# SV Manager Toolkit Reference

ABSTRACT: THE SV MANAGER TOOLKIT REFERENCE COVERS THE CLASSES AND GLOBAL FUNCTIONS THAT WILL MAKE UP A USER PROGRAMMING DLL INTERFACING WITH THE SCADA SOFTWARE.

KEY WORDS: SV2, SV32, SV MANAGER TOOLKIT, VARIABLES, DLL

## EVOLUTIONS

| Revision | Author | Action | Date | Diffusion |
|---|---|---|---|---|
| 4.20 | FrM | Change look of document + explain the .def file | | Advanced |
| 5.1 | DL | New API functions for read/write CimWay frame | 11/05/07 | |
| 6.0 | CB | Change look of document + new functions | 01/08/07 | |
| 6.1 | CB | New Wizard | 23/10/07 | |
| 6.2 | CB | Remove and Add sections | 31/10/07 | |
| 6.3 | DL | Some littles corrections | 09/11/07 | |
| 6.4 | JS | Major upgrade | 13/11/07 | |
| 6.5 | CB | Ambiguous names clarified to SV Manager Toolkit+Title number | 01/04/08 | |
| 6.6 | CB | OPC functions included | 08/04/08 | |
| | JS | | 17/04/09 | |
| 7.0 | CB | Modification for Version 10: New generator, functions and Visual Studio | 05/05/11 | |
| 7.1 | CB | Parameters and comments updated for TxtVarWrite | 17/08/12 | |
| 7.2 | JS | Presentation reworking | 12/07/2013 | |
| 7.3 | JSL | Read group and recipes | 13/11/2013 | |
| 7.4 | ED | Use of Trace function in SCADA Basic | 26/11/2014 | |
| 7.5 | ED | Improvment of GetProjectDirectory CreateRecipe | 19/02/2015 | |
| 7.6a | ED | Data connection functions | 26/09/2017 | |
| 7.6b | ED | Update Visual Studio version | 23/05/2018 | |
| 7.6c | ED | Add VarRecords | 22/08/2018 | |
| 7.6 | JS | Review | 20/10/2018 | |

**CONTENT**

# 1 Scope of this document

The scope of this document is to describe and explain functional aspects of the *SV Manager Toolkit*. It is targeted at developers willing to develop an interface to Scada Software.

# 2 SV Manager Toolkit concepts

## 2.1 Introduction

A manager is a part of Scada software. There are many managers like alarm manager, historical manager, real-time database manager or blank manager.

Specifically, the blank manager is called *SV Manager Toolkit* and enable a programmer to interface with Scada software. This method is very helpful to make links between Scada and others applications, or to have specific treatments on variables that are not managed by the Scada software.

SV Manager Toolkit loads dynamically users DLL and invoke a specific function.

## 2.2 Architecture

The SV Manager Toolkit hasits own executive task, which is a CWinThread object. (See Microsoft Documentation for more information).

SV Manager Toolkit loads dynamically users DLL and invoke the specific function svmgrExchangeInterface of this DLL. This function receives an interface (IAPIMgr) and returns another (IUsrMgr) which contains predefined methods. The implementation of these methods is the responsibility of the programmer. The SV Manager Toolkit on specific events invokes all these methods.

The user DLL dialogs with the Supervisor by the *IAPIMgr* interface, which support a set of methods.

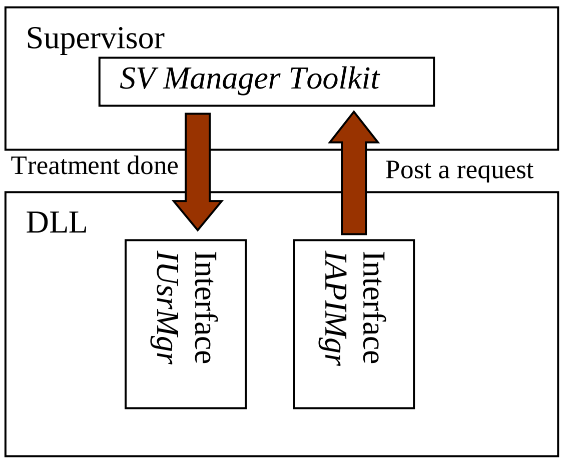
In fact, the *manager* task is a loop, waiting for treating of receipted message.

**Example**
When you use the IAPIMgr::TextVarWrite, you send a write command message to the VAR *manager* and when this command is executed the *SV Manager Toolkit* receives a positive or a negative acknowledge message, which is translated for the User DLL. This last one calls the method IUsrMgr::OnWriteCompleted.

So, you must keep in mind that a lot of functions of the SV Manager Toolkit API are asynchronous, you post a request, have an immediate response and, later, you receive the request acknowledgement.
If another message is received by the *SV Manager Toolkit*, when the User DLL treats a previous one, the received message is buffered in a FIFO and the *SV Manager Toolkit* invokes the User DLL in the appropriate IUsrMgr method when the current message treatment is done.

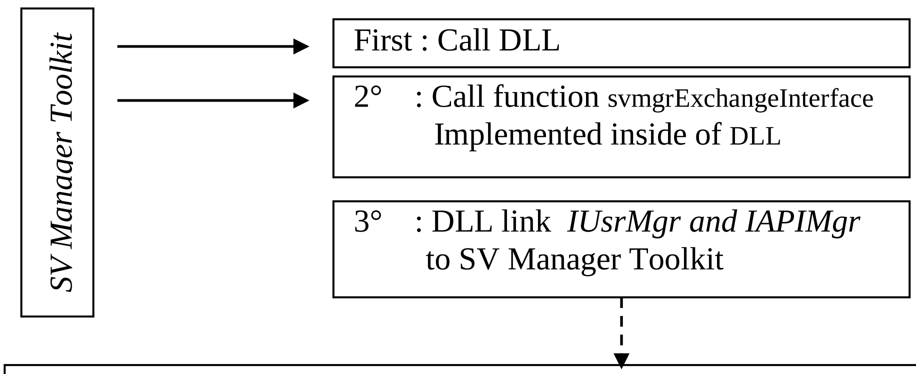
## 2.3 Basic principles

*The SV Manager Toolkit* takes advantage of Dll functionalities.
*The SV Manager Toolkit* is part of the Supervisor and it loads a Dll, which the name is specified in USRMGR.DAT file (this file is in the sub-directory BIN of the SV directory).
This Dll needs the interfaces to communicate with the Supervisor (*IUsrMgr and IAPIMgr*)



The function *svmgrExchangeInterface ( LPVOID * ppvInterface, IAPIMgr * pSvAPI)* makes possible to link the SV interfaces to the DLL interfaces:
IAPIMgr * svmgrAPI
IUsrMgr theIUsrMgr

The next diagram explains the steps:


SV Manager Toolkit
First : Call DLL
2° : Call function svmgrExchangeInterface Implemented inside of DLL
3° : DLL link IUsrMgr and IAPIMgr to SV Manager Toolkit

```
HRESULT WINAPI svmgrExchangeInterface (LPVOID * ppvInterface, IAPIMgr * pSvAPI)
{
                *ppvInterface = &theIUsrMgr;
                svmgrAPI = pSvAPI;
                 return S_OK;
}
```

Imperiously SV Manager Toolkit ⟶

Done by the project - - - - - - - - - ▶

These interfaces *IAPIMgr* and *IUsrMgr* are very important to communicate between the Supervisor and the DLL.
*IAPIMgr* defines all possible functions (further information in chapter SV Manager Toolkit API) to perform “reading of variables”, “setting of variables”, “advising variables” and many others.
*IUsrMgr* defines all possible methods (further information in chapter User DLL interface) that should answer to functions of the *IAPIMgr*, for example “OnReadCompleted” is the method called by the SV Manager Toolkit in response to the method “read variables”, there you will get the value asked; another example would be “OnDataChange”, this is the method called by the SV Manager Toolkit when a variable has changed its value in the Supervisor, but just if this variable had been advised before.

## Examples

### Advising a variable

Advising a variable is very useful to know the value of a variable in the Supervisor and to make use of this.

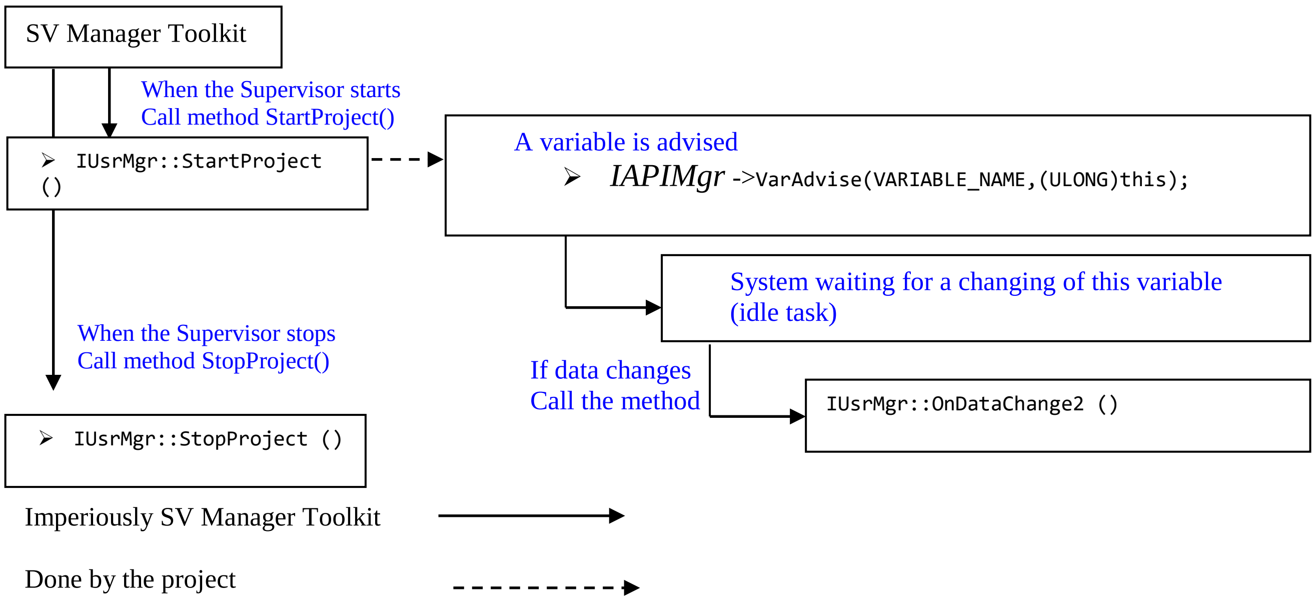
How to do that is explained below:

*IAPIMgr:*

- VarAdvise
- VarUnadvise

*IUsrMgr:*

- StartProject
- StopProject
- OnDataChange2



SV Manager Toolkit
When the Supervisor starts
Call method StartProject()
IUsrMgr::StartProject ()
A variable is advised
IAPIMgr ->VarAdvise(VARIABLE_NAME,(ULONG)this);
System waiting for a changing of this variable (idle task)
When the Supervisor stops
Call method StopProject()
If data changes
Call the method
IUsrMgr::OnDataChange2 ()
IUsrMgr::StopProject ()
Imperiously SV Manager Toolkit
Done by the project

## Writing a variable

Writing a variable is very useful to change the value of a variable in the Supervisor.

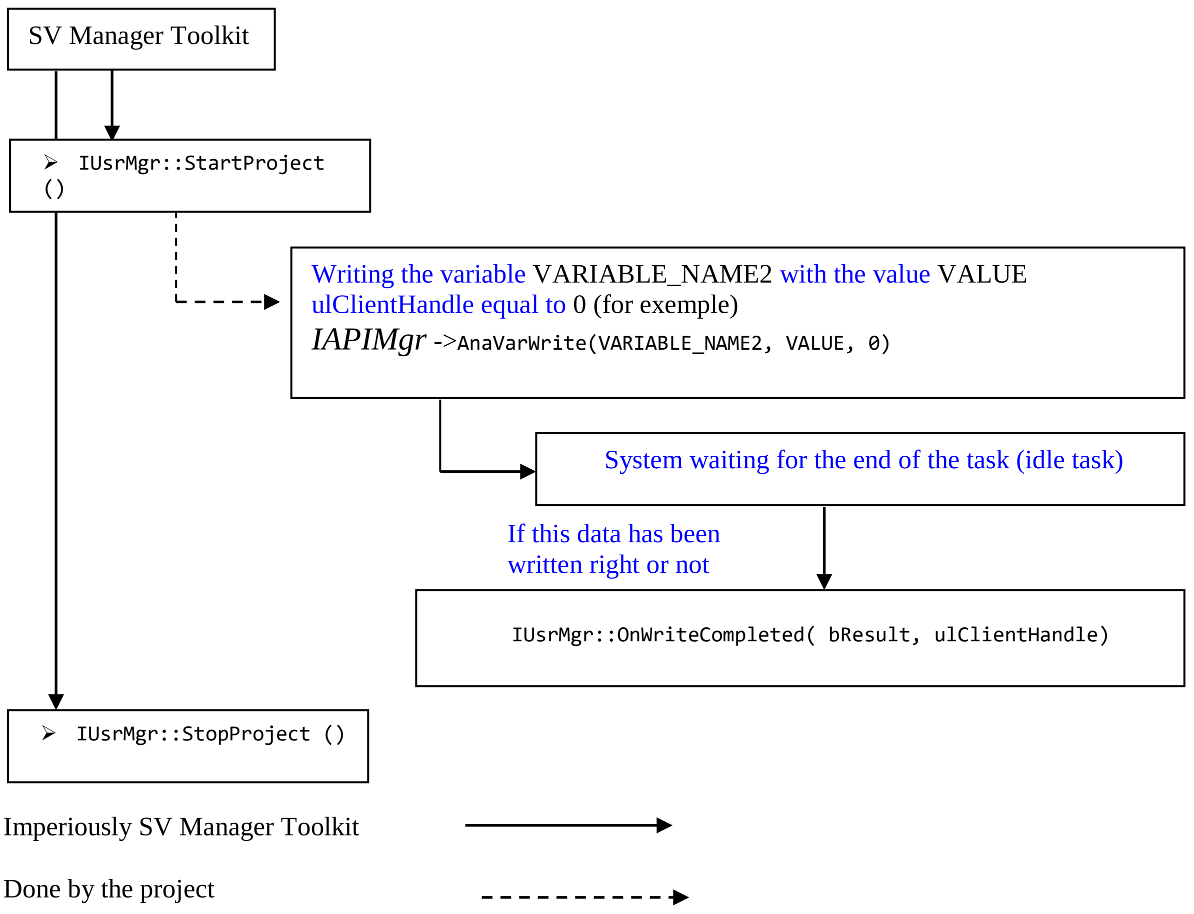
How to do that is explained below:

*IAPIMgr:*

- AnaVarWrite

*IUsrMgr:*

- StartProject
- StopProject


SV Manager Toolkit
IUsrMgr::StartProject ()
Writing the variable VARIABLE_NAME2 with the value VALUE
ulClientHandle equal to 0 (for exemple)
IAPIMgr ->AnaVarWrite(VARIABLE_NAME2, VALUE, 0)
System waiting for the end of the task (idle task)
If this data has been written right or not
IUsrMgr::OnWriteCompleted( bResult, ulClientHandle)
IUsrMgr::StopProject ()
Imperiously SV Manager Toolkit
Done by the project

** **ulClientHandle** is useful to identify what IUsrMgr method is associated with an object function of *IAPIMgr*. This is a handle (or number) to identify.
**For example** if we call AnaVarWrite(VARIABLE_NAME2, VALUE, 1) where ulClientHandle=1, the answer will be done by OnWriteCompleted( bResult, *ulClientHandle*) and you will get *ulClientHandle* being equal to 1, otherwise, the answer make reference to another call.

To make unique the **ulClientHandle,** it is usually set as (ULONG)this.
**For example** AnaVarWrite(VARIABLE_NAME2, VALUE, (ULONG)this)

# 3 Making the project

## 3.1 User DLL generation

The Microsoft Visual Studio development environment allows you to generate your own dlls.
After installing the Supervisor, in the folder “Bin” folder you will find ProGen.exe file.
This exe helps you generate your project based on template.
The template is installed in the following directory if you have installed the product with the Development Kits features:
“Development kits\Manager Toolkit\Templates\ManagerToolkitTemplate.zip“.

