WinObjEx64

Windows Object Explorer 64 bit Plugin architecture overview

Document version 1.0.1 (20 Nov 2019)

Plugins version 1.0.1

Plugins architecture

Implemented as dynamic link libraries (dll).

Dll requirements

Plugin implemented as dll must export "*PluginInit*" (without a quotes) routine. Library must have VERSION_INFO block with *FileDescription* field set to "*WinObjEx64 Plugin*" (case sensitive, without a quotes). This is used by WinObjEx64 to ensure that given dll is a proper plugin.

Plugin initialization

PluginInit is a WinObjEx64 plugin initialization routine with the following prototype

```
BOOLEAN CALLBACK PluginInit(
    _Out_ PWINOBJEX_PLUGIN PluginData
);
```

Parameters

PluginData – pointer to WINOBJEX_PLUGIN structure that will be filled by plugin. This structure describes plugin and gives WinObjEx64 ability to start/stop it execution.

```
typedef struct _WINOBJEX_PLUGIN {
    BOOLEAN NeedAdmin;
    BOOLEAN NeedDriver;
    BOOLEAN SupportWine;
    WINOBJEX_PLUGIN_STATE State;
    WORD MajorVersion;
    WORD MinorVersion;
    WCHAR Description[64];
    pfnStartPlugin StartPlugin;
    pfnStopPlugin StopPlugin;
    pfnStateChangeCallback StateChangeCallback;
} WINOBJEX_PLUGIN, *PWINOBJEX_PLUGIN;
```

Members

NeedAdmin – a BOOLEAN flag. Plugin must set it to TRUE if it require administrator rights to execute;

NeedDriver – a BOOLEAN flag. Plugin must set it to TRUE if it will require helper driver usage;

SupportWine – a BOOLEAN flag. Plugin must set it to true if it can be used with Wine/WineStaging;

State – WINOBJEX_PLUGIN_STATE enumeration describing current plugin state.

```
typedef enum _WINOBJEX_PLUGIN_STATE {
    PluginInitialization = 0,
    PluginStopped = 1,
    PluginRunning = 2,
    PluginError = 3,
    MaxPluginState
} WINOBJEX_PLUGIN_STATE;
```

By default at *PluginInit* this member must be set to *PluginInitialization* (0). Note that *MaxPluginState* is unused.

MajorVersion – Major version field, plugin self defined;

MinorVersion – Minor version field, plugin self defined;

Description – is a wide char array with maximum size of 64 elements used to keep plugin human readable description. The following description is used to identify plugin in WinObjEx64 plugins menu. Use brief description if possible;

StartPlugin – is a pointer to callback routine used by WinObjEx64 to initiate actual plugin work. This field must be set by plugin during *PluginInit* execution.

Prototype defined as following

```
NTSTATUS CALLBACK StartPlugin(
    _In_ PWINOBJEX_PARAM_BLOCK ParamBlock
);
```

The *ParamBlock* is a pointer to WINOBJEX_PARAM_BLOCK structure that will be passed from WinObjEx64 to plugin. Detailed description below;

StopPlugin – is a pointer to callback routine used by WinObjEx64 to initiate plugin shutdown. This field must be set by plugin during *PluginInit* execution.

Prototype defined as following

```
void CALLBACK StopPlugin(
    VOID
);
```

This routine has no parameters;

StateChangeCallback – is a pointer to WinObjEx64 routine that is used to access/modify plugin *State* field. Filled by WinObjEx64, plugins must not modify it.

Remarks

It is advised to make a plugin global variable that reference plugin data during *PluginInit*.

Starting a plugin

WinObjEx64 starts plugins by calling ***StartPlugin*** routine which is set by plugin during *PluginInit* in *WINOBJEX_PLUGIN* structure.

Prototype defined as following

```
NTSTATUS CALLBACK StartPlugin(
    _In_ PWINOBJEX_PARAM_BLOCK ParamBlock
);
```

Parameters

ParamBlock – input parameter, a pointer to *WINOBJEX_PARAM_BLOCK* structure filled by WinObjEx64. Contain pointers to various WinObjEx64 helper routines. Note that structure maybe expanded in the future (growing from tail), for recent version see plugin_def.h in WinObjEx64 Plugins code directory.

```
typedef struct _WINOBJEX_PARAM_BLOCK {
    HWND ParentWindow;
    HINSTANCE hInstance;
    ULONG_PTR SystemRangeStart;
    RTL_OSVERSIONINFOW osver;

    pfnReadSystemMemoryEx ReadSystemMemoryEx;
    pfnGetInstructionLength GetInstructionLength;
    pfnGetSystemInfoEx GetSystemInfoEx;
    pfnFindModuleEntryByName FindModuleEntryByName;
    pfnFindModuleEntryByAddress FindModuleEntryByAddress;
    pfnFindModuleNameByAddress FindModuleNameByAddress;
    pfnGetWin32FileName GetWin32FileName;

    pfnuiGetMaxOfTwoU64FromHex uiGetMaxOfTwoU64FromHex;
    pfnuiGetMaxCompareTwoFixedStrings uiGetMaxCompareTwoFixedStrings;
    pfnuiCopyTreeListSubItemValue uiCopyTreeListSubItemValue;
    pfnuiCopyListViewSubItemValue uiCopyListViewSubItemValue;
    pfnuiShowFileProperties uiShowFileProperties;
    pfnuiGetDPIValue uiGetDPIValue;

} WINOBJEX_PARAM_BLOCK, *PWINOBJEX_PARAM_BLOCK;
```

