tbuffer[2048];

	/** Stamps the results so far to logger. */
	void stampResults();

	int total;					//Test counts.
	int success;
	int fail;


};
```

```
/**
	This class manages high performance timing.
**/
class DLLEXPORT HighPerformanceTimer
{
public:
	/** Constructor. */
	HighPerformanceTimer();

	/** Destructor. */
	~HighPerformanceTimer();

	/** Ticks the timer one on. */
	void tick();

	/** Used to stamp the diff, as human readable to an output buffer. */
	void stampDiff(LPWSTR p_tbuffer, int maxsize);

	LARGE_INTEGER scale;    //Obtained by calling query performance frequency, used to scale the performance counter to ms.
	LARGE_INTEGER counter;  //Obtained by calling query performance counter, converted to ms, at each stamp.

	INT64 tmp;                              //Used to hold the previous tick information.
	INT64 diff;                             //Used to hold/calculate the actual tick value.


};

//Stores window and container handles.
struct HWNDContainer
{
	//Constructor.
	HWNDContainer::HWNDContainer()
	{
		hwnd = NULL;
		container = NULL;
	};

	HWND hwnd;
	ComponentContainer *container;
};


//Contains components.
class DLLEXPORT ComponentContainer
{
public:
	ComponentContainer(AlCLogger *vLogger, HINSTANCE vInst);
	virtual ~ComponentContainer();

	bool addComponent(Component *component);
	void displayCurrentComponents();
	bool createWindowClass(LPWSTR vClassName);
	bool createWindow(LPWSTR windowText);

	//Generic
	bool addComponent(LPWSTR className, int id, LPWSTR text, int x, int y, int width, int height, DWORD style);
	void getText(int componentId, LPWSTR out, int max);
	void setText(int componentId, LPWSTR in, int max);
	LRESULT sendMessage(int componentId, UINT msg, WPARAM w, LPARAM l);

	//Static
	bool addStatic(int id, LPWSTR text, int x, int y, int width, int height, DWORD style);

	//Button
	bool addButton(int id, LPWSTR text, int x, int y, int width, int height, DWORD style);

	//Is button checked?
	bool isButtonChecked(int id);

	//Set button check state.
	void setButtonCheck(int id, bool checked);

	//Edit
	bool addEdit(int id, LPWSTR text, int x, int y, int width, int height, DWORD style);

	//List View
	bool addListView(int id, int x, int y, int width, int height, DWORD style);

	//Add Column to list view.
	bool addListViewColumn(int listViewId, int subItemIndex, LPWSTR colText, int colWidth, int lvcfmtFmt, UINT colMask);

	//Add a item to the listview.
	bool addListViewItem(int listViewId, int itemIndex, LPWSTR coltext, UINT colMask);

	//Set a subitem to the listview.
	bool setListViewSubItem(int listViewId, int itemIndex, int subItemId, LPWSTR coltext, UINT colMask);

	//Listbox
```

```
        bool addListBox(int id, LPWSTR text, int x, int y, int width, int height, DWORD style);
        int listBoxCount(int id);
        void listBoxClear(int id);
        void listBoxRemoveItem(int id, int index);
        void listBoxAddString(int id, LPWSTR string);
        void listBoxGetSelectedItems(int id, int arraySize, int *indexArray);
        int listBoxGetSelectedIndex(int id);

        //Font
        void setFont(int id, HFONT font);
        HWND getHWND(int componentId);

        //Virtual methods.
        virtual LRESULT wndMsgProc(HWND hwnd, UINT m, WPARAM w, LPARAM l, bool defaultX);

        //Click events
        virtual void buttonClicked(int id);
        virtual void buttonDblClicked(int id);

        //Listbox
        virtual void listboxSelectionChange(int id);

        //Window instance.
        HINSTANCE inst;

        //Window handle.
        HWND hwnd;

        //Child Components of this window.
        Component **components;
        int componentCount;

        //The msg used by the message loop.
        MSG msg;

        //The window class.
        WNDCLASSEX windowClass;

        //The window position.
        RECT position;

        //Window syle.
        DWORD style;

        AlCLogger *logger;

        wchar_t tbuffer[2048];
        wchar_t tmp[2048];

        bool fullscreen;
        int screenWidth;
        int screenHeight;

};


//The component class
class DLLEXPORT Component
{
public:
        Component();
        ~Component();

        int id;
        HWND hwnd;

        LPWSTR className;
        LPWSTR text;

        RECT position;
        DWORD style;

};
```

Macro Definitions

These MACRO definitions are used across the board by most of the C++ libraries that reference this.

Most commonly will be the DLLEXPORT macro, which when placed correctly either in front of the class, function or structure will allow that class, function or structure to be directly derived from and/or used in other libraries, or applications that link to or use this DLL library.