To generate a project you should start ProGen.exe

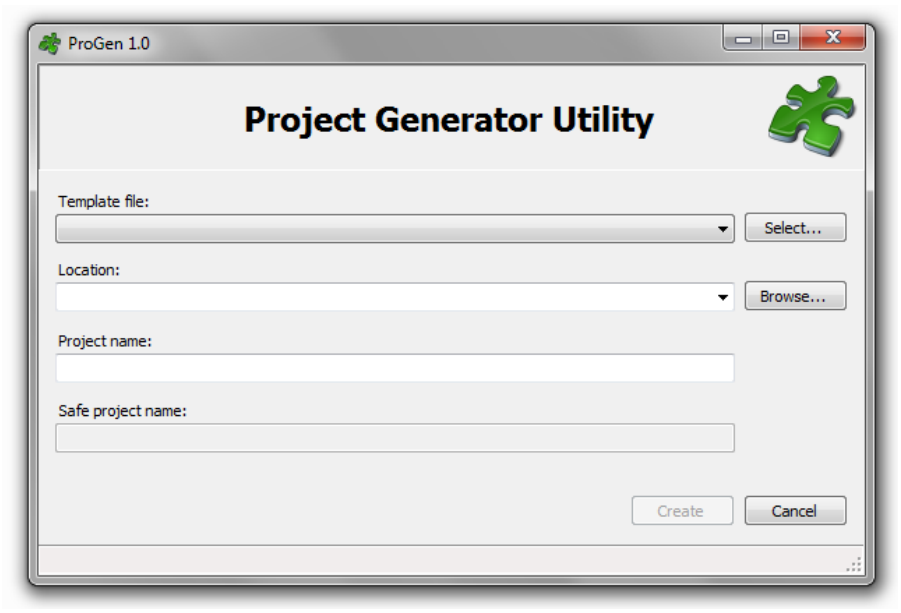
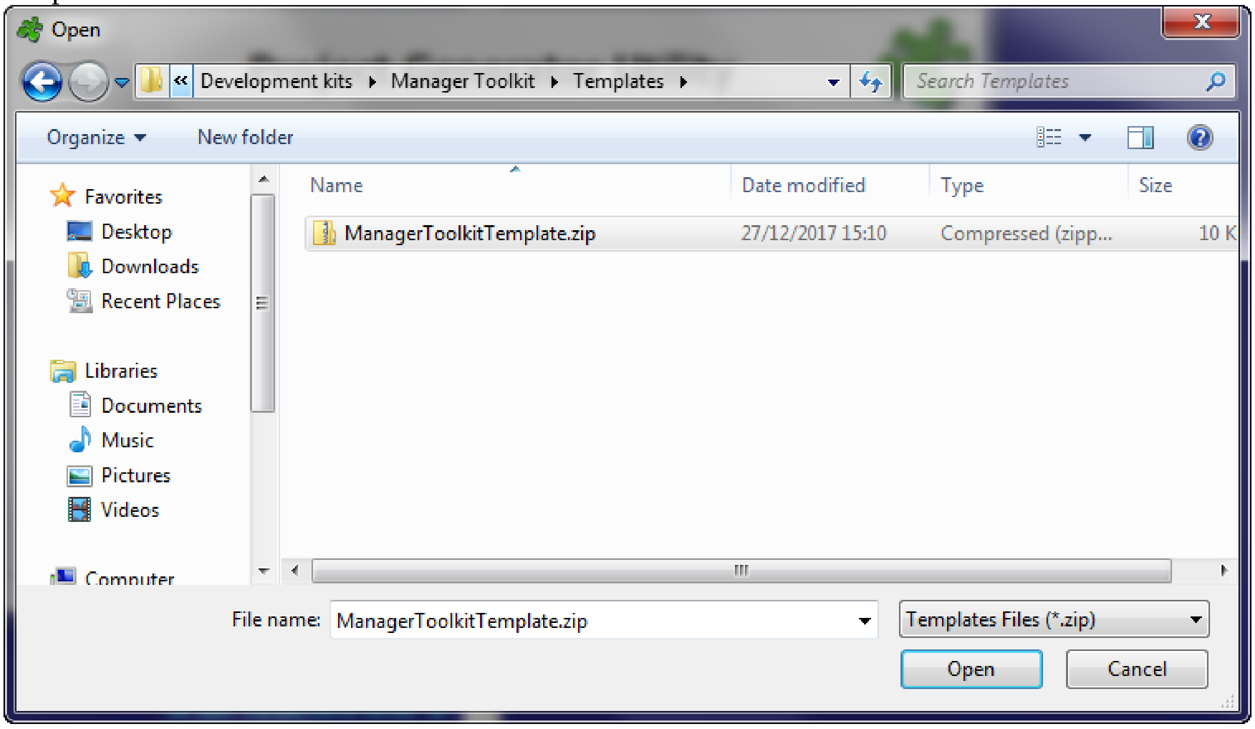


**Figure 1: ProGen.exe start window**

The next step is to select a “Template file” by clicking on “Select…”, and then selecting the template as below:



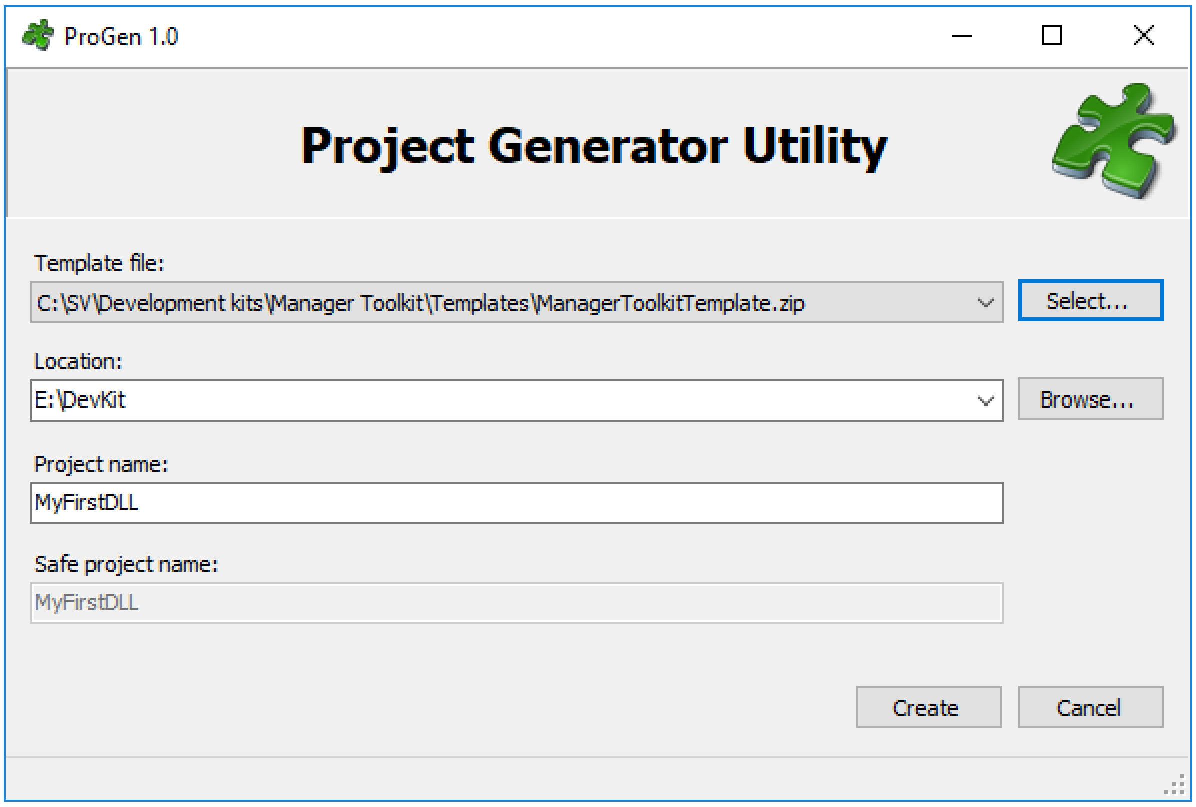
| Name | Date modified | Type | Size |
|---|---|---|---|
| ManagerToolkitTemplate.zip | 27/12/2017 15:10 | Compressed (zipp... | 10 K |

File name: ManagerToolkitTemplate.zip

Templates Files (*.zip)

The next step is to select a “Location” for your project by clicking on “Browse…”.

A requirement is that the solution must include the files svmgrAPI2.h and svmgrDefaultImpl.h. You can find these files in the folder “…\Development kits\Manager Toolkit\Include”.



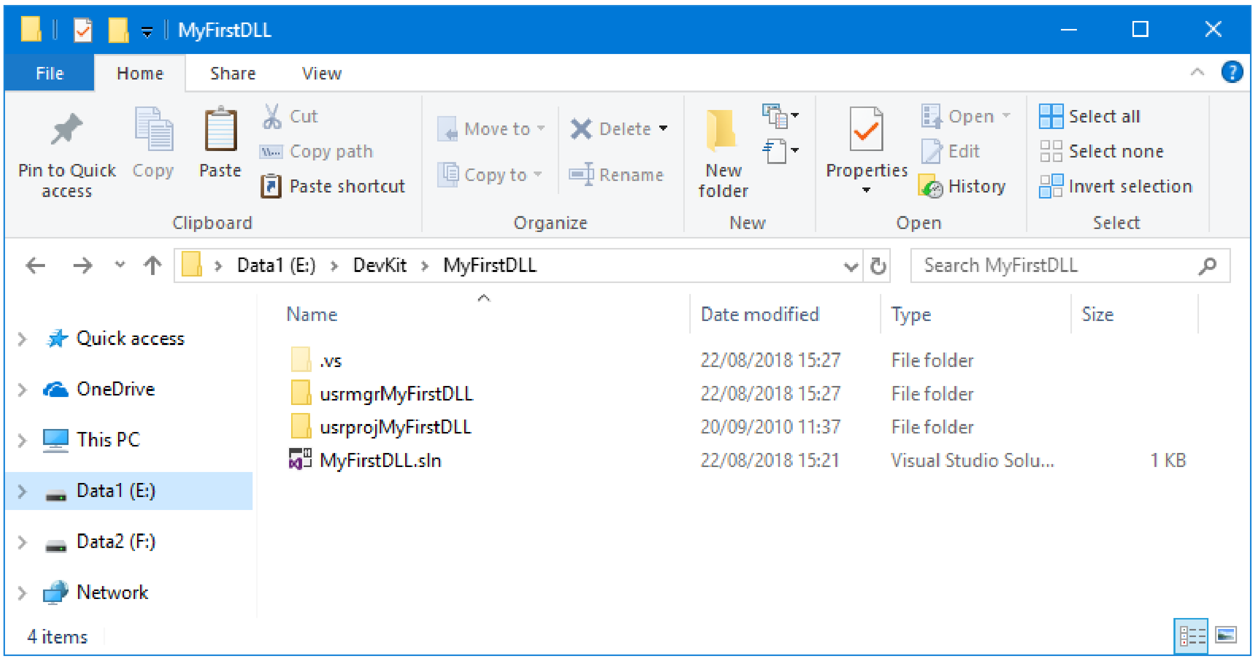
Template file: C:\SV\Development kits\Manager Toolkit\Templates\ManagerToolkitTemplate.zip

Location: E:\DevKit

Project name: MyFirstDLL

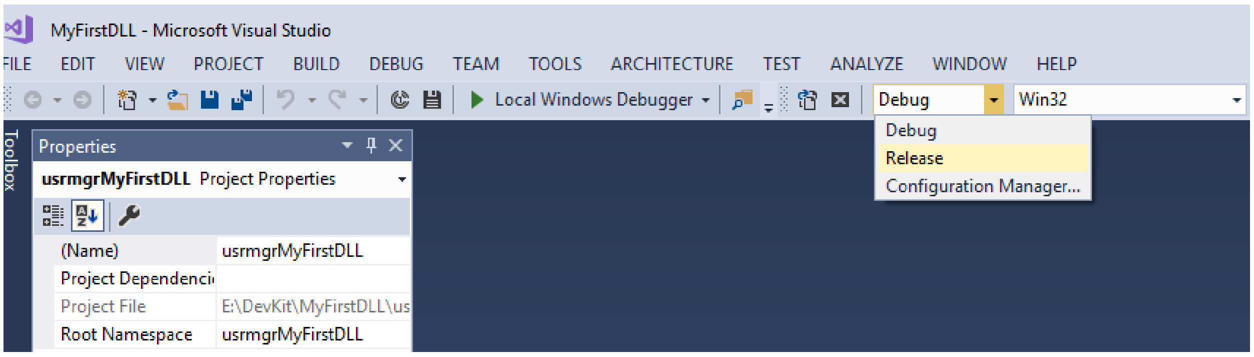
Safe project name: MyFirstDLL

After clicking on the button “Create”, a new folder and files are generated in the location selected before:



| Name | Date modified | Type | Size |
|---|---|---|---|
| .vs | 22/08/2018 15:27 | File folder | |
| usrmgrMyFirstDLL | 22/08/2018 15:27 | File folder | |
| usrprojMyFirstDLL | 20/09/2010 11:37 | File folder | |
| MyFirstDLL.sln | 22/08/2018 15:21 | Visual Studio Solu... | 1 KB |

Clicking on MyFirstDLL.sln, Visual Studio will open the project with all concerned files. You must be careful of selecting the *Release* mode when compiling. Thus the Supervisor will be able to link this DLL at start up, once you copy it in the bin folder.

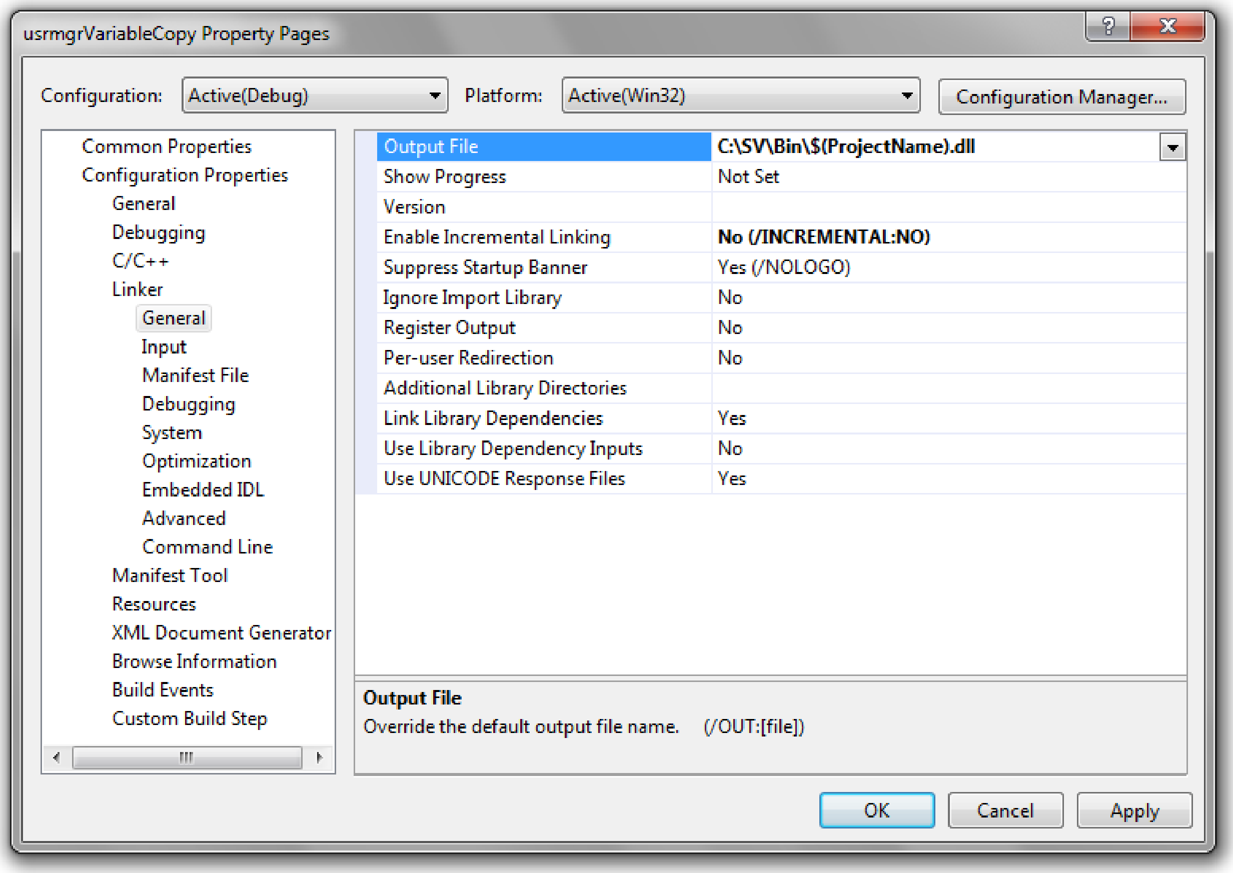


| | |
|---|---|
| (Name) | usrmgrMyFirstDLL |
| Project Dependenci | |
| Project File | E:\DevKit\MyFirstDLL\us |
| Root Namespace | usrmgrMyFirstDLL |

Is essential to copy the DLL generated (*usrmgrMyFirstDLL.dll*) and the file *usrmgr.dat* in the folder *SV\bin. This usrmgr.dat file* includes information necessary to the Supervisor for loading the DLL.

If debugging is necessary, some modifications should be done in the project properties before compiling:

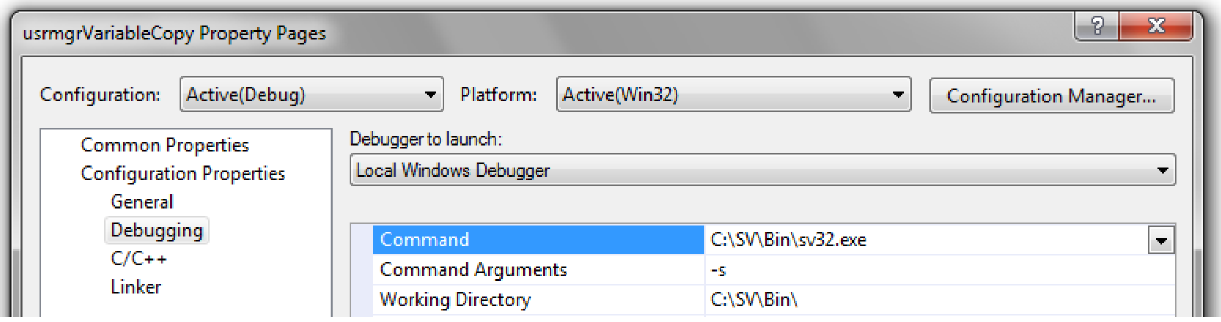
- **“Output File” property** must be modified so that the DLL generated is copied into the Supervisor Bin folder.



| | |
|---|---|
| Output File | C:\SV\Bin\$(ProjectName).dll |
| Show Progress | Not Set |
| Version | |
| Enable Incremental Linking | No (/INCREMENTAL:NO) |
| Suppress Startup Banner | Yes (/NOLOGO) |
| Ignore Import Library | No |
| Register Output | No |
| Per-user Redirection | No |
| Additional Library Directories | |
| Link Library Dependencies | Yes |
| Use Library Dependency Inputs | No |
| Use UNICODE Response Files | Yes |

**Output File**
Override the default output file name.   (/OUT:[file])

- **“Command”, “Command Arguments”** and **“Working Directory” properties** must be modified as well so that the DLL can be debug using the Supervisor.



Debugger to launch: Local Windows Debugger

| | |
|---|---|
| Command | C:\SV\Bin\sv32.exe |
| Command Arguments | -s |
| Working Directory | C:\SV\Bin\ |

Once these modifications have been done, you will be able to debug your project.

## 3.2 Inserting a new method

In the usrmgr[ProjectName]Impl.cpp a structure called IUsrMgr is generated by default. In this structure will be implemented all functions to interact with the Supervisor.
All possible methods to be called from the Supervisor are defined in svmgrDefaultImpl.h and svmgrAPI2.h. Concerning the first file, it defines all the possible callbacks. Any answer from the Supervisor is done by callbacks because of asynchronous behaviour of the system.