Members

ParentWindow – is a handle of WinObjEx64 main window;

hInstance – is a handle of WinObjEx64 instance;

SystemRangeStart – is a value describing lower possible system start address;

osver – is a RTL_OSVERSIONINFOW structure filled by WinObjEx64 by calling *RtlGetVersion* ntdll function;

ReadSystemMemoryEx – pointer to WinObjEx64 function used to read kernel memory;
GetInstructionLength – pointer to WinObjEx64 length disassembler wrapper used to determinate instruction length with HDE;

GetSystemInfoEx – pointer to WinObjEx64 wrapper around ntdll *NtQuerySystemInformation* routine;

FindModuleEntryByName – pointer to WinObjEx64 function used to locate module in kernel module list by it name, returns pointer to ntdll RTL_PROCESS_MODULE_INFORMATION structure or NULL in case of error;

FindModuleEntryByAddress – pointer to WinObjEx64 function used to locate module in kernel module list by it address, return index of element in list or -1 in case of error;

FindModuleNameByAddress – pointer to WinObjEx64 function used to locate module name in kernel module list by address, return TRUE on success, sets *Buffer* output parameter to located module name;

GetWin32FileName – pointer to WinObjEx64 routine used to query filename by handle;

uiGetMaxOfTwoU64FromHex – pointer to WinObjEx64 function used in ListView control comparison handlers;

uiGetMaxCompareTwoFixedStrings – pointer to WinObjEx64 function in ListView control comparison handlers;

uiCopyTreeListSubItemValue – pointer to WinObjEx64 function used to copy TreeList subitem value;

uiCopyListViewSubItemValue – pointer to WinObjEx64 function used to copy ListView subitem value;

uiShowFileProperties – pointer to WinObjEx64 function used to spawn shell defined file properties dialog;

uiGetDPIValue – pointer to WinObjEx64 function used to determine current DPI value (system wide or for associated window).

Remarks

If the selected plugin reports in it *State* field that it is already running (*State* is set to *PluginRunning*) then WinObjEx64 will ask user either to restart plugin or leave it as is. In case if user want to restart plugin, WinObjEx64 will first try to stop plugin and then start it again.

Stopping a plugin

WinObjEx64 stops plugin by calling ***StopPlugin*** routine which is set by plugin during *PluginInit* in *WINOBJEX_PLUGIN* structure.

Prototype defined as following

```
void CALLBACK StopPlugin(
   VOID
);
```

This routine has no parameters;

Remarks

Upon successful stop plugin must set state to *PluginStopped* by calling *StateChangeCallback* routine of WINOBJEX_PLUGIN.

Plugin parameters block (WINOBJEX_PARAM_BLOCK)

This parameters block is filled by WinObjEx64. Below is a prototypes of functions within it.

```
BOOL CALLBACK ReadSystemMemoryEx(
    _In_ ULONG_PTR Address,
    _Inout_ PVOID Buffer,
    _In_ ULONG BufferSize,
    _Out_opt_ PULONG NumberOfBytesRead);
```

Read kernel memory to the preallocated buffer.

Parameters

Address – kernel mode address to read;

Buffer – pointer to plugin allocated buffer to receive data;

BufferSize – size of buffer to receive data;

NumberOfBytesRead – optional, return actual number of bytes read upon successful execution.

Return Value

Return TRUE on success, FALSE on failure.

Remarks

Debug privilege and thus administrative rights are required.

```
UCHAR CALLBACK GetInstructionLength(
    _In_ PVOID ptrCode,
    _Out_ PULONG ptrFlags);
```

Length disassembler wrapper.

Parameters

ptrCode – pointer to code;

ptrFlags – pointer to ULONG type variable to receive flags returned by disassembler engine.

Return Value

Return number of bytes describing instruction length of given code buffer.

```
PVOID GetSystemInfoEx(
    _In_ ULONG SystemInformationClass,
    _Out_opt_ PULONG ReturnLength,
    _In_ PMEMALLOCROUTINE MemAllocRoutine,
    _In_ PMEMFREEROUTINE MemFreeRoutine);
```

NTDLL NtQuerySystemInformation wrapper with custom memory allocation/deallocation.