```
#ifndef DLLEXPORT
#define DLLEXPORT __declspec(dllexport)
#endif

#ifndef DLLIMPORT
#define DLLIMPORT __declspec(dllimport)
#endif


 #ifndef SAFE_DEL
 #define SAFE_DEL(obj) if (obj) delete obj; obj = NULL;
 #endif

#ifndef _REALLOC_COPYARRAY
#define _REALLOC_COPYARRAY(TYPE, SRCARRAY, SRCARRAYLENVAR, NEWLEN)  TYPE* t = new TYPE[NEWLEN];  SAFE_ZERO(t, TYPE, NEWLEN); int clen
= (NEWLEN < SRCARRAYLENVAR ? NEWLEN : SRCARRAYLENVAR); memcpy(t, SRCARRAY, sizeof(TYPE)* clen);  delete SRCARRAY;  SRCARRAYLENVAR =
NEWLEN; SRCARRAY = new TYPE[SRCARRAYLENVAR]; SAFE_ZERO(SRCARRAY, TYPE, SRCARRAYLENVAR); memcpy(SRCARRAY, t, sizeof(TYPE)* clen);
delete t;
#endif

#ifndef SAFE_ZERO
#define SAFE_ZERO(obj, obj_type, obj_sz) ZeroMemory(obj, sizeof(obj_type) * obj_sz);
#endif

#ifndef ARRAY_DEL
#define ARRAY_DEL(arr, count) if (count > 0) { for (int ix=0; ix<count; ix++) if (arr[ix]) delete [] arr[ix]; delete [] arr; } arr =
NULL;
#endif

#ifndef ARRAY_DEL_ELEMENT
#define ARRAY_DEL_ELEMENT(objArr) if (objArr) delete [] objArr; objArr = NULL;
#endif

#ifndef ARRAY_DEL_SINGLE
#define ARRAY_DEL_SINGLE(arr, count) if (arr && count > 0) { for (int ix=0; ix<count; ix++) if (arr[ix]) delete arr[ix]; delete []
arr; } arr = NULL; count = 0;
#endif

#ifndef ARRAY_RESIZE
#define ARRAY_RESIZE(objTyp, objArray, objCount, obj) objTyp** tmp = new objTyp*[objCount+1]; size_t newsz = sizeof(objTyp*) *
(objCount+1); ZeroMemory(tmp, newsz); if (objArray) {    size_t oldsz = sizeof(objTyp*) * objCount;       memcpy_s(tmp, newsz, objArray,
oldsz); delete [] objArray;     objArray = NULL;} objArray = tmp;objArray[objCount] = obj;objCount += 1;
#endif

#ifndef MEMCPY_WCHAR
#define MEMCPY_WCHAR(dest, src, count) memcpy_s(dest, sizeof(wchar_t) * count, src, sizeof(wchar_t) * count)
#endif

#ifndef ARRAY_ELEMENT_RESIZE
#define ARRAY_ELEMENT_RESIZE(objTyp, obj, count) ARRAY_DEL_ELEMENT(obj); obj = new objTyp[count]; ZeroMemory(obj, sizeof(objTyp) *
count);
#endif

#ifndef FAILED_GOTO
#define FAILED_GOTO(hresult, msg, gotox) if (FAILED(hresult)) { getGlobalAlCLogger()->debug(msg); goto gotox; }
#endif

#ifndef SAFE_RELEASE
#define SAFE_RELEASE(obj) if (obj) obj->Release(); obj = NULL;
#endif

#ifndef B2S
#define B2S(x) (x ? "true" : "false")
#endif

#ifndef STAMPTIME
#define STAMPTIME(time) time = GetTickCount64() - time; char msTime[255]; formatTimeFromMs(time, msTime, 255)
#endif
```