For example, to add the OnWriteCompleted method (this method is called by the Supervisor when writing a variable is completed) in your project should be necessary to do:

In the file usrmgr[ProjectName]Impl.cpp and inside of the `IUsrMgr` structure you should insert the next function:

```
void __stdcall OnWriteCompleted(BOOL bResult, ULONG ulClientHandle)
```

An example of the final structure is:

```
struct IUsrMgr : public ISVMgr
{

        void __stdcall  StartProject()
                                { …    }

        void __stdcall  StopProject()
                                { … }

        void __stdcall OnWriteCompleted(BOOL bResult, ULONG ulClientHandle)
                                { … }

};
```

For any further function defined in svmgrDefaultImpl.h is necessary to follow the same procedure.

## 3.3 User DLL selection

SV Manager Toolkit has to know the names of the Users DLL. Theses names are configured in the file USRMGR.DAT in the BIN sub-directory of the SV directory.

The USRMGR.DAT file is a text file that you have to create with a text editor (for example NOTEPAD).

You have to use the following syntax.

```
[USRMGR\server1]
DLL=UserName1

[USRMGR\server2]
DLL=UserName2
```

Where *UserName1* and *UserName2* are the names of the Users DLL that will be loaded by SV Manager Toolkit instance, note that server1 and server2 are free names.

For example, if you want to load svmgr1.dll and svmgr2.dll, you have to write the following lines:

```
[USRMGR\server1]
DLL=svmgr1.dll

[USRMGR\server2]
DLL=svmgr2.dll
```

**Note:**
You will be able to add up to eight User DLLs in the USRMGR.DAT file.
The Users DLL must be in the BIN sub-directory of the SV directory.

# 4 User DLL interface

The User DLL interface is a pointer on a specific structure called IUsrMgr. The IUsrMgr structure contains methods that you will be able to implement.

The SV Manager Toolkit invokes each method when specific events occur. Even if multi-threading is implemented in the User DLL, all these methods are called on the main thread context of the User DLL, which is the CWinThread context of the SV Manager Toolkit.

There is a specific function exported by the user DLL to exchange interfaces pointers. It's the svmgrExchangeInterface function.

## 4.1 Methods and functions scheduling at start / stop time

There are some IUsrMgr methods invoked only once by the SV Manager Toolkit. These methods are invoked only when start/stop events occur.

At Start time :

1. svmgrExchangeInterface
2. IUsrMgr::GetApiVersion
3. IUsrMgr::InitInstance
4. IUsrMgr::StaticInit
5. IUsrMgr::StartProject

And, at Stop time :

1. IUsrMgr::StopProject
2. IUsrMgr::DelProject
3. IUsrMgr::StaticEnd
4. IUsrMgr::ExitInstance

All these methods and functions are described in the next pages.

## 4.2 svmgrExchangeInterface

```
HRESULT WINAPI svmgrExchangeInterface (
        LPVOID *        ppvInterface,
        IAPIMgr *       pSvAPI
        );
```

### Description

Exchange the User DLL and SV Manager Toolkit interface pointer.
The User DLL invokes SV Manager Toolkit functions with the pointer pSvAPI.
The SV Manager Toolkit uses the pointer ppvInterface to access the User DLL when specific events occur.
If this method returns value other then S_OK, the User DLL will never be used by the SV Manager Toolkit.

This method is called once just after the User DLL is loaded by the SV Manager Toolkit.

### Parameters

*ppvInterface* [out]
    Address of a variable that receives the User DLL interface pointer

*pSvAPI* [in]
    Address of the SV Manager Toolkit interface

### Returns

| | |
|---|---|
| S_OK | The interface pointer was successfully retrieved and the SV Manager Toolkit interface pointer was successfully transmitted to the User DLL.<br>The ppvInterface parameter contains the User DLL interface pointer.<br>The pSvAPI parameter contains the SV Manager Toolkit interface pointer. |
| | No interface pointer was retrieved. The SV Manager Toolkit will never use the DLL. |

## Example

This is the example **usrmgr[ProjectName]Impl.cpp** and:

First,  define IUSRMGR interface and function to implement.
Second, define API interface.
Third,  implement svmgrExchangeInterface

```
// FIRST -------------------------- IUsrMgr interface -------------------------------------------

struct IUsrMgr : public ISVMgr
{
       // Implemented methods inside IUsrMgr
       void __stdcall  StartProject()                          { StartAdvise();       }
       void __stdcall  StopProject()                           { StopAdvise();        }
       void __stdcall  OnDataChange (      char *Buffer)  { ::OnDataChange( Buffer);  }

} ;

// The one and only IUsrMgr object instance

IUsrMgr theIUsrMgr;

////////////////////////////////////////////////////////////////////////////////
// Interface pointer to the SV Manager Toolkit API

IAPIMgr * svmgrAPI = NULL;
// Exchanges the User DLL and SV Manager Toolkit interface pointers

HRESULT WINAPI svmgrExchangeInterface (LPVOID * ppvInterface, IAPIMgr * pSvAPI)
{
    *ppvInterface = &theIUsrMgr;
    svmgrAPI = pSvAPI;
    return S_OK;
}
```

## 4.3 svmgrGetInterface

**HRESULT WINAPI svmgrGetInterface(**
    **LPVOID *ppvInterface**
    **) ;**

### Description

Retrieve the User DLL interface pointer. The SV Manager Toolkit uses the pointer to access the User DLL when specific events occur. If this method returns value other then S_OK, the User DLL will never be use by the SV Manager Toolkit.

### Parameters

This method is called once just after the User DLL is loaded by the SV Manager Toolkit.

*ppvInterface* [out]
    Address of a variable that receives the interface pointer

### Returns

| | |
|---|---|
| S_OK | The interface pointer was successfully retrieved. The ppvInterface parameter contains the interface pointer. |
| Others | No interface pointer was retrieved. The SV Manager Toolkit will never use the DLL. |

### Example

```
// ------------- svmgrGetInterface -------------

IUsrMgr theIUsrMgr;

HRESULT WINAPI svmgrGetInterface (LPVOID * ppvInterface)
{
	ASSERT(ppvInterface);
	*ppvInterface = &theIUsrMgr;
	return S_OK;
}
```

## 4.4 IUsrMgr::DelProject

```
void IUsrMgr::DelProject(
        ) ;
```

### Description

DelProject is an SV event. All SV Manager Toolkit instances receive this event. It signals that the SV project is being deleting.

In this method, you deallocate memory or free resources using by tasks stopping in IUsrMgr::StopProject. The resources that are freed in this method are project dependent. Prefer the method StaticEnd to free general-purpose resources.

### Returns

None

## 4.5 IUsrMgr::ExitInstance

```
void IUsrMgr::ExitInstance(
        ) ;
```

### Description

The SV Manager Toolkit derived method ExitInstance of CWinThread calls this method and just before unloading the user DLL.

The ExitInstance method is when exiting this instance of the thread, or if a call to InitInstance fails.

For more details, see in Microsoft Documentation CWinThread::ExitInstance.

### Returns

None

## 4.6 IUsrMgr::GetApiVersion

**DWORD IUsrMgr::GetApiVersion(**
**);**

### Description

Retrieves the SV Manager Toolkit API Version use to link the User DLL. The SV Manager Toolkit uses the version to maintain compatibility between the Supervisor and the User DLL.

This method is called once just after the SV Manager Toolkit retrieves the interface pointer.

To be sure that GetApiVersion returns the correct value for the version, use the global define of the SV Manager Toolkit API called SVMGR_API_VERSION.

### Returns

| | |
|---|---|
| S_OK | The SV Manager Toolkit API version number |

### Example

This function is very important to know the version and compatibility. It is defined always in file *svmgrBaseIntf.cpp*

```
//------------------------------ GetApiVersion ----------------------------------------

DWORD IUsrMgr::GetApiVersion ()
{
return SVMGR_API_VERSION; // Defined in SVMGRAPI2.h
}
```

## 4.7 IUsrMgr::InitInstance

**void IUsrMgr::InitInstance(**
**) ;**

### Description

The SV Manager Toolkit derived method InitInstance of CWinThread calls this method.

This method is used to initialize the new instance of a user-interface thread. Typically, in InitInstance, you perform tasks that must be completed when a thread is first created.

For more details, see in Microsoft Documentation CWinThread::InitInstance.

### Returns

None

## 4.8 IUsrMgr::OnAckAlarmCompleted

```
void IUsrMgr::OnAckAlarmCompleted(
        BOOL            bResult,
        ULONG           ulClientHandle
        );
```

### Description

The SV Manager Toolkit signals that a pending acknowledge command is completed.

### Parameters

*bResult* [in]

Indicates if the acknowledge command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]

Contains the client handle specified by AckAlarm function.

### Returns

None

## 4.9 IUsrMgr::OnAddOPCGroupCompleted

**void IUsrMgr::OnAddOPCGroupCompleted (**
```
        BOOL        bResult,
        ULONG       ulClientHandle,
        ULONG       ulGroupServerHandle,
        HRESULT     hrErrorCode
        );
```

### Description
The SV Manager Toolkit signals that a pending acknowledge command is completed.

### Parameters
*bResult* [out]

Indicates if the acknowledge command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [out]

Contain the client handle specified by AddOPCGroup function.

.

*ulGroupServerHandle* [out]

Handle given by function OnAddOPCGroupCompleted.

*hrErrorCode* [out]

Error associated to COM link. (Value 0 is ok, otherwise there is COM error)

### Returns
None

### Example:
```
void OnAddOPCGroupCompleted(  BOOL bResult,       ULONG ulClientHandle,       ULONG
ulGroupServerHandle,
                                HRESULT hrErrorCode)
{
        LogMessage("Group creation %s (hr=0x%08X, groupHandle=0x%08X)",  besult?"OK":"failed",
                                                                         hrErrorCode,
                                                                         ulGroupServerHandle);

        g_ulGroupServerHandle = ulGroupServerHandle;// Saving the Group handle into a global
variable
                                                    // it will be necessary to other functions

        if(bResult == TRUE)
        {
                if(g_pSvAPI->SubscribeOPCItem(g_ulGroupServerHandle, "Dev.Tag1", 1234) == FALSE)
                {
                        LogMessage("Unable to subscribe");
                }
        }
}
```

## 4.10 IUsrMgr::OnDataChange

**void IUsrMgr::OnDataChange(**
    **char *pBuffer**
    **) ;**

### Description

The SV Manager Toolkit signals its data changes for advised variables. The pBuffer parameter contains the data changes.
This method is obsolete. Please use IUsrMgr::OnDataChange2

### Parameters

*pBuffer* [in]
    This buffer is called data stream. This data stream is composed of 3 parts: First one, the stream header, then, the variable header and finally the values for these variables. So, we can represent the stream as:

| | | |
|---|---|---|
| _svmgrStreamHeader | _svmgrStreamVarHeader[] | _svmgrVarValue[] |

For more details about the data stream, see documentation chapter Data Stream.

### Returns

None

### Example

Assuming CAdvisedVar as the object advised and related to a change of a variable, you will be able to get it through splitting the buffer as is shown in the next function:

```
void OnDataChange( char *Buffer)
{
         _svmgrStreamHeader *pHeader =  ( _svmgrStreamHeader *)  Buffer;

        _svmgrStreamVarHeader * pVarHeader  =  ( _svmgrStreamVarHeader *)( Buffer +
                   sizeof( _svmgrStreamHeader));

        _svmgrVarValue * pValue =  (_svmgrVarValue *)( Buffer + sizeof( _svmgrStreamHeader) +
                   sizeof(_svmgrStreamVarHeader) );

        for ( DWORD dwCmpt = 0; dwCmpt < pHeader->dwVarCount; dwCmpt++)
        {
               //Here you get the pointer to the object CAdvisedVar
               CAdvisedVar *pVar = ( CAdvisedVar *)pVarHeader[ dwCmpt].ulClientHandle;

               if (pVar->VarToChange()!="") // Do nothing if the variable name to write is
        empty
                      g_pSvAPI->AnaVarWrite(pVar->VarToChange(),pValue->dAna(),0);
               //Function writing VARS.V1 to VARS.V2
        }
}
```

## 4.11 IUsrMgr::OnDataChange2

```
void IUsrMgr::OnDataChange2(
        DWORD                   dwCount,
        ULONG*                  pulClientHandles,
        BOOL*                   pbResults,
        _svmgrVarValue2**       pValues,
        FILETIME*               pftTimestamps,
        _svmgrVarStatus*        pStatus
        ) ;
```

### Description

The SV Manager Toolkit signals its data changes for advised variables.

### Parameters

*dwCount* [in]

Contains the number of changes

*pulClientHandles* [in]

Array of dimension *dwCount*.
Contains the client handles specifying on advise request (see function IAPIMgr::VarAdvise).

*pbResults* [in]

Array of dimension *dwCount*.
If  FALSE, value, timestamp and status are not consistent.

*pValues* [in]

Array of dimension *dwCount*.
Contains the type and value of the variable.

*pftTimestamps* [in]

Array of dimension *dwCount*.
Contains the time stamp of the value if the *status* is good or the time when the value quality become bad if *status* is bad.

*pStatus* [in]

Array of dimension *dwCount*.
Contains the status of the variable.

### Returns

| | |
|---|---|
| TRUE | OnDataChange2 is managed by the user DLL. |
| FALSE | OnDataChange2 isn’t managed by the user DLL; the SV Manager Toolkit will now call OnDataChange method. |

## Example

```
BOOL OnDataChange2(      DWORD                      dwCount,
                         ULONG                  *   pulClientHandle,
                         BOOL                   *   pbResults,
                         _svmgrVarValue2        **  pValues,
                         FILETIME               *   pftTimestamps,
                         _svmgrVarStatus        *   pStatus)
{
       for ( DWORD dwCmpt = 0; dwCmpt < dwCount; dwCmpt++)
       {
              if ( pbResult[dwCmpt] == FALSE)
                     continue;

              switch ( pValue->vt)
              {
              case svmgr_vtLOG:
                     // Logic value is pValues[dwCmpt]->bLog()
                     break;
              case svmgr_vtALARM:
                     // Alarm state is pValues[dwCmpt]-> bAlarm()
                     // Alarm acknowledgment is pValues[dwCmpt]-> bAlarmAck()
                     break;
              case svmgr_vtANA:
                     // Analogic value is pValues[dwCmpt]->dAna()
                     break;
              case svmgr_vtTXT:
                     // Text value is pValues[dwCmpt]->szTxt()
                     break;
              }
       }

       return TRUE;
}
```

## 4.12 IUsrMgr::OnExtentedAttributesChange

```
void OnExtentedAttributesChange(
        ULONG                              ulClientHandle,
        WORD                               wNbExtStringAttributes,
        _svmgrExtStringAttributeIds *      peExtStringAttributeIds,
        LPSTR *                            pszExtStringAttributeValues,
        WORD                               wNbExtBinaryAttributes,
        _svmgrExtBinaryAttributeIds *      peExtBinaryAttributeIds,
        _svmgrExtBinaryAttributeValue *    puExtBinaryAttributeValues
        );
```

### Description
The SV Manager Toolkit signals extended attribute changes for advised variables.

### Parameters
*ulClientHandle* [in]
    Contains the client handle specified by VarAdvise function.

*wNbExtStringAttributes* [in]
    Number of extended string attributes that change. This is the size of the 2 arrays *peExtStringAttributeIds* and *pszExtStringAttributeValues*.

*peExtStringAttributeIds* [in]
    Array containing the id of the extended string attributes that change.

*pszExtStringAttributeValues* [in]
    Array containing the value of the extended string attributes that change.

*wNbExtBinaryAttributes* [in]
    Number of extended string attributes that change. This is the size of the 2 arrays *peExtBinaryAttributeIds* and *pszExtBinaryAttributeValues*.

*peExtBinaryAttributeIds* [in]
    Array containing the id of the extended binary attributes that change.

*puExtBinaryAttributeValues* [in]
    Array containing the value of the extended binary attributes that change.

### Returns
None

### See Also
IAPIMgr::InitExtendedAttributesStructure, IAPIMgr::FreeExtendedAttributesStructure, IAPIMgr::GetExtendedAttributes, IAPIMgr::SetStringExtendedAttribute, IAPIMgr::SetBinaryExtendedAttribute, IAPIMgr::VarAdvise

## 4.13 IUsrMgr::OnMaskVarCompleted

```
void IUsrMgr::OnMaskVarCompleted(
        BOOL            bResult,
        ULONG           ulClientHandle
        );
```

### Description

The SV Manager Toolkit signals that pending mask command is completed.

### Parameters

*bResult* [in]

Indicates if the mask command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]

Contains the client handle specified by MaskVar function.

### Returns

None

## 4.14 IUsrMgr::OnModifyEqtAddressCompleted

```
void IUsrMgr::OnModifyEqtAddressCompleted (
        BOOL        bResult,
        ULONG       ulClientHandle,
        USHORT      usErrorCode
        );
```

### Description
The SV Manager Toolkit signals that pending modification address command is completed.

### Parameters
*bResult* [in]

Indicates if the modification command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]

Contains the client handle specified by CimwayModifyEqtAddress function.

*usErrorCode* [in]

Equal 0 if the command has succeeded.

### Returns
None

## 4.15 IUsrMgr::OnModifyFrameAddressCompleted

```
void IUsrMgr::OnModifyFrameAddressCompleted (
        BOOL            bResult,
        ULONG           ulClientHandle,
        USHORT          usErrorCode
        );
```

### Description
The SV Manager Toolkit signals that pending modification address command is completed.

### Parameters
*bResult* [in]
    Indicates if the modification command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]
    Contains the client handle specified by CimwayModifyFrameAddress function.

*usErrorCode* [in]
    Equal 0 if the command has succeeded.

### Returns
None

## 4.16 IUsrMgr::OnNotify

```
void IUsrMgr::OnNotify(
        ULONG           ulClientHandle
        );
```

### Description
The SV Manager Toolkit signals that notification has been set.

### Parameters
*ulClientHandle* [in]

Contains the client handle specified by Notify function.