Parameters

SystemInformationClass – information class to query;

ReturnLength – optional, return length in bytes;

MemAllocRoutine – pointer to plugin defined memory allocation routine with the following prototype

```
typedef PVOID(*PMEMALLOCROUTINE)(
    _In_ SIZE_T NumberOfBytes);
```

MemFreeRoutine – pointer to plugin defined memory deallocation routine with the following prototype

```
typedef BOOL(*PMEMFREEROUTINE)(
    _In_ PVOID Memory);
```

Return Value

Returns pointer to data allocated by *MemAllocRoutine* in case if query was successful and NULL in case of error.

```
PVOID FindModuleEntryByName(
    _In_ PVOID pModulesList,
    _In_ LPCSTR ModuleName);
```

Find module entry in kernel modules list by module name.

Parameters

pModulesList – pointer to modules list to search in;

ModuleName – name of module to lookup.

Return Value

Pointer to module entry in case of success, NULL otherwise.

```
ULONG FindModuleEntryByAddress(
    _In_ PVOID pModulesList,
    _In_ PVOID Address);
```

Find module entry in kernel modules list by module address.

Parameters

pModulesList – pointer to modules list to search in;

Address – address of module to lookup.

Return Value

On success - index of module in module list. Returns -1 if no module with such address found in modules list.

```
BOOL FindModuleNameByAddress(
    _In_ PVOID pModulesList,
    _In_ PVOID Address,
    _Inout_ LPWSTR Buffer,
    _In_ DWORD ccBuffer);
```

Find module name by given module address in kernel modules list.

Parameters

pModulesList – pointer to modules list to search in;

Address – address of module to lookup;

Buffer – address of buffer to receive module name;

ccBuffer – size of *Buffer* in chars.

Return Value

If function succeeded it returns TRUE. *Buffer* parameter receives module name. If function call was unsuccessful it returns FALSE and *Buffer* stay untouched.

```
BOOL GetWin32FileName(
    _In_ LPWSTR FileName,
    _Inout_ LPWSTR Win32FileName,
    _In_ SIZE_T ccWin32FileName);
```

Convert NT native filename to Win32 friendly filename.

Parameters

FileName – pointer to native NT filename string;

Win32FileName – buffer to receive result;

ccWin32FileName – size of *Win32FileName* buffer in chars.

Return Value

If function succeeded it returns TRUE. *Win32FileName* parameter receives conversion result. If function call was unsuccessful it returns FALSE and *Win32FileName* buffer stay untouched.

```
INT GetMaxOfTwoU64FromHex(
    _In_ HWND ListView,
    _In_ LPARAM lParam1,
    _In_ LPARAM lParam2,
    _In_ LPARAM lParamSort,
    _In_ BOOL Inverse);
```

Compares two 16 bytes long hexadecimal values converted from list view item strings.

```
INT GetMaxCompareTwoFixedStrings(
    _In_ HWND ListView,
    _In_ LPARAM lParam1,
    _In_ LPARAM lParam2,
    _In_ LPARAM lParamSort,
    _In_ BOOL Inverse);
```

Compares two strings with fixed maximum possible length (MAX_PATH).

Parameters

ListView – handle of list view control to work with;

lParam1 – index of first item to compare;

lParam2 – index of second item to compare;

lParamSort – selected column index (works as list view *lParam1* and *lParam2* subitem index);

Inverse – if TRUE then do inversed sorting.

Return Value

The comparison function return a negative value if the first item should precede the second, a positive value if the first item should follow the second, or zero if the two items are equivalent.

```
VOID CopyTreeListSubItemValue(
    _In_ HWND TreeList,
    _In_ UINT ValueIndex);
```

Copy tree list control value to the clipboard.

Parameters

TreeList – handle of tree list control to work with;

ValueIndex – index of value to copy.

Return Value

No return value.

Remarks

Empties clipboard before copy.

```
VOID CopyListViewSubItemValue(
    _In_ HWND ListView,
    _In_ UINT ValueIndex);
```

Copy list view control value to the clipboard.

Parameters

ListView – handle of list view control to work with;

ValueIndex – index of value to copy.

Return Value

No return value.

Remarks

Empties clipboard before copy.

```
VOID ShowFileProperties(
   _In_ HWND hwndDlg,
   _In_ LPWSTR lpFileName);
```

Show shell defined file properties dialog.

Parameters

hwndDlg – handle of parent window;

lpFileName – filename to view properties of.

Return Value

No return value.

```
UINT GetDPIValue(
  _In_opt_ HWND hwnd);
```

Query DPI value, system wide or for associated window.

Parameters

hwnd – optional, the window you want to get information about. If hwnd is NULL then current system wide DPI value will be returned.

Return Value

DPI value. In case of error default system DPI value (96) will be returned.

Examples

1) Example plugin located in Source/Plugins/ExamplePlugin. Implement basic plugin sceleton. Shows message box as payload.

2) Complex GUI based plugins – Source/Plugins/Sonar and Source/Plugins/ApiSetView.