### Returns
None

## 4.17 IUsrMgr::OnOPCItemChange

**void IUsrMgr::OnOPCItemChange(**

```
        ULONG        ulClientHandle,
        VARIANT      value,
        FILETIME     ftTimeStamp,
        USHORT       usQuality,
        HRESULT      hrErrorCode
        );
```

### Description

The SV Manager Toolkit signals that a pending acknowledge command is completed.

### Parameters

*ulClientHandle* [in]

Contains the client handle specified by SubscribeOPCItem function.

*value* [in]

Value read.  Type VARIANT.

*ftTimeStamp* [in]

Variable time in FILETIME format.

*usQuality* [in]

OPC quality.

*hrErrorCode* [in]

Error associated to COM link. (Value 0 is ok, otherwise there is COM error)

### Returns

None

### Example

```
void OnOPCItemChange(ULONG ulClientHandle, VARIANT value, FILETIME ftTimeStamp,
                     USHORT usQuality, HRESULT hrErrorCode)
{
if (SUCCEEDED(hrErrorCode))
{
    switch ( V_VT(&value))
    {
     case VT_I2:
         LogMessage("OPC changed : V_I2=%d usQuality=%u hr=0x%08X", V_I2(&value),
                                usQuality, hrErrorCode);
           break;

     default:
       LogMessage("OPC changed VT=%u", V_VT(&value));
       }
   }
}
```

## 4.18 IUsrMgr::OnReadCompleted

```
void IUsrMgr::OnReadCompleted(
        char *   pBuffer
        ) ;
```

### Description

The SV Manager Toolkit signals that pending read command data are completed. The pBuffer parameter contains the values of these read variables.
This method is obsolete. Please use IUsrMgr::OnReadComplete2

### Parameters

*pBuffer* [in]

As the IUsrMgr::OnDataChange parameter, this buffer is called the data stream and has the same structure. The data stream is composed of 3 parts: First, the stream header, then, the variable header and finally the values for these variables. So, we can represent the stream as:

| | | |
|---|---|---|
| _svmgrStreamHeader | _svmgrStreamVarHeader[] | _svmgrVarValue[] |

For more details about the data stream, see documentation chapter Data Stream.

### Returns

None

## 4.19 IUsrMgr::OnReadCompleted2

```
void IUsrMgr::OnReadCompleted2(
        DWORD                       dwCount,
        ULONG                    *  pulClientHandles,
        BOOL  *                  pbResults,
        _svmgrVarValue2          ** pValues,
        FILETIME                 *  pftTimestamps,
        _svmgrVarStatus          *  pStatus
        ) ;
```

### Description

The SV Manager Toolkit signals those data changes for the advised variables are occurred.

### Parameters

*dwCount* [in]

Contains the number of changes

*pulClientHandles* [in]

Array of dimension *dwCount*.
Contains the client handle specified by VarRead function.

*pbResults* [in]

Array of dimension *dwCount*.
If FALSE, the read command has failed and value, timestamp and status are not consistent.

*pValues* [in]

Array of dimension *dwCount*.
Contains the type and value of the variable.

*pftTimestamps* [in]

Array of dimension *dwCount*.
Contains the time stamp of the value if the *status* is good or the time when the value quality become bad if *status* is bad.

*pStatus* [in]

Array of dimension *dwCount*.
Contains the status of the variable.

### Returns

| | |
|---|---|
| TRUE | OnDataChange2 is managed by the user DLL. |
| FALSE | OnDataChange2 isn't managed by the user DLL; the SV Manager Toolkit will now call OnDataChange method. |

## 4.20 IUsrMgr::OnReadFrameCompleted

```
void IUsrMgr::OnReadFrameCompleted (
        BOOL            bResult,
        ULONG           ulClientHandle,
        USHORT          usErrorCode,
        FILETIME        ftTimeStamp,
        DWORD           dwSize,
        BYTE            *pbyBuffer,
        ULONG           ulFrameStatus,
        ULONG           ulFrameComplementaryStatus
        );
```

### Description

The SV Manager Toolkit signals that pending read is completed. The parameters of the callback contain the value, the timestamp and the quality of the read of frame identified by *ulClientHandle*.

### Parameters

*bResult* [in]

Indicates if the read command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]

Contains the client handle specified by CimwayReadFrame function.

*usErrorCode* [in]

Equal 0 if the read command has succeeded.

*ftTimeStamp* [in]

Time stamp of data of the frame

*dwSize* [in]

The size in bytes of *pbyBuffer*. It is the size of the frame converted in bytes.

*pbyBuffer* [in]

Pointer of frame data. The buffer size is *dwSize* bytes. This buffer is freed after the call.

*ulFrameStatus* [in]

Frame state after the read

*ulFrameComplementaryeStatus* [in]

Frame complementary state after the read

### Returns

None

## 4.21 IUsrMgr::OnReadGroupCompleted

```
BOOL __stdcall OnReadGroupCompleted (
        DWORD                   dwCount,
        ULONG                   ulClientHandle,
        LPTSTR*                 pszVariables,
        BOOL*                   pbResults,
        _svmgrVarValue2**       pValues,
        FILETIME*               pftTimestamps,
        _svmgrVarStatus*        pStatus
        );
```

### Description

Signal from the SV Manager Toolkit that a read for a group of variables has been completed. It is triggered by completion of the activity started by IAPIMgr::ReadGroup.

### Parameters

*dwCount* [out]

Specifies the number of values read and accessible in *pValues*.

*ulClientHandle* [out]

Contains the client handle specified by ReadGroup function.

*pszVariables* [out]

Array of dimension dwCount. Contains the value of the Text variable.

*pbResults* [out]

Array of dimension dwCount. Contains the value of the Bit or Alarm variable.

*pValues* [out]

Array of dimension dwCount. Contains the value of the Register variable.

*pftTimestamps* [out]

Array of dimension dwCount. Contains the last timestamp of the variable.

*pStatus* [out]

Array of dimension dwCount. Contains the operational status of the variable.

### Returns

| | |
|---|---|
| TRUE | Return always TRUE |

## 4.22 IUsrMgr::OnReadOPCItemCompleted

**void IUsrMgr:: OnReadOPCItemCompleted (**

| | |
|---|---|
| **BOOL** | **bResult,** |
| **ULONG** | **ulClientHandle,** |
| **VARIANT** | **value,** |
| **FILETIME** | **ftTimeStamp,** |
| **USHORT** | **usQuality,** |
| **HRESULT** | **hrErrorCode** |

**);**

### Description

The SV Manager Toolkit signals that a pending acknowledge command is completed.

### Parameters

*bResult* [in]

Indicates if the acknowledge command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]

Contains the client handle specified by ReadOPCItem function.

*value* [in]

Value read.  Type VARIANT.

*ftTimeStamp* [in]

Variable time in FILETIME format.

*usQuality* [in]

OPC quality.

hrErrorCode

[in]

Error associated to COM link. (Value 0 is ok, otherwise there is COM error)

### Returns

None

### Example

```
void OnReadOPCItemCompleted(BOOL bResult, ULONG ulClientHandle, VARIANT value,
      FILETIME ftTimeStamp, USHORT usQuality, HRESULT hrErrorCode)
{
   if (SUCCEEDED(hrErrorCode))
      switch ( V_VT(&value))
      {
      case VT_I2:
                  LogMessage("Read OPC completed: V_I2=%d usQuality=%u hr=0x%08X",
      V_I2(&value), usQuality, hrErrorCode);
                  break;
      default :
                  LogMessage("Read OPC completed VT=%u", V_VT(&value));
       }
}
```

## 4.23 IUsrMgr::OnRemoveOPCGroupCompleted

```
void IUsrMgr::OnWriteOPCItemCompleted (
        BOOL            bResult,
        ULONG           ulClientHandle,
        HRESULT         hrErrorCode
        );
```

### Description

The SV Manager Toolkit signals that a pending acknowledge command is completed.

### Parameters

*bResult* [in]

Indicates if the acknowledge command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]

Contains the client handle specified by RemoveOPCGroup function.

*hrErrorCode* [in]

Error associated to COM link. (Value 0 is ok, otherwise there is COM error)

### Returns

None

## 4.24 IUsrMgr::OnSendRecipeCompleted

```
BOOL IUsrMgr::OnSendRecipeCompleted (
        BOOL            bResult,
        ULONG           ulClientHandle
        );
```

### Description
The SV Manager Toolkit signals that a send recipe is completed.

### Parameters
*bResult* [out]

The result of the recipe action. TRUE represents a successful sent recipe, FALSE represents a failed sent recipe.

*ulClientHandle* [out]

Contains the client handle specifying in the SendRecipe function.

### Returns

| | |
|---|---|
| TRUE | Return always TRUE |

### See Also
IAPIMgr::SendRecipe
IAPIMgr::CreateRecipe
IAPIMgr::AddVariableToRecipe

## 4.25 IUsrMgr::OnSetAlarmAttribute

```
void IUsrMgr::OnSetAlarmAttribute(
        BOOL      bResult,
        ULONG     ulClientHandle
        );
```

### Description
The SV Manager Toolkit signals that alarm set command is completed.

### Parameters
*bResult* [in]

Indicates if the set command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]

Contains the client handle specifying in the SetAlarmAttribute function.

### Returns
None

### See Also
IAPIMgr::SetAlarmAttribute

## 4.26 IUsrMgr::OnSetDataSetCompleted

```
void IUsrMgr::OnSetDataSetCompleted(
        BOOL        bResult,
        ULONG       ulClientHandle
        ) ;
```

### Description
The SV Manager Toolkit signals that pending data set command are completed.

### Parameters
*bResult* [in]
Indicates if the set command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]
Contains the client handle specifying in the SetDataSet function.

### Returns
None

### See Also
IAPIMgr::SetDataSet
IAPIMgr::GetDataSetMaxSize

## 4.27 IUsrMgr::OnSetExtendedBinaryAttribute

```
void IUsrMgr::OnSetExtendedBinaryAttribute(
        BOOL            bResult,
        ULONG           ulClientHandle
        );
```

### Description

The SV Manager Toolkit signals that pending set extended binary attribute command is completed.

### Parameters

*bResult* [in]

Indicates if the set command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]

Contains the client handle specifying in the SetBinaryExtendedAttribute function.

### Returns

None

### See Also

IAPIMgr::SetBinaryExtendedAttribute

## 4.28 IUsrMgr::OnSetExtendedStringAttribute

```
void IUsrMgr::OnSetExtendedStringAttribute(
        BOOL            bResult,
        ULONG           ulClientHandle
        );
```

### Description

The SV Manager Toolkit signals that pending set extended string attribute command is completed.

### Parameters

*bResult* [in]

Indicates if the set command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]

Contains the client handle specifying in the SetStringExtendedAttribute function.

### Returns

None

### See Also

IAPIMgr::SetStringExtendedAttribute

## 4.29 IUsrMgr::OnSetGroupQualityCompleted

```
BOOL __stdcall OnSetGroupQualityCompleted (
        BOOL            bResult,
        ULONG           ulClientHandle
        );
```

### Description

The SV Manager Toolkit signals that the command to set the quality of a group of variables has been completed.

### Parameters

*bResult* [in]

The result of the quality setting action. TRUE represents success in setting variable quality, FALSE represents a failure in setting variable quality.

*ulClientHandle* [in]

Contains the client handle specifying in the SetGroupQuality function.

### Returns

| | |
|---|---|
| TRUE | Return always TRUE |

### See Also

IAPIMgr::SetGroupQuality
IAPIMgr::CreateVariablesGroup
IAPIMgr::AddVariableToGroup
IAPIMgr::CloseVariablesGroup

## 4.30 IUsrMgr::OnSetSimulatedVariablesCompleted

```
void IUsrMgr::OnSetSimulatedVariables(
        BOOL            bResult,
        ULONG           ulClientHandle
        ) ;
```

### Description

The SV Manager Toolkit signals that pending data set command are completed.

### Parameters

*bResult* [in]

Indicates if the set command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]

Contains the client handle specifying in the SetSimulatedVariables function.

### Returns

None

### See Also

IAPIMgr::SetSimulatedVariables

## 4.31 IUsrMgr::OnSetVariableAttribute

```
void IUsrMgr::OnSetVariableAttribute(
        BOOL            bResult,
        ULONG           ulClientHandle
        );
```

### Description
The SV Manager Toolkit signals that pending set variable attribute command is completed.

### Parameters
*bResult* [in]
Indicates if the set command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]
Contains the client handle specifying in the SetVariableAttribute function.

### Returns
None

### See Also
IAPIMgr::SetVariableAttribute

## 4.32 IUsrMgr::OnSqlCmdCancelCompleted

```
void IUsrMgr::OnSqlCmdExecuteDataReaderCompleted (
        BOOL        bResult,
        ULONG       ulClientHandle,
        LPCTSTR     pszErrorMsg
        );
```

### Description

The SV Manager Toolkit signals that pending SqlCmdExecuteDataReader command is completed.

### Parameters

*bResult* [in]

Indicates if the set command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]

Contains the client handle specifying in the SqlCmdCancelRequest function.

*pszErrorMsg* [in]

Contains the error message when bResult is FALSE or the request status is FAILED.

### Returns

None

### See Also

IAPIMgr::SqlCmdCancelRequest

## 4.33 IUsrMgr::OnSqlCmdExecuteDataReaderCompleted

```
void IUsrMgr::OnSqlCmdExecuteDataReaderCompleted (
        BOOL            bResult,
        ULONG           ulClientHandle,
        LPCTSTR         pszErrorMsg,
        _svmgrSqlCommandExecuteDataReaderResult      *pResult
        );
```

### Description
The SV Manager Toolkit signals that pending SqlCmdExecuteDataReader command is completed.

### Parameters
*bResult* [in]
    Indicates if the set command has succeeded (TRUE) or failed (FALSE)
*ulClientHandle* [in]
    Contains the client handle specifying in the SqlCmdExecuteDataReader function.
*pszErrorMsg* [in]
    Contains the error message when bResult is FALSE or the request status is FAILED.
*pResult* [in]
    Contains the answer result.

### Type
```
typedef struct
{
    _svmgrSqlCmdStatus cmdStatus;
    ULONG       ulRowCount;
    ULONG       ulFieldCount;
    LPCTSTR*    pszFieldNames;
    LPCTSTR*    pszFieldAdoTypes;
    VARIANT**   ppValues;
} _svmgrSqlCmdExecuteDataReaderResult;


typedef enum
{
    svmgrSqlCmdStatus_Undefined = 0,
    svmgrSqlCmdStatus_Success = 1,
    svmgrSqlCmdStatus_Failed = 2,
    svmgrSqlCmdStatus_SqlConnectionDeletedBeforeAnswer = 3,
    svmgrSqlCmdStatus_CommandDeletedBeforeAnswer = 4,
    svmgrSqlCmdStatus_BadParameter = 5,
    svmgrSqlCmdStatus_CommunicationException = 6,
    svmgrSqlCmdStatus_TimeoutException = 7,
    svmgrSqlCmdStatus_Exception = 8,
} _svmgrSqlComStatus;
```

### Returns
None

### See Also
IAPIMgr::SqlCmdExecuteDataReader
IAPIMgr::SqlCmdCancelRequest

## 4.34 IUsrMgr::OnSqlCmdExecuteNonQueryCompleted

```
void IUsrMgr::OnSqlCmdExecuteNonQueryCompleted (
        BOOL            bResult,
        ULONG           ulClientHandle,
        LPCTSTR         pszErrorMsg,
        _svmgrSqlCommandExecuteNonQueryResult       *pResult
        );
```

### Description
The SV Manager Toolkit signals that pending SqlCmdExecuteNonQuery command is completed.

### Parameters
*bResult* [in]
    Indicates if the set command has succeeded (TRUE) or failed (FALSE)
*ulClientHandle* [in]
    Contains the client handle specifying in the SqlCmdExecuteNonQuery function.
*pszErrorMsg* [in]
    Contains the error message when bResult is FALSE or the request status is FAILED.
*pResult* [in]
    Contains the answer result.

### Type
```
typedef struct
{
    _svmgrSqlCmdStatus      cmdStatus;
    int                     iValue;
} _svmgrSqlCmdExecuteDataReaderResult;

typedef enum
{
    svmgrSqlCmdStatus_Undefined   = 0,
    svmgrSqlCmdStatus_Success     = 1,
    svmgrSqlCmdStatus_Failed      = 2,
    svmgrSqlCmdStatus_SqlConnectionDeletedBeforeAnswer= 3,
    svmgrSqlCmdStatus_CommandDeletedBeforeAnswer      = 4,
    svmgrSqlCmdStatus_BadParameter             = 5,
    svmgrSqlCmdStatus_CommunicationException   = 6,
    svmgrSqlCmdStatus_TimeoutException         = 7,
    svmgrSqlCmdStatus_Exception                = 8,
} _svmgrSqlComStatus;
```

### Returns
None

### See Also
IAPIMgr::SqlCmdExecuteNonQuery

## 4.35 IUsrMgr::OnSqlCmdExecuteScalarCompleted

```
void IUsrMgr::OnSqlCmdExecuteNonQueryCompleted (
        BOOL            bResult,
        ULONG           ulClientHandle,
        LPCTSTR         pszErrorMsg,
        _svmgrSqlCommandExecuteScalarResult    *pResult
        );
```

### Description
The SV Manager Toolkit signals that pending SqlCmdExecuteScalar command is completed.

### Parameters
*bResult* [in]
    Indicates if the set command has succeeded (TRUE) or failed (FALSE)
*ulClientHandle* [in]
    Contains the client handle specifying in the SqlCmdExecuteScalar function.
*pszErrorMsg* [in]
    Contains the error message when bResult is FALSE or the request status is FAILED.
*pResult* [in]
    Contains the answer result.

### Returns
None

### Type
```
typedef struct
{
    _svmgrSqlCmdStatus cmdStatus;
    VARIANT*   ppValues;
} _svmgrSqlCmdExecuteScalarResult;

typedef enum
{
    svmgrSqlCmdStatus_Undefined = 0,
    svmgrSqlCmdStatus_Success = 1,
    svmgrSqlCmdStatus_Failed = 2,
    svmgrSqlCmdStatus_SqlConnectionDeletedBeforeAnswer = 3,
    svmgrSqlCmdStatus_CommandDeletedBeforeAnswer = 4,
    svmgrSqlCmdStatus_BadParameter = 5,
    svmgrSqlCmdStatus_CommunicationException = 6,
    svmgrSqlCmdStatus_TimeoutException = 7,
    svmgrSqlCmdStatus_Exception = 8,
} _svmgrSqlComStatus;
```

### See Also
IAPIMgr::SqlCmdExecuteScalar

## 4.36 IUsrMgr::OnSqlConnectionTestConnectionCompleted

```
void IUsrMgr::OnSqlCmdExecuteNonQueryCompleted (
        BOOL            bResult,
        ULONG           ulClientHandle,
        LPCTSTR         pszErrorMsg
        );
```

### Description

The SV Manager Toolkit signals that pending SqlConnectionTestConnection command is completed.

### Parameters

*bResult* [in]

Indicates if the set command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]

Contains the client handle specifying in the SqlConnectionTestConnection function.

*pszErrorMsg* [in]

Contains the error message when bResult is FALSE or the request status is FAILED.

### Returns

None

### See Also

IAPIMgr::SqlConnectionTestConnection

## 4.37 IUsrMgr::OnSubscribeOPCItemCompleted

```
void IUsrMgr::OnSubscribeOPCItemCompleted (
        BOOL            bResult,
        ULONG           ulClientHandle,
        HRESULT         hrErrorCode
        );
```

### Description
The SV Manager Toolkit signals that a pending acknowledge command is completed.

### Parameters
*bResult* [in]
Indicates if the acknowledge command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]
Contains the client handle specifying in the SubscribeOPCItem function.

*hrErrorCode* [in]
Error associated to COM link. (Value 0 is ok, otherwise there is COM error)

### Returns
None

## 4.38 IUsrMgr::OnTimerElapsed

```
void IUsrMgr::OnTimerElapsed(
        ULONG           ulClientHandle
        );
```

### Description
The SV Manager Toolkit signals that a timer has elapsed.

### Parameters
*ulClientHandle* [in]
    Contains the client handle specifying in the SetTimer function.

### Returns
None

## 4.39 IUsrMgr::OnUnMaskVarCompleted

```
void IUsrMgr::OnUnMaskVarCompleted(
        BOOL            bResult,
        ULONG           ulClientHandle
        );
```

### Description
The SV Manager Toolkit signals that a pending unmask command is completed.

### Parameters
*bResult* [in]
: Indicates if the unmask command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]
: Contains the client handle specifying in the UnMaskVar function.

### Returns
None

## 4.40 IUsrMgr::OnUnsubscribeOPCItemCompleted

```
void  IUsrMgr:: OnUnsubscribeOPCItemCompleted (
          BOOL            bResult,
          ULONG           ulClientHandle,
          HRESULT         hrErrorCode
          );
```

### Description
The SV Manager Toolkit signals that a pending acknowledge command is completed.

### Parameters
*bResult* [in]
Indicates if the acknowledge command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]
Contains the client handle specifying in the UnsubscribeOPCItem function.

*hrErrorCode* [in]
Error associated to COM link. (Value 0 is ok, otherwise there is COM error)

### Returns
None

## 4.41 IUsrMgr::OnVariableConfigurationChange

```
void IUsrMgr::OnVariableConfigurationChange(
        blkModificationType   mdfType,
        LPCTSTR               szVariableName
        );
```

### Description
The SV Manager Toolkit signals that a configuration change command is done.

### Parameters
*blkModificationType* [out]

Indicates the modification type: blk_modADD, blk_modMDF and blk_modDEL.

*szVariableName* [out]

Indicates the name of variable which the name change

### Note
Is necessary before IAPIMgr::AdviseConfiguration and to finish and cancel IAPIMgr::CancelAdviseConfiguration

### Returns
None

### Example

```
void OnVariableConfigurationChange( blkModificationType  mdfType, LPCTSTR szVariableName)
{
        LPCTSTR szType;

        switch ( mdfType)
        {
        case blk_modADD:
                szType = "ADD";
                break;
        case blk_modMDF:
                szType = "MDF";
                break;
        case blk_modDEL:
                szType = "DEL";
                break;
        }
}
```

## 4.42 IUsrMgr::OnWriteCompleted

```
void IUsrMgr::OnWriteCompleted(
        BOOL            bResult,
        ULONG           ulClientHandle
        );
```

### Description
The SV Manager Toolkit signals that pending unitary write command are completed.

### Parameters
*bResult* [in]

Indicates if the write command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]

Contains the client handle specifying on writing request (see functions IAPIMgr::LogVarWrite, IAPIMgr::AnaVarWrite and IAPIMgr::TxtVarWrite).

### Note
Variables are written with functions IAPIMgr::LogVarWrite, IAPIMgr::AnaVarWrite and IAPIMgr::TxtVarWrite.

### Returns
None

## 4.43 IUsrMgr::OnWriteFrameCompleted

```
void IUsrMgr::OnWriteFrameCompleted (
        BOOL        bResult,
        ULONG       ulClientHandle,
        USHORT      usErrorCode,
        ULONG       ulFrameStatus,
        ULONG       ulFrameComplementaryStatus
        );
```

### Description
The SV Manager Toolkit signals that pending write command is completed.

### Parameters
*bResult* [in]
Indicates if the write command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]
Contains the client handle specifying in the CimwayWriteFrame function.

*usErrorCode* [in]
Equal 0 if the write command has succeeded.

*ulFrameStatus* [in]
Frame state after the write

*ulFrameComplementaryeStatus* [in]
Frame complementary state after the write

### Returns
None

## 4.44 IUsrMgr::OnWriteOPCItemCompleted

```
void IUsrMgr::OnWriteOPCItemCompleted (
        BOOL            bResult,
        ULONG           ulClientHandle,
        HRESULT         hrErrorCode
        );
```

### Description

The SV Manager Toolkit signals that a pending acknowledge command is completed.

### Parameters

*bResult* [in]

Indicates if the acknowledge command has succeeded (TRUE) or failed (FALSE)

*ulClientHandle* [in]

Contains the client handle specifying in the WriteOPCItem function.

*hrErrorCode* [in]

Error associated to COM link. (Value 0 is ok, otherwise there is COM error)

### Returns

None

### Example

```
void OnWriteOPCItemCompleted(BOOL bResult,
                                                ULONG ulClientHandle,
                                                HRESULT hrErrorCode)
{
        LogMessage("Write %s (hr=0x%08X, ulClientHandle=0x%08X)",
                bResult?"OK":"failed",
                hrErrorCode,
                ulClientHandle);
}
```

## 4.45 IUsrMgr::StartProject

```
void IUsrMgr::StartProject(
        ) ;
```

### Description

StartProject is a Supervisor’s event. All SV Manager Toolkit instances receive this event. It signals that the Supervisor project is starting. On the StartProject method, you are assured that all project dependant configurations are read (Especially, the real-time database is valid, so you can enumerate it).

The StartProject method is helpful to allocate, initialise project specific purpose.

### Returns

None

## 4.46 IUsrMgr::StaticEnd

```
void IUsrMgr::StaticEnd(
        ) ;
```

### Description

StaticEnd is a Supervisor’s event. All SV Manager Toolkit instances receive this event. It signals that the Supervisor is stopping (the project has already been stopped before).

The StaticEnd method is helpful to deallocate general purpose (not depending of the project). Generally, what is allocated in IUsrMgr::StaticInit is deallocated in StaticEnd.

### Returns

None

## 4.47 IUsrMgr::StaticInit

```
void IUsrMgr::StaticInit(
        ) ;
```

### Description

StaticInit is a Supervisor’s event. All SV Manager Toolkit instances receive this event. It signals that the Supervisor is starting (only the application, not the project).

The StaticInit method is helpful to allocate, initialise general purpose (not depending of the project).

### Returns

None

## 4.48 IUsrMgr::StopProject

```
void IUsrMgr::StopProject(
        ) ;
```

### Description

StopProject is a Supervisor’s event. All SV Manager Toolkit instances receive this event. It signals that the Supervisor’s project is stopping.

The StopProject method is helpful to stop some task (like advising variables…). And all task depending of the project.

### Note

Prefer to only stop some management in this method, but not deallocate memory,… that is placed in the IUsrMgr::DelProject.

### Returns

None

## 4.49 Data Stream

The data stream is a structured stream containing information about Supervisor variable value, timestamp and quality. Data streams are parameters of the 2 interface methods: OnDataChange and OnReadCompleted.

The data stream is composed of 3 parts. First, the stream header, then, the variables header and finally the values for these variables. So, we can represent the stream as :

| | | |
|---|---|---|
| _svmgrStreamHeader | _svmgrStreamVarHeader[] | _svmgrVarValue[] |

**_svmgrStreamHeader**
In this structure, we find information about the size (in byte and in variables) of the stream.

```
typedef struct
{
        DWORD dwSize;    // Size in byte
        DWORD dwVarCount;        // Size in variables
} _svmgrStreamHeader;
```

**_svmgrStreamVarHeader**
This structure contains the header of variables: the Client handle (specified by advise or read functions), the timestamp of the last change and the status.

Note that the *dwValueOffset* is the offset in byte of the value in the stream.

```
typedef struct
{
       ULONG          ulClientHandle;       // Specified by advise or read function
       DWORD          dwValueOffset;        // Offset of the values in the stream
       FILETIME       ftTimeStamp;          // Timestamp of the last change
       BYTE           bQuality;             // Quality of the variables
       LONG           lQualityEx;           // Reserved
       BOOL           bResult;              // Signal if advise or read are succeeded or not.
} _svmgrStreamVarHeader;
```

The *ulClientHandle* is the client handle passed as parameter at the call of the IAPIMgr::VarAdvise and IAPIMgr::VarRead functions.

The *dwValueOffset* value is the offset in byte in the data stream from the beginning of the data stream.

The *ftTimeStamp* is the time stamp of the value if the *bQuality* is good or the time when the value quality become bad if *bQuality* is bad.

The *bQuality* is the quality of the value. A *bQuality* value different from 0 means that the quality of this value is bad and the value at the *dwValueOffset* offset is the last good quality value.

The *lQualityEx* can be used with the *_svmgrVarStatus* to determine the meaning of the quality.

If *bResult* is set to FALSE, *dwValueOffset*, *ftTimeStamp* and *bQuality* are not consistent.

**_svmgrStreamVarValue**
This structure contains the type and value of the variables. Variable types are listed in the enum **_svmgrVarType** and are: *svmgr_vtLOG*, *svmgr_vtANA*, *svmgr_vtALARM* and *svmgr_vtTXT*.

This structure has method to get values depending of the type. So, if the type is *svmgr_vtLOG*, you must call the method *bLog()*, for *svmgr_vtANA* the method *dAna()* and for the text variables the method *szTxt()*.

```
typedef struct
{
	_svmgrVarType vt;
	union
	{
		BOOL   bLog;
		double dAna;
		char   cTxt;

		struct
		{
			BOOL bAlarmValue;
			BOOL bLogValue;
			BOOL bAck;

		} alarm;

	} val;

	BOOL   bLog( void)      { return vt == svmgr_vtALARM? val.alarm.bLogValue: val.bLog; }
	double dAna( void)      { return val.dAna; }
	char   *szTxt( void)    { return &(val.cTxt); }

	BOOL   bAlarm( void)    { return val.alarm.bAlarmValue; }
	BOOL   bAlarmLog( void) { return val.alarm.bLogValue; }
	BOOL   bAlarmAck( void) { return val.alarm.bAck; }

} _svmgrVarValue;
```

# 5 SV Manager Toolkit API

The SV Manager Toolkit API is provided by the SV Manager Toolkit by using svmgrExchangeInterface.

## 5.1 Categories

### 5.1.1 Check variables attributes

Allow the programmer to get information about variables present in the real-time database.

| | |
|---|---|
| IAPIMgr::VarIsExist | Check if a variable exist or not |
| IAPIMgr::VarGetType | Get the type of a variable |

### 5.1.2 Enumeration functions

Allow the programmer to get the enumeration of all variables present in the real-time database.

| | |
|---|---|
| IAPIMgr::CreateVarEnum | Create a variable enumeration |
| IAPIMgr::CloseVarEnum | Close a variable enumeration |
| IAPIMgr::ClearVarEnum | Clear a variable enumeration |
| IAPIMgr::VarEnum | Get a variable enumeration |

### 5.1.3 Advise variables functions

Allow the programmer to advise variables present in the real-time database.

| | |
|---|---|
| IAPIMgr::VarAdvise | Begin an advice on the value, timestamp and quality of a variable |
| IAPIMgr::VarUnadvise | Stop an advice on the value, timestamp and quality of a variable |

### 5.1.4 Read variables functions

Allow the programmer to read variables present in the real-time database.

| | |
|---|---|
| IAPIMgr::VarRead | Get the value, timestamp and quality of a variable |
| IAPIMgr::ReadGroup | Get the value, timestamp and quality of a group of variable |

### 5.1.5 Mask and UnMask variables functions

Allow the programmer to mask and unmask variables present in the real-time database.

| | |
|---|---|
| IAPIMgr::MaskVar | Mask the variable. |
| IAPIMgr::UnMaskVar | Unmask the variable. |

### 5.1.6 Ack variables functions

Allow the programmer to ack alarm variables present in the real-time database.

| | |
|---|---|
| IAPIMgr::AckAlarm | Acknowledge the alarm variable. |

### 5.1.7 Write variables functions

Allow the programmer to write variables present in the real-time database.

| | |
|---|---|
| IAPIMgr::LogVarWrite | Write a logical value in a logical or alarm variable. |
| IAPIMgr::AnaVarWrite | Write an analogical value in an analogical variable. |
| IAPIMgr::TxtVarWrite | Write a text value in a text variable. |

### 5.1.8 Set variables functions

Allow the programmer to set variables (value, timestamp and quality) present in the real-time database.

| | |
|---|---|
| IAPIMgr::SetDataSet | Set VTQ a set of internal variables. |
| IAPIMgr::GetDataSetMaxSize | Get the maximum array size can that be use with IAPIMgr::SetDataSet. |

| | |
|---|---|
| IAPIMgr::CreateRecipe | Create a recipe. |
| IAPIMgr::AddVariableToRecipe | Add variable (any source) and value to the recipe. |
| IAPIMgr::SendRecipe | Send the recipe. |
| IAPIMgr::CloseRecipe | Remove the recipe. |

| | |
|---|---|
| IAPIMgr::VarRecords | Record the VTQ of a set of internal variables. |
| IAPIMgr::GetVarRecordsMaxSize | Get the maximum array size that can be use with IAPIMgr::VarRecord. |

### 5.1.9 CimWay access

| | |
|---|---|
| IAPIMgr::CimwayReadFrame | Read a frame |
| IAPIMgr::CimwayWriteFrame | Write a frame |
| IAPIMgr::CimwayModifyFrameAddress | Modify the address of a frame |
| IAPIMgr::CimwayModifyEqtAddress | Modify the address of an equipment |

### 5.1.10 Timer and notification functions

Allow the programmer to have asynchronous treatment.

| | |
|---|---|
| IAPIMgr::SetTimer | Start a timer and notify the User DLL when elapsed. |
| IAPIMgr::Notify | Notify the User DLL. |

### 5.1.11 Display message functions

Allow the programmer to display information

| | |
|---|---|
| IAPIMgr::LogMessage | Log a formatted message in the event viewer or in the file T. |
| IAPIMgr::Trace | Log a formatted message in the event viewer or in the file TRACE.DAT if the specified trace is active. |

### 5.1.12 Variable configuration functions

Allow the programmer to get and set the attributes of a variable

| | |
|---|---|
| IAPIMgr::InitExtendedAttributesStructure | Initialize the extended attributes structure |
| IAPIMgr::FreeExtendedAttributesStructure | Free the extended attributes structure |
| IAPIMgr::GetExtendedAttributes | Get all the extended attributes of a variable |
| IAPIMgr::SetStringExtendedAttribute | Set a string extended attribute of a variable |
| IAPIMgr::SetBinaryExtendedAttribute | Set a binary extended attribute of a variable |

| | |
|---|---|
| IAPIMgr::InitAlarmAttributesStructure | Initialize the alarm attributes structure |
| IAPIMgr::FreeAlarmAttributesStructure | Free the alarm attributes structure |
| IAPIMgr::GetAlarmAttributes | Get the alarm attributes of a variable |
| IAPIMgr::InitVariableAttributesStructure | Initialize the variable attributes structure |
| IAPIMgr::FreeVariableAttributesStructure | Free the variable attributes structure |
| IAPIMgr::GetVariableAttributes | Get the attributes of a variable |
| IAPIMgr::SetVariableAttribute | Set an attribute of a variable |

### 5.1.13 Universal Data Connector functions

| | |
|---|---|
| IAPIMgr::SqlConnectonStart | Start a Sql connection |
| IAPIMgr::SqlConnectionStop | Stop a Sql connection |
| IAPIMgr::SqlConnectionTestConnection | Test a Sql connection |
| IAPIMgr::SqlCmdCancelRequest | Cancel a Sql query |
| IAPIMgr::SqlCmdExecuteDataReader | Send a Sql query with a table return |
| IAPIMgr::SqlCmdExecuteNonQuery | Send a Sql query with no return |
| IAPIMgr::SqlCmdExecuteScalar | Send a Sql query with a scalar return |

### 5.1.14 Miscellaneous functions

| | |
|---|---|
| IAPIMgr::GetVersion | Get the SV Manager Toolkit version |
| IAPIMgr::GetCurrentUser | Get the current login user name |
| IAPIMgr::GetProjectDirectory | Get the project directory name |

## 5.2 IAPIMgr::AckAlarm

```
SVMGRAPI2 IAPIMgr::AckAlarm (
        LPCSTR      pszVarName,
        LPCSTR      pszOperator,
        ULONG       ulClientHandle
        );
```

### Description

Ack alarm variable named *pszVarName*.

These ack alarm function is asynchronous one. When the pending ack command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnAckAlarmCompleted method, specifying the result of the command.

### Parameters

*pszVarName* [in]
: Contains the name of the variable.

*pszOperator* [in]
: Contains the name of the operator.

*ulClientHandle* [in]
: Associated client handle for this request.

### Returns

| | |
|---|---|
| TRUE | Acknowledge  is pending. |
| FALSE | Variable will be not acquitted. Probably not present in the real-time database. |

## 5.3 IAPIMgr::AddVariableToRecipe

```
SVMGRAPI2 IAPIMgr::AddVariableToRecipe (
        HANDLE      hRecipe,
        LPCTSTR     szVariableName,
        bool    bValue
        );
```

```
SVMGRAPI2 IAPIMgr::AddVariableToRecipe (
        HANDLE      hRecipe,
        LPCTSTR     szVariableName,
        double dValue
        );
```

```
SVMGRAPI2 IAPIMgr::AddVariableToRecipe (
        HANDLE      hRecipe,
        LPCTSTR     szVariableName,
        LPCTSTR     szValue
        );
```

### Description
Overloaded method that provides a way to add a variable to the recipe object initialized beforehand in the IAPIMgr::CreateRecipe function.

### Parameters
*hRecipe* [in-out]

The target recipe object.

*szVariableName* [in]

The name of the variable that the recipe should write to.

*bValue / dValue / szValue* [in]

The value to write to the specified variable. The recipe can set the value of boolean, register or string variables, make sure that the data types match.

### Returns

| | |
|---|---|
| TRUE | Recipe is updated with variable and write value. |
| FALSE | Recipe is not updated. Variable might not exist in the database, or incompatible value data type. |

## 5.4 IAPIMgr::AnaVarWrite

```
SVMGRAPI2 IAPIMgr::AnaVarWrite (
        LPCSTR          pszVarName,
        double          dValue,
        ULONG           ulClientHandle
        );
```

### Description

Write a variable named *pszVarName*.

This function is asynchronous. When the pending write command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnWriteCompleted method, specifying the result of the command.

### Parameters

*pszVarName*

[in]
Contains the name of the variable to write.

*dValue*

[in]
Value to write.

*ulClientHandle*

[in]
Associated client handle for this write command.

### Returns

| | |
|---|---|
| TRUE | Write  is pending. |
| FALSE | Variable cannot be written. Probably not present in the real-time database. |

## 5.5 IAPIMgr::AddOPCGroup

**SVMGRAPI2 IAPIMgr::AddOPCGroup(**
```
    LPCTSTR          pszServerID,
    LPCTSTR          pszGroupID,
    _svmgrAddOPCGroupParameters &  sAddOPCGroupParameters,
    ULONG            ulClientHandle
    );
```

### Description
Create an OPC group called *pszGroupID.*

This function is asynchronous. When the pending creation command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnAddOPCGroupCompleted method, specifying the result of the command.

### Parameters
*pszServerID* [in]

Contains the name of the existing OPC server.

*pszGroupID* [in]

Contains the name of the OPC group to create.

_svmgrAddOPCGroupParameters &sAddOPCGroupParameters [in]

Structure to write.
```
typedef struct
{
        DWORD dwPeriod;  // Milliseconds
} _svmgrAddOPCGroupParameters;
```

*ulClientHandle*[in]

Associated client handle for this command.

### Returns
| | |
|---|---|
| TRUE | Successful creation |
| FALSE | Creation failed |

### Example
```
_svmgrAddOPCGroupParameters opcParameters;
opcParameters.dwPeriod = 1000;

if ( g_pSvAPI->AddOPCGroup("SRV1", "MyGroup", opcParameters, 12345) = = FALSE)
{
        LogMessage("Unable to create group");
  }
```

## 5.6 IAPIMgr::AddVariableToGroup

```
SVMGRAPI2 IAPIMgr::AddVariableToGroup(
        HANDLE    hVariablesGroup,
        LPCTSTR   szVariableName
        );
```

### Description

Add a variable to specified group (defined by the handle hVariablesGroup).
A group handle must be closed after use by IAPIMgr::CloseVariablesGroup.

### Parameters

*hVariablesGroup* [in]

Contains the variable group handle.

*szVariableName* [in]

Contains the variable name to be added to the group.

### Returns

| | |
|---|---|
| TRUE | Successful addition |
| FALSE | Addition failed |

### See Also

IAPIMgr::CreateVariablesGroup
IAPIMgr::SetGroupQuality
IAPIMgr::CloseVariablesGroup

## 5.7 IAPIMgr::AdviseConfiguration

```
SVMGRAPI2 IAPIMgr::AdviseConfiguration(
          );
```

### Description
This function advises the configuration.

For each configuration modification on variables, the SV Manager Toolkit invokes the IUsrMgr::OnVariableConfigurationChange method, specifying the nature of the change.

### Returns
TRUE

### Note
This function has to be called only once. Additional call will generate additional call to IUsrMgr::OnVariableConfigurationChange. With the same information

## 5.8 IAPIMgr::CancelAdviseConfiguration

**SVMGRAPI2 IAPIMgr::CancelAdviseConfiguration(
);**

### Description

This function cancels the advised configuration done by IAPIMgr::AdviseConfiguration.

### Returns

TRUE

## 5.9 IAPIMgr::CimwayModifyEqtAddress

```
SVMGRAPI2 IAPIMgr::CimwayModifyEqtAddress(
        LPCSTR      pszFullEqtName,
        ULONG       ulClientHandle
        LPCSTR      pszEqtAddress
        );
```

### Description
Modify the address of the equipment named *pszFullEqtName*.

The *svmgrModifyEqtAddress* function is asynchronous. When the pending modification is completed, the SV Manager Toolkit invokes the IUsrMgr::OnModifyEqtAddressCompleted method, specifying the result of this command.

### Parameters
*pszFullFrameName* [in]

Contains the full name of the equipment to modify.
The full name of equipment is composed by the network name and equipment name with dot as separator.
Example: “MODBUS.EQT1”

*ulClientHandle* [in]

Associated client handle for this modification.

*pszEqtAddress* [in]

New address of the equipment. The range of values depends on the protocol type. See the topic CommnicationObject Parameters for this setting in scada basic help.

### Returns

| | |
|---|---|
| TRUE | The modification of equipment address is in progress |
| FALSE | The modification of equipment address is impossible. Probably the equipment is not configured in Cimway |

## 5.10 IAPIMgr::CimwayModifyFrameAddress

```
SVMGRAPI2 IAPIMgr::CimwayModifyFrameAddress(
        LPCSTR      pszFullFrameName,
        ULONG       ulClientHandle
        LPCSTR      pszFrameAddress
        );
```

### Description

Modify the address of the frame named *pszFullFrameName*.

The *svmgrModifyFrameAddress* function is asynchronous. When the pending modification is completed, the SV Manager Toolkit invokes the IUsrMgr::OnModifyFrameAddressCompleted method, specifying the result of this command.

### Parameters

*pszFullFrameName* [in]

Contains the full name of the frame to modify.
The full name of frame is composed by the network name, equipment name and frame name with dot as separator.
Example: "MODBUS.EQT1.WORD"

*ulClientHandle* [in]

Associated client handle for this modification.

*pszFrameAddress* [in]

New address of the frame. Usually, the parameter is a 16 bit integer. The range of values depends on the protocol type. See the topic CommnicationObject Parameters for this setting in scada basic help.

### Returns

| | |
|---|---|
| TRUE | The modification of frame address is in progress |
| FALSE | The modification of frame address is impossible. Probably the frame is not configured in Cimway |

## 5.11 IAPIMgr::CimwayReadFrame

```
SVMGRAPI2 IAPIMgr::CimwayReadFrame(
        LPCSTR          pszFullFrameName,
        ULONG           ulClientHandle,
        BOOL            bUseCache
        );
```

### Description

Read a frame named pszFullFrameName

The *svmgrReadFrame* function is asynchronous. When the pending read is completed, the SV Manager Toolkit invoke the IUsrMgr::OnReadFrameCompleted method, specifying the result of this read command, and if successful, the value and its quality.

### Parameters

*pszFullFrameName* [in]

Contains the full name of the frame to read.
The full name of frame is composed by the network name, equipment name and frame name with dot as separator.
Example: “MODBUS.EQT1.WORD”

*ulClientHandle* [in]

Associated client handle for this read.

*bUseCache* [in]

Force the reading to the device when this value is set to FALSE. If this value is set to TRUE, the reading can use the last cyclic refresh.

### Returns

| | |
|---|---|
| TRUE | Read frame is in progress |
| FALSE | Read frame is impossible. Probably the frame is not configured in Cimway |

## 5.12 IAPIMgr::CimwayWriteFrame

```
SVMGRAPI2  IAPIMgr::CimwayWriteFrame(
        LPCSTR          pszFullFrameName,
        ULONG           ulClientHandle
        ULONG           ulSize,
        BYTE            *pbyBuffer
        );
```

### Description

Write a frame named *pszFullFrameName* with the buffer *pbyBuffer*.

The *svmgrWriteFrame* function is asynchronous. When the pending write  is completed, the SV Manager Toolkit invoke the IUsrMgr::OnWriteFrameCompleted method, specifying the result of this write command.

### Parameters

*pszFullFrameName* [in]

Contains the full name of the frame to write.
The full name of frame is composed by the network name, equipment name and frame name with dot as separator.
Example: "MODBUS.EQT1.WORD"

*ulClientHandle* [in]

Associated client handle for this write.

*ulSize* [in]

The size in bytes of pbyBuffer. This size is in bytes.

*pbyBuffer* [in]

Pointer of frame data. The buffer size is ulSize bytes. This buffer is freed by caller.

### Returns

| | |
|---|---|
| TRUE | Write frame is in progress |
| FALSE | Write frame is impossible. Probably the frame is not configured in Cimway |

## 5.13 IAPIMgr::ClearVarEnum

```
SVMGRAPI2 IAPIMgr::ClearVarEnum (
        ULONG                ulNbVar,
        _svmgrVarEnum *      peVar
        );
```

### Description
Clear the *ulNbVar* first item of an enumerated variables array *peVar*.

### Parameters
*ulNbVar* [in]

Contains the number of item of *peVar* to clear.
If call after a svmgrVarEnum, use the content of *pulNbVar* for this parameter.

*peVar*[in-out]

The array to clear.

### Returns

| | |
|---|---|
| TRUE | Array is cleared. |
| FALSE | Array is not cleared. |

### Example
See example.

## 5.14 IAPIMgr::CloseRecipe

```
SVMGRAPI2 IAPIMgr::CloseRecipe (
        HANDLE hRecipe
        );
```

### Description
Closes and garbage collects the given recipe object.

### Parameters
*hRecipe* [in]

The target recipe object to close and delete.

### Returns

| | |
|---|---|
| TRUE | Recipe object is successfully deleted. |
| FALSE | Recipe was not deleted. The object may not currently exist. |

## 5.15 IAPIMgr::CloseVarEnum

```
SVMGRAPI2 IAPIMgr::CloseVarEnum (
        HANDLE hEnum
        );
```

### Description
Close an existing variable enumerator, created with CreateVarEnum function.

### Parameters
*hEnum* [in]

Contains the enumerator handle to close.

### Returns

| | |
|---|---|
| TRUE | The close of the enumerator has succeeded. |
| FALSE | The close of the enumerator has failed. |

### Example
See example.

## 5.16 IAPIMgr::CloseVariablesGroup

**SVMGRAPI2 IAPIMgr::CloseVariablesGroup (**
**HANDLE hVariablesGroup,**
**);**

### Description

Set free the memory used by the variable group (defined by the handle hVariablesGroup).
A group handle is created by IAPIMgr::CreateVariablesGroup,

### Parameters

*hVariablesGroup* [in]
Contains the group handle.

### Returns

| | |
|---|---|
| TRUE | The close of the variables group has succeeded. |
| FALSE | The close of the variables group has failed. |

### See Also

IAPIMgr::CreateVariablesGroup
IAPIMgr::SetGroupQuality
IAPIMgr::AddVariableToGroup

## 5.17 IAPIMgr::CreateRecipe

```
SVMGRAPI2 IAPIMgr::CreateRecipe (
        HANDLE                              &hRecipe,
        _svmgrRecipeType                    recipeType,
        _svmgrOpcRecipeExecuteMode          opcRecipeExecuteMode
        );
```

### Description

Initializes an object for holding the recipe plan.

### Parameters

*hRecipe* [in-out]

Represents the object that holds the recipe.

Example:

```
HANDLE g_hRecipe = NULL;
svmgrAPI->CreateRecipe (g_hRecipe, svmgrRecipeType_Multiple,
              svmgrOpcRecipeExecuteMode_OptimizedSerialization);
```

*recipeType* [in]

Specifies the type of recipe:
- Bloc (`svmgrRecipeType_Bloc`)
- Multiple (`svmgrRecipeType_Multiple`).

```
svmgrOpcRecipeExecuteMode_OptimizedSerialization = 0,
svmgrOpcRecipeExecuteMode_FullSerialization = 1,
svmgrOpcRecipeExecuteMode_NoOptimization = 2,
svmgrOpcRecipeExecuteMode_FullOptimization = 3,
```

*opcRecipeExecuteMode* [in]

Determines the execution mode with regards to serialization and optimization. The available options are:
- no optimization (`svmgrOpcRecipeExecuteMode_NoOptimization`)
- full optimization (`svmgrOpcRecipeExecuteMode_FullOptimization`)
- full serialization (`svmgrOpcRecipeExecuteMode_FullSerialization`)
- optimized serialization (`svmgrOpcRecipeExecuteMode_OptimizedSerialization`)

### Returns

| | |
|---|---|
| TRUE | Recipe is created. |
| FALSE | Recipe is not created. Check whether the given parameters are valid. |

## 5.18 IAPIMgr::CreateVarEnum

```
SVMGRAPI2 IAPIMgr::CreateVarEnum (
        HANDLE & hEnum
        );
```

### Description
Create a variable enumerator.
An enum handle after use must be close with the IAPIMgr::CloseVarEnum.

### Parameters
*hEnum* [out]
    Contains the enumerator handle.

### Returns
| | |
|---|---|
| TRUE | The creation of the enumerator has succeeded.<br>The parameter *hEnum* contains the enumerator handle. |
| FALSE | The creation of the enumerator has failed.<br>The parameter *hEnum* is not consistent. |

### See also
IAPIMgr::CloseVarEnum
IAPIMgr::ClearVarEnum
IAPIMgr::VarEnum

### Example
See example.

## 5.19 IAPIMgr::CreateVariablesGroup

```
SVMGRAPI2 IAPIMgr::CreateVariablesGroup (
        HANDLE & hVariablesGroup
        );
```

### Description

Create a variable group.
A group handle must be closed after use by IAPIMgr::CloseVariablesGroup.

### Parameters

*hVariablesGroup* [out]
Contains the group handle.

### Returns

| | |
|---|---|
| TRUE | The creation of the variable group has succeeded.<br>The parameter *hVariableGroup* contains the group handle. |
| FALSE | The creation has failed.<br>The parameter *hVariableGroup* is not consistent. |

### See also

IAPIMgr::AddVariableToGroup
IAPIMgr::SetGroupQuality
IAPIMgr::CloseVariablesGroup

## 5.20 IAPIMgr::FreeAdviseOptionsStructure

```
SVMGRAPI2 IAPIMgr::FreeAdviseOptionsStructure(
        _svmgrAdviseOptions & sAdviseOptions
        );
```

### Description

Free the structure initialized with IAPIMgr::InitAdviseOptionsStructure.

### Parameters

*sAdviseOptions* [in]

Reference to the structure to free.

### Returns

| | |
|---|---|
| TRUE | The structure is freed |
| FALSE | The structure cannot be freed |

### Example

See example.

## 5.21 IAPIMgr::FreeAlarmAttributesStructure

```
SVMGRAPI2 IAPIMgr::FreeAlarmAttributesStructure(
        _svmgrAlarmAttributes & sAlarmAttributes
        );
```

### Description
Free the structure initialized with IAPIMgr::InitAlarmAttributesStructure.

### Parameters
*sAlarmAttributes* [in]

Reference to the structure to free.

### Returns

| | |
|---|---|
| TRUE | The structure is freed |
| FALSE | The structure cannot be freed |

### Example
See example

### See also
IAPIMgr::InitAlarmAttributesStructure
IAPIMgr::GetAlarmAttributes

## 5.22 IAPIMgr::FreeExtendedAttributesStructure

```
SVMGRAPI2 IAPIMgr::FreeExtendedAttributesStructure(
        _svmgrExtAttributes & sExtendedAttributes
        );
```

### Description
Free the structure initialized with IAPIMgr::InitExtendedAttributesStructure.

### Parameters
*sExtendedAttributes* [in]

Reference to the structure to free.

### Returns

| | |
|---|---|
| TRUE | The structure is freed |
| FALSE | The structure cannot be freed |

### Example
See example

### See Also
IAPIMgr::InitExtendedAttributesStructure
IAPIMgr::GetExtendedAttributes
IAPIMgr::SetStringExtendedAttribute
IAPIMgr::SetBinaryExtendedAttribute
IUsrMgr::OnExtendedAttributesChange

## 5.23_IAPIMgr::FreeVariableAttributesStructure

```
SVMGRAPI2 IAPIMgr::FreeVariableAttributesStructure(
        _svmgrVariableAttributes & sVariableAttributes
        );
```

### Description
Free the structure initialized with IAPIMgr::InitVariableAttributesStructure.

### Parameters
*sVariableAttributes* [in]
Reference to the structure to free.

### Returns

| | |
|---|---|
| TRUE | The structure is freed |
| FALSE | The structure cannot be freed |

### See Also
IAPIMgr::InitVariableAttributesStructure
IAPIMgr::GetVariableAttributes,

## 5.24 IAPIMgr::GetAlarmAttributes

```
SVMGRAPI2 IAPIMgr::GetAlarmAttributes(
        LPCSTR  szVarName,
        _svmgrAlarmAttributes & sAlarmAttributes
        );
```

### Description

Get the alarm attributes of a variable named *szVarName*.

### Parameters

*szVarName* [in]

Contains the name of the variable.

*sAlarmAttributes* [out]

Structure containing the alarm attributes of this variable.

### Returns

| | |
|---|---|
| TRUE | The *sAlarmAttributes*' fields are available. |
| FALSE | The variable doesn't exist or isn't an alarm. The *sAlarmAttributes*' fields are unavailable. |

### Example

See example

### See also

IAPIMgr::InitAlarmAttributesStructure
IAPIMgr::FreeAlarmAttributesStructure

## Example

### Example : How to get alarm attributes ?

```
{
	// g_pSvAPI is the pointer to the SV Manager Toolkit.
	// szVarName contains the name of the variable

	_svmgrAlarmAttributes sAlarmAttributes;
	g_pSvAPI->InitAlarmAttributesStructure( sizeof(_svmgrAlarmAttributes),
sAlarmAttributes);

	if ( g_pSvAPI->GetAlarmAttributes( szVarName, sAlarmAttributes) == TRUE)
	{
		…
		// Use the information stored in sAlarmAttributes
		…
	}

	g_pSvAPI->FreeAlarmAttributesStructure(sAlarmAttributes);
}
```

## 5.25 IAPIMgr::GetCurrentUser

```
SVMGRAPI2 IAPIMgr::GetCurrentUser (
        LPSTR           pszUserName,
        int             iSize
        );
```

### Description

Get the current login user name.

### Parameters

*pszUserName* [out]

Address of the buffer to store user name. This buffer must be allocated by the caller. The string is always null terminated.

*iSize* [in]

Buffer size. The string is truncated when the size is greater than the buffer size.

### Returns

| | |
|---|---|
| TRUE | The name is stored in the buffer. |
| FALSE | The name cannot be stored in the buffer (Invalid pointer). |

## 5.26 IAPIMgr::GetDataSetMaxSize

```
SVMGRAPI2 ULONG IAPIMgr::GetDataSetMaxSize (
        );
```

### Description

Get the maximum size supported for a data set.

### Returns

The maximum size of the array *pDataSets* when calling the IAPIMgr::SetDataSet function.

### See also

IAPIMgr::SetDataSet
IUsrMgr::OnSetDataSetCompleted

## 5.27 IAPIMgr::GetExtendedAttributes

```
SVMGRAPI2 IAPIMgr::GetExtendedAttributes (
        LPCSTR                          szVarName,
        _svmgrExtAttributes &           sExtendedAttributes
        );
```

### Description

Get the extended attributes of a variable named *szVarName*.

### Parameters

*szVarName* [in]

Contains the name of the variable.

*sExtendedAttributes* [out]

Structure containing the extended attributes of this variable.

### Returns

| | |
|---|---|
| TRUE | The *sExtendedAttributes* structure fields are available. |
| FALSE | The variable doesn't exist or isn't an alarm. The *sExtendedAttributes* structure fields are unavailable. |

### Example

See example

### See Also

IAPIMgr::InitExtendedAttributesStructure
IAPIMgr::FreeExtendedAttributesStructure
IAPIMgr::SetStringExtendedAttribute
IAPIMgr::SetBinaryExtendedAttribute
IUsrMgr::OnExtendedAttributesChange

## Example

### Example : How to get extended attributes ?

```
{
	// g_pSvAPI is the pointer to the SV Manager Toolkit.
	// szVarName contains the name of the variable to get the alarm attributes

	_svmgrExtAttributes sExtendedAttributes;
	g_pSvAPI->InitExtendedAttributesStructure( sizeof(_svmgrExtAttributes),
sExtendedAttributes);

	if ( g_pSvAPI->GetExtendedAttributes( szVarName, sExtendedAttributes) == TRUE)
	{
		…
		// Use the information stored in sExtendedAttributes
		// For example, domain and nature attributes
		// sExtendedAttributes.StringAttributes[ExtStringAttribute_Domain]
		// sExtendedAttributes.StringAttributes[ExtStringAttribute_Nature]
		…
	}

	g_pSvAPI-> FreeExtendedAttributesStructure (sExtendedAttributes);
}
```

## 5.28 IAPIMgr::GetProjectDirectory

```
SVMGRAPI2 IAPIMgr:: GetProjectDirectory (
        LPSTR szProjectDirectoryName,
        );

SVMGRAPI2 IAPIMgr:: GetProjectDirectory (
        _svmgrProjectDirectory      eDirectory,
        LPSTR                       szProjectDirectoryName,
        Int                         nSize
        );
```

### Description
Get the project directory and write it over *szProjectDirectoryName*.

### Parameters
*eDirectory* [in]

Specify the directory to get
Can be:
- svmgrProjectDirectory_Root: project root directory
- svmgrProjectDirectory_Cfg: project configuration directory (subdirectory C)
- svmgrProjectDirectory_Persistent: project persistent directory (subdirectory PER)
- svmgrProjectDirectory_Temporary: project temporary directory (subdirectory TMP)

*szProjectDirectory* [out]

String contains the name of project directory.

*nSize* [in]

Maximum size of the string *szProjectDirectory*

### Returns
TRUE
FALSE

### Example

```
#define PLC_FILE_NAME "plcRecipe.txt"
…
{
        CString szIniFileName; // contain the name of directory plus the file name
        char szProjectDirectory[_MAX_PATH+1]; // this variable will contain the name directory

        g_pSvAPI->GetProjectDirectory(svmgrProjectDirectory_Cfg, szProjectDirectory,
_MAX_PATH);

        szIniFileName.Format("%s%s", szProjectDirectory, PLC_FILE_NAME);

}
```

## 5.29 IAPIMgr::GetVariableAttributes

**SVMGRAPI2 IAPIMgr::GetVariableAttributes (**

| | |
|---|---|
| **LPCSTR** | **szVarName,** |
| **_svmgrVariableAttributes &** | **sVariableAttributes** |
| **);** | |

### Description
Get the attributes of a variable named *szVarName*.

### Parameters
*szVarName* [in]

Contains the name of the variable.

*sVariableAttributes* [out]

Structure containing the attributes of this variable.

### Returns

| | |
|---|---|
| TRUE | The *sVariableAttributes* structure fields are available. |
| FALSE | The variable doesn't exist or isn't an alarm. The *sVariableAttributes* structure fields are unavailable. |

### See Also
IAPIMgr::InitVariableAttributesStructure
IAPIMgr::FreeVariableAttributesStructure
IAPIMgr::SetVariableAttribute

### Example
```
CString szVarName="test.var1"; //Name of variable to ask for attributes
_svmgrVariableAttributes sAttributes; //Struct Attributes
//First: initialize avec InitVariableAttributesStructure
g_pSvAPI->InitVariableAttributesStructure( sizeof(_svmgrVariableAttributes), sAttributes);
//Second: is possible get the Attributes
g_pSvAPI->GetVariableAttributes(szVarName,sAttributes);
//To save into a Varibale
szResult.Format ("VARIABLEATTRIBUTES(1),%s,OK, Description:%s, Format:%s, Unit:%s",
              szVarName, sAttributes.szDescription, sAttributes.szFormat, sAttributes.szUnit);
```

## 5.30 IAPIMgr::GetVarRecordsMaxSize

**SVMGRAPI2 ULONG IAPIMgr::GetVarRecordsMaxSize (**
**);**

### Description

Get the maximum size supported for a VarRecords.

### Returns

The maximum size of the array *pVarRecords* when calling the IAPIMgr::VarRecords function.

### See also

IAPIMgr::VarRecords

## 5.31 IAPIMgr::GetVersion

**SVMGRAPI2 IAPIMgr::GetVersion (**
**SVMGR_VERSION_INFO *      pVersionInfo**
**);**

### Description

Get the version of the SV Manager Toolkit.

This structure containsdata about version of SV Manager Toolkit.

```
typedef struct
{
        DWORD dwMajorVersion;
        DWORD dwMinorVersion;
        DWORD dwBuildNumber; // Not used
}
SVMGR_VERSION_INFO;
```

### Parameters

*pVersionInfo* [out]

Pointer to a structure of type SVMGR_VERSION_INFO.

### Returns

| | |
|---|---|
| TRUE | Informations are stored in the structure. |
| FALSE | Informations cannot be stored in the structure (Invalid pointer). |

## 5.32 IAPIMgr::InitAdviseOptionsStructure

```
SVMGRAPI2 IAPIMgr::InitAdviseOptionsStructure(
        int                             iStructSize,
        _svmgrAdviseOptions &           sAdviseOptions
        );
```

### Description

Initialize the structure used by IAPIMgr::VarAdvise.

### Parameters

*iStructSize* [in]

Size of the structure. This parameter must be set by the caller at sizeof(_svmgrAdviseOptions). It allows the user DLL to be compatible with the next versions of the SV Manager Toolkit.

*sAdviseOptions* [out]

Reference to the structure to initialize.

### Returns

| | |
|---|---|
| TRUE | The structure is initialized |
| FALSE | The structure cannot be initialized |

### Example

See example.

### See also

IAPIMgr::FreeAdviseOptionsStructure

## 5.33 IAPIMgr::InitAlarmAttributesStructure

```
SVMGRAPI2 IAPIMgr::InitAlarmAttributesStructure(
        int                          iStructSize,
        _svmgrAlarmAttributes &      sAlarmAttributes
        );
```

### Description
Initialize the structure used by IAPIMgr::GetAlarmAttributes.

### Parameters
*iStructSize* [in]

Size of the structure. This parameter must be set by the caller at sizeof(_svmgrAlarmAttributes). It allows the user DLL to be compatible with the next versions of the SV Manager Toolkit.

*sAlarmAttributes* [out]

Reference to the structure to initialize.

### Returns

| | |
|---|---|
| TRUE | The structure is initialized |
| FALSE | The structure cannot be initialized |

### Example
See example

### See also
IAPIMgr::FreeAlarmAttributesStructure
IAPIMgr::GetAlarmAttributes

## 5.34 IAPIMgr::InitExtendedAttributesStructure

```
SVMGRAPI2 IAPIMgr::InitExtendedAttributesStructure(
        int                          iStructSize,
        _svmgrExtAttributes &        sExtendedAttributes
        );
```

### Description

Initialize the structure used by IAPIMgr::GetExtendedAttributes.

### Parameters

*iStructSize* [in]

Size of the structure. This parameter must be set by the caller at sizeof (_svmgrExtAttributes). It allows the user DLL to be compatible with the next versions of the SV Manager Toolkit.

*sExtendedAttributes* [out]

Reference to the structure to initialize.

### Returns

| | |
|---|---|
| TRUE | The structure is initialized |
| FALSE | The structure cannot be initialized |

### Example

See example

### See also

IAPIMgr::FreeExtendedAttributesStructure
IAPIMgr::GetExtendedAttributes
IAPIMgr::SetStringExtendedAttribute
IAPIMgr::SetBinaryExtendedAttribute
IUsrMgr::OnExtendedAttributesChange

## 5.35 IAPIMgr::InitVariableAttributesStructure

```
SVMGRAPI2 IAPIMgr::InitVariableAttributesStructure(
        int                             iStructSize,
        _svmgrVariableAttributes&       sVariableAttributes
        );
```

### Description

Initialize the structure used by IAPIMgr::GetVariableAttributes.

### Parameters

*iStructSize* [in]

Size of the structure. This parameter must be set by the caller at sizeof (_svmgrVariableAttributes). It allows the user DLL to be compatible with the next versions of the SV Manager Toolkit.

*sVariableAttributes* [out]

Reference to the structure to initialize.

### Returns

| | |
|---|---|
| TRUE | The structure is initialized |
| FALSE | The structure cannot be initialized |

### See Also

IAPIMgr::FreeVariableAttributesStructure
IAPIMgr::GetVariableAttributes

## 5.36 IAPIMgr::LogMessage

```
SVMGRAPI2 IAPIMgr::LogMessage (
        USHORT          usLevel,
        USHORT          usDest,
        LPSTR           pszFormat,
        [argument]      …
        );
```

### Description

Log a formatted message in the SCADA software event viewer. This control format used in this function is the same format used by the printf function.

### Parameters

*usLevel* [in]

Message level. The level of the message you log is an information (SVMGR_LVL_INFO), a warning (SVMGR_LVL_WARNING) or a fatal error (SVMGR_LVL_FATAL)

*pszDest* [in]

Message destination. The destination message is the event viewer (SVMGR_DEST_VIEWER) or the file T, located in the ETC directory in the base directory of your project (SVMGR_DEST_FILET) or any combination of these destinations.

*pszFormat* [in]

Control format. See the printf documentation for the format specification.

*[argument]* [in]

Optional argument depending of the control format.

### Returns

TRUE
FALSE

### Note

The source of the message will be the SV Manager Toolkit id (MgrToolkit1 to MgrToolkit8).

## 5.37 IAPIMgr::LogVarWrite

```
SVMGRAPI2 IAPIMgr::LogVarWrite (
        LPCSTR          pszVarName,
        BOOL            bValue,
        ULONG           ulClientHandle
        );
```

### Description

Write a variable named *pszVarName*.

This function is asynchronous. When the pending write command is completed, the SV Manager Toolkit invoke the IUsrMgr::OnWriteCompleted method, specifying the result of the command.

### Parameters

*pszVarName* [in]

Contains the name of the variable to write.

*bValue* [in]

Value to write.

*ulClientHandle* [in]

Associated client handle for this write command.

### Returns

| | |
|---|---|
| TRUE | Write is pending. |
| FALSE | Variable cannot be written. Probably not present in the real-time database. |

## 5.38 IAPIMgr::MaskVar

```
SVMGRAPI2 IAPIMgr::MaskVar (
        _svmgrLevelMask     MaskLevel,
        LPCSTR              pszVarName,
        LPCSTR              pszOperator,
        ULONG               ulClientHandle
        );
```

### Description

Mask variable named *pszVarName*.

This function is asynchronous. When the pending mask command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnMaskVarCompleted method, specifying the result of the command.

### Parameters

*MaskLevel* [in]

Contains the mask level to activate, it can be one of the following values.

| | |
|---|---|
| svmgr_lvlUSERPROG1 | // program level 1 |
| svmgr_lvlUSERPROG2 | // program level 2 |
| svmgr_lvlUSERPROG3 | // program level 3 |
| svmgr_lvlUSERPROG4 | // program level 4 |
| svmgr_lvlOPERATOR | // operator |
| svmgr_lvlINHIB | // inhibited |
| svmgr_lvlALARM | // alarm |

*pszVarName* [in]

Contains the name of the variable.

*pszOperator* [in]

Contains the name of the operator.

*ulClientHandle* [in]

Associated client handle for this request.

### Returns

| | |
|---|---|
| TRUE | Acknowledge is pending. |
| FALSE | Variable will be not mask. Probably not present in the real-time database. |

## 5.39 IAPIMgr::Notify

```
SVMGRAPI2 IAPIMgr::Notify (
        ULONG  ulClientHandle
        );
```

### Description

Notify the SV Manager Toolkit that an user event occurs.

This function is very useful in a multi-threading environment. In fact, it allows an another thread to notify a specific event to the main thread of the User DLL. Note that the main thread of your User DLL is the CwinThread of the SV Manager Toolkit.

Or, if you have to do a long lasting calculation or else on the main thread of the User DLL, the new messages must wait for the treatment completion to be treat, except if you split the calculation and treat them on the IUsrMgr::OnNotify method. Very useful if you want to have a time watchdog on your calculation see IAPIMgr::SetTimer function.

### Parameters

*ulClientHandle* [in]

Associated client handle for this notification.

### Returns

TRUE
FALSE

## 5.40 IAPIMgr::ReadGroup

**SVMGRAPI2 IAPIMgr::ReadGroup (**
**HANDLE hVariablesGroup,**
**ULONG   ulClientHandle**
**);**

### Description

Reads the specified group of variables. Works in conjunction with IAPIMgr::CreateVariablesGroup and IAPIMgr::AddVariableToGroup to specify the variables to be read.

This function is asynchronous. The SV Manager Toolkit will invoke the IUsrMgr::OnReadGroupCompleted method when the read is fully completed.

### Parameters

*hVariablesGroup* [in]

The list of variables to be read. The group is defined by IAPIMgr::CreateVariablesGroup and IAPIMgr::AddVariableToGroup.

Example:

```
HANDLE varGrp;
std::vector<char *> varList;

// fill varList with names of variables to be read

if(svmgrAPI->CreateVariablesGroup(varGrp))
{
        size_t size = varList.size();

        for(size_t i = 0; i < size; i++)
        {
                svmgrAPI->AddVariableToGroup(varGrp, varList[i]);
        }
        svmgrAPI->ReadGroup(varGrp, (ULONG)varGrp);
}
```

*ulClientHandle* [in]

The associated client handle for this read.

### Returns

| | |
|---|---|
| TRUE | Read is pending. |
| FALSE | Read is cancelled. The read could not complete probably because the variables are not present in the realtime database. |

## 5.41 IAPIMgr::ReadOPCItem

**SVMGRAPI2 IAPIMgr::ReadOPCItem (**
**ULONG ulGroupServerHandle,**
**LPCTSTR pszItemID,**
**ULONG ulClientHandle**
**);**

### Description

Make a read of an OPC Item.

This function is asynchronous. When the pending command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnReadOPCItemCompleted method, specifying the result of the command.

### Parameters

*ulGroupServerHandle* [in]
Handle given by function OnAddOPCGroupCompleted.

*pszItemID* [in]
Contains the name of the variable (TAG). Example "Device1.Tag1"

*ulClientHandle* [in]
Associated client handle for this request.

### Returns

| | |
|---|---|
| TRUE | Successful. |
| FALSE | Read failed |

### Example

```
g_pSvAPI->ReadOPCItem(g_ulGroupServerHandle /*Value has got from OnAddOPCGroupCompleted */,
"Device1.ReadTag1", 4321);
```

## 5.42 IAPIMgr::RemoveOPCGroup

**SVMGRAPI2 IAPIMgr::RemoveOPCGroup (**
**ULONG ulGroupServerHandle,**
**ULONG ulClientHandle**
**);**

### Description

Remove an OPC Group.

This function is asynchronous. When the pending command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnRemoveOPCGroupCompleted method, specifying the result of the command.

### Parameters

*ulGroupServerHandle* [in]
Handle given by function OnAddOPCGroupCompleted.

*ulClientHandle* [in]
Associated client handle for this request.

### Returns

| | |
|---|---|
| TRUE | Successful. |
| FALSE | Read failed |

### Example

```
void StopProject()
{
	if(g_ulGroupServerHandle != 0)
	{
		g_pSvAPI->RemoveOPCGroup(g_ulGroupServerHandle, 12345);
	}
}
```

## 5.43 IAPIMgr::SendRecipe

**SVMGRAPI2 IAPIMgr::SendRecipe (**
**HANDLE hRecipe,**
**ULONG ulClientHandle**
**);**

### Description

Signal the SV Manager Toolkit to execute the recipe provided.

### Parameters

*hRecipe* [in]

The target recipe object which defines the execution of the recipe.

*ulClientHandle* [in]

The associated client handle for this group write operation.

### Returns

| | |
|---|---|
| TRUE | The recipe is now pending / being processed. |
| FALSE | Recipe will not be executed. The contents of the recipe may be corrupt or malformed. |

### Example

```
HANDLE hRecipe = NULL;
g_pSvAPI->CreateRecipe (hRecipe, svmgrRecipeType_Bloc,
                      svmgrOpcRecipeExecuteMode_OptimizedSerialization);

POSITION pos;
pos = g_FIFOList.GetHeadPosition();
while (pos)
{
       CVariable * pVarFIFO;
        pVarFIFO = g_FIFOList.GetNext(pos);
       g_pSvAPI->AddVariableToRecipe (hRecipe,pVarFIFO->GetName(),pVarFIFO->GetFIFOValue());
}
g_pSvAPI->SendRecipe(hRecipe,(ULONG)0);
g_pSvAPI->CloseRecipe(hRecipe);
```

## 5.44 IAPIMgr::SetAlarmAttribute

```
SVMGRAPI2 IAPIMgr:: SetAlarmAttribute (
        LPCSTR                          szAlarmName,
        _svmgrAlarmAttributeIds         eAttributeId,
        int                             iAttributeValue,
        ULONG                           ulClientHandle
        );
```

### Description

Set an Alarm attribute of a variable named *szAlarmName*.

This function is asynchronous. When the pending set command is completed, the SV Manager Toolkit invoke the IUsrMgr::OnSetAlarmAttribute method, specifying the result of the command.

### Parameters

*SzAlarmName* [in]
Contains the name of the variable.

*eAttributeId* [in]
Id of the attribute to set.

*iAttributeValue* [in]
Value to set for this attribute.

*ulClientHandle* [in]
Associated client handle for this request.

### Returns

| | |
|---|---|
| TRUE | The setting of this value is pending. |
| FALSE | The setting of this value has been refused. The variable doesn’t exist. |

### See Also

IUsrMgr::OnSetAlarmAttribute

## 5.45 IAPIMgr::SetBinaryExtendedAttribute

```
SVMGRAPI2 IAPIMgr::SetBinaryExtendedAttribute(
        LPCTSTR                          szName,
        _svmgrExtBinaryAttributeIds      eExtAttributeId,
        _svmgrExtBinaryAttributeValue    uExtAttributeValue,
        ULONG                            ulClientHandle
        );
```

### Description
Set a binary attribute of a variable named *pszVarName*.

This function is asynchronous. When the pending set command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnSetExtendedBinaryAttribute method, specifying the result of the command.

### Parameters
*szName* [in]
    Contains the name of the variable.

*eExtAttributeId* [in]
    Id of the attribute to set.

*uExtAttributeValue* [in]
    Value to set for this attribute.

*ulClientHandle* [in]
    Associated client handle for this request.

### Returns

| | |
|---|---|
| TRUE | The setting of this value is pending. |
| FALSE | The setting of this value has been refused. The variable doesn't exist. |

### See Also
IAPIMgr::InitExtendedAttributesStructure
IAPIMgr::FreeExtendedAttributesStructure
IAPIMgr::GetExtendedAttributes
IAPIMgr::SetStringExtendedAttribute
IUsrMgr::OnSetExtendedBinaryAttribute

## 5.46 IAPIMgr::SetDataSet

```
SVMGRAPI2 IAPIMgr::SetDataSet(
        ULONG                ulDataSetSize,
        _svmgrDataSet *      pDataSets,
        ULONG                ulClientHandle
        );
```

### Description

Set value, timestamp and quality of a set of internal variables. Note that to set variables from other sources you must use IAPIMgr::SendRecipe method.

This function is asynchronous. When the pending set command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnSetDataSetCompleted method, specifying the result of the command.

### Parameters

*ulDataSetSize* [in]

Contains the size of the array *pDataSets*. This value cannot be greater than the value returns by the function IAPIMgr::GetDataSetMaxSize.

*pDataSets* [in]

Array of variables to set.

*ulClientHandle* [in]

Associated client handle for this request.

### Returns

| | |
|---|---|
| TRUE | The variable settings are pending. |
| FALSE | The variable settings have been refused. For example, array size is too large. |

### See Also

IUsrMgr::OnSetDataSetCompleted
IAPIMgr::GetDataSetMaxSize

## 5.47 IAPIMgr::SetGroupQuality

```
SVMGRAPI2 IAPIMgr::SetGroupQuality(
        HANDLE          hVariablesGroup,
        _svmgrQuality quality,
        ULONG           ulClientHandle
        );
```

### Description

Set a quality in a group defined by the handle hVariablesGroup. After the setting is done the callback function **OnSetGroupQualityCompleted**(BOOL bResult, ULONG ulClientHandle) is called and the bResult in it is set to TRUE.

Take a look at the following functions: IAPIMgr::CreateVariablesGroup, IAPIMgr::AddVariableToGroup, IAPIMgr::CloseVariablesGroup and IUsrMgr::OnSetGroupQualityCompleted.

A group handle must be closed after use by IAPIMgr::CloseVariablesGroup.

### Parameters

*hVariablesGroup* [in]
Contains the variable group handle.

*quality* [in]
Contains the variable quality to set.

*ulClientHandle* [in]
Contains the handle used by OnSetGroupQualityCompleted to identify this command.

### Returns

| | |
|---|---|
| TRUE | The variable settings are pending. |
| FALSE | The variable settings have been refused. For example, array size is too large. |

### See Also

IUsrMgr::OnSetSimulatedVariablesCompleted

## 5.48 IAPIMgr::SetSimulatedVariables

```
SVMGRAPI2 IAPIMgr::SetSimulatedVariables (
        ULONG                           ulSeriesSize,
        _svmgrSimulatedVariable *       pSimulatedVariables,
        ULONG                           ulClientHandle
        );
```

### Description
Set attribute Simulated of variables.

This function is asynchronous. When the pending set command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnSetSimulatedVariablesCompleted method, specifying the result of the command.

### Parameters
*ulDataSetSize* [in]
Contains the size of the array *pSimulatedVariables* .

*pSimulatedVariables* [in]
Array of variables to set.

*ulClientHandle* [in]
Associated client handle for this request.

### Returns

| | |
|---|---|
| TRUE | The variable settings are pending. |
| FALSE | The variable settings have been refused. For example, array size is too large. |

### See Also
IUsrMgr::OnSetSimulatedVariablesCompleted

### Example
```
_svmgrSimulatedVariable SimulatedVariables;
SimulatedVariables.szVariableName="VARS.V1"; //It is the name of the variable
SimulatedVariables.bEnableSimulated=TRUE;     //The State of attribute Simuate

g_pSvAPI->SetSimulatedVariables( 1 , &SimulatedVariables, (ULONG)ulClientHandle);
```

## 5.49 IAPIMgr::SetStringExtendedAttribute

```
SVMGRAPI2 IAPIMgr::SetStringExtendedAttribute(
        LPCTSTR                         szName,
        _svmgrExtStringAttributeIds     eExtAttributeId,
        LPCSTR                          szExtAttributeValue,
        ULONG                           ulClientHandle
        );
```

### Description

Set a string attribute of a variable named *pszVarName*.

This function is asynchronous. When the pending set command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnSetExtendedStringAttribute method, specifying the result of the command.

### Parameters

*szName* [in]
  Contains the name of the variable.

*eExtAttributeId* [in]
  Id of the attribute to set.

*szExtAttributeValue* [in]
  Value to set for this attribute.

*ulClientHandle* [in]
  Associated client handle for this request.

### Returns

| | |
|---|---|
| TRUE | The setting of this value is pending. |
| FALSE | The setting of this value has been refused. The variable doesn’t exist. |

### See Also

IAPIMgr::InitExtendedAttributesStructure
IAPIMgr::FreeExtendedAttributesStructure
IAPIMgr::GetExtendedAttributes
IAPIMgr::SetBinaryExtendedAttribute
IUsrMgr::OnSetExtendedStringAttribute

## 5.50 IAPIMgr::SetVariableAttribute

```
SVMGRAPI2 IAPIMgr::SetVariableAttribute(
        LPCTSTR                      szName,
        _svmgrVariableAttributeIds   eAttributeId,
        LPCSTR                       szAttributeValue,
        ULONG                        ulClientHandle
        );
```

### Description
Set an attribute of a variable named *pszVarName*.

This function is asynchronous. When the pending set command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnSetVariableAttribute method, specifying the result of the command.

### Parameters
*szName* [in]

Contains the name of the variable.

*eAttributeId* [in]

Id of the attribute to set. (ex : svmgrVariableAttribute_Unit, svmgrVariableAttribute_Format,etc)

*szAttributeValue* [in]

Value to set for this attribute.

*ulClientHandle* [in]

Associated client handle for this request.

### Returns

| | |
|---|---|
| TRUE | The setting of this value is pending. |
| FALSE | The setting of this value has been refused. The variable doesn't exist. |

### See Also
IAPIMgr::GetVariableAttributes
IUsrMgr::OnSetVariableAttribute

### Example
```
CString szVarName="test.var1";
g_pSvAPI->SetVariableAttribute(szVarName,svmgrVariableAttribute_Unit,"KW",0);
```

## 5.51 IAPIMgr::SetTimer

```
SVMGRAPI2 IAPIMgr::SetTimer (
        ULONG  ulMsDelay,
        ULONG  ulClientHandle
        );
```

### Description

When you call this function, the User DLL will be invoked in the IUsrMgr::OnTimerElapsed when the delay specified as parameter is elapsed.

Note that the delay value is a minimal one. In fact as explain in the **SV Manager Toolkit overview**, the IAPIMgr::SetTimer send a message to the TIMER manager and when this delay is elapsed, this manager send a message to the SV Manager Toolkit. If other messages are already in the SV Manager Toolkit message queue (as DataChange or WriteCompleted messages) the User DLL will receive the message TimerElapsed only after that all the previous messages have been treated.

### Parameters

*ulClientHandle* [in]

Associated client handle for this timer.

### Returns

| | |
|---|---|
| TRUE | The timer is set |
| FALSE | The time ris not triggered |

## 5.52 IAPIMgr::SqlCmdCancelRequest

```
SVMGRAPI2 IAPIMgr:: SqlCmdCancelRequest (
        ULONG       ulClientHandle,
        LPCTSTR     pszSqlConnectionName
        );
```

### Description

Cancel the current SQL request.

This function is asynchronous. When the pending command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnSqlCmdCancelRequestCompleted method, specifying the result of the command.

### Parameters

*ulClientHandle* [in]

Associated client handle for this request.

*pszSqlConnectionName* [in]

Contains the name of the configured Sql connection.

### Returns

| | |
|---|---|
| TRUE | Successful. |
| FALSE | Reqest failed |

### See Also

IUsrMgr::OnSqlCmdCancelRequestCompleted

## 5.53 IAPIMgr::SqlCmdExecuteDataReader

```
SVMGRAPI2 IAPIMgr:: SqlCmdExecuteDataReader (
        ULONG           ulClientHandle,
        LPCTSTR         pszSqlConnectionName,
        LPCTSTR         pszSqlCmd,
        );
```

### Description
Send a SQL command using a configured Sql connection. The answer expected is an array.

This function is asynchronous. When the pending command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnSqlCmdExecuteDataReaderCompleted method, specifying the result of the command.

### Parameters
*ulClientHandle* [in]
	Associated client handle for this request.
*pszSqlConnectionName* [in]
	Contains the name of the configured Sql connection.
*pszSqlCmd* [in]
	Contains the Sql request

### Returns

| | |
|---|---|
| TRUE | Successful. |
| FALSE | Reqest failed |

### See Also
IUsrMgr::OnSqlCmdExecuteDataReaderCompleted
IAPIMgr::SqlCmdCancelRequest

## 5.54 IAPIMgr::SqlCmdExecuteNonQuery

```
SVMGRAPI2 IAPIMgr:: SqlCmdExecuteNonQuery (
        ULONG       ulClientHandle,
        LPCTSTR     pszSqlConnectionName,
        LPCTSTR     pszSqlCmd,
        );
```

### Description

Send a SQL command using a configured Sql connection.  The query should not return any value.

This function is asynchronous. When the pending command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnSqlCmdExecuteNonQueryCompleted method, specifying the result of the command.

### Parameters

*ulClientHandle* [in]
    Associated client handle for this request.
*pszSqlConnectionName* [in]
    Contains the name of the configured Sql connection.
*pszSqlCmd* [in]
    Contains the Sql request

### Type

```
typedef struct
{
    _svmgrSqlCmdStatus cmdStatus;
    int iValue;

} _svmgrSqlCmdExecuteDataReaderResult;

typedef enum
{
    svmgrSqlCmdStatus_Undefined = 0,
    svmgrSqlCmdStatus_Success = 1,
    svmgrSqlCmdStatus_Failed = 2,
    svmgrSqlCmdStatus_SqlConnectionDeletedBeforeAnswer = 3,
    svmgrSqlCmdStatus_CommandDeletedBeforeAnswer = 4,
    svmgrSqlCmdStatus_BadParameter = 5,
    svmgrSqlCmdStatus_CommunicationException = 6,
    svmgrSqlCmdStatus_TimeoutException = 7,
    svmgrSqlCmdStatus_Exception = 8,
} _svmgrSqlComStatus;
```

### Returns

| | |
|---|---|
| TRUE | Successful. |
| FALSE | Reqest failed |

### See Also

IUsrMgr::OnSqlCmdExecuteNonQueryCompleted
IAPIMgr::SqlCmdCancelRequest

## 5.55 IAPIMgr::SqlCmdExecuteScalar

```
SVMGRAPI2 IAPIMgr:: SqlCmdExecuteScalar (
        ULONG          ulClientHandle,
        LPCTSTR        pszSqlConnectionName,
        LPCTSTR        pszSqlCmd,
        );
```

### Description

Send a SQL command using a configured Sql connection.  The answer expected is a scalar value

This function is asynchronous. When the pending command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnSqlCmdExecuteScalarCompleted method, specifying the result of the command.

### Parameters

*ulClientHandle* [in]

Associated client handle for this request.

*pszSqlConnectionName* [in]

Contains the name of the configured Sql connection.

*pszSqlCmd* [in]

Contains the Sql request

### Returns

| | |
|---|---|
| TRUE | Successful. |
| FALSE | Reqest failed |

### See Also

IUsrMgr::OnSqlCmdExecuteScalarCompleted
IAPIMgr::SqlCmdCancelRequest

## 5.56 IAPIMgr::SqlConnectionStart

```
SVMGRAPI2 IAPIMgr:: SqlConnectionStart (
        LPCTSTR     pszSqlConnectionName
        );
```

### Description
Force a connection to start.

This function is synchronous.

### Parameters
*pszSqlConnectionName* [in]
Contains the name of a configured Sql connection.

### Returns

| | |
|---|---|
| TRUE | Successful. |
| FALSE | Connection failed |

## 5.57 IAPIMgr::SqlConnectionStop

```
SVMGRAPI2 IAPIMgr:: SqlConnectionStop (
        LPCTSTR     pszSqlConnectionName
        );
```

### Description
Force a connection to stop.

This function is synchronous.

### Parameters
*pszSqlConnectionName* [in]
Contains the name of a configured Sql connection.

### Returns

| | |
|---|---|
| TRUE | Successful. |
| FALSE | Connection stop failed |

## 5.58 IAPIMgr::SqlConnectionTestConnection

```
SVMGRAPI2 IAPIMgr:: SqlConnectionTestConnection (
        ULONG           ulClientHandle,
        LPCTSTR         pszSqlConnectionName
        );
```

### Description

Send a test connection query using a configured Sql connection.

This function is asynchronous. When the pending command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnSqlConnectionTestConnectionCompleted method, specifying the result of the command.

### Parameters

*ulClientHandle* [in]

Associated client handle for this request.

*pszSqlConnectionName* [in]

Contains the name of the configured Sql connection.

### Returns

| | |
|---|---|
| TRUE | Successful. |
| FALSE | Test connection failed |

### See Also

IUsrMgr::OnSqlConnectionTestConnectionCompleted

## 5.59 IAPIMgr::SubscribeOPCItem

**SVMGRAPI2 IAPIMgr:: SubscribeOPCItem (**
```
        ULONG           ulGroupServerHandle,
        LPCTSTR         pszItemID,
        ULONG           ulClientHandle
        );
```

### Description
Subscribe an OPC Item.

This function is asynchronous. When the pending command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnSubscribeOPCItemCompleted method, specifying the result of the command.

### Parameters
*ulGroupServerHandle* [in]
    Handle given by function OnAddOPCGroupCompleted.

*pszItemID* [in]
    Contains the name of the variable (TAG). Example "Device1.Tag1"

*ulClientHandle* [in]
    Associated client handle for this request.

### Returns
| | |
|---|---|
| TRUE | Successful. |
| FALSE | Read failed |

### Example
```
void OnAddOPCGroupCompleted ( BOOL bResult, ULONG ulClientHandle, ULONG ulGroupServerHandle,
                                HRESULT hrErrorCode)
{
        g_ulGroupServerHandle = ulGroupServerHandle;

        if(bResult == TRUE)
        {

                if(g_pSvAPI->SubscribeOPCItem(g_ulGroupServerHandle, "Device1.Tag1", 555) ==
FALSE)
                {
                        LogMessage("Unable to subscribe");
                }
        }
}
```

## 5.60 IAPIMgr::Trace

```
SVMGRAPI2 IAPIMgr::Trace (
        ULONG           ulTraceFlag,
        LPSTR           pszFormat,
        [argument]      …
        );
```

### Description

Log a formatted message in the SCADA software event viewer and in the TRACE.DAT in ETC directory under the Supervisor installation directory. This control format used in this function is the same format used by the printf function.

### Parameters

*ulTraceFlag* [in]

Trace Flag. The trace flag is the condition for a trace to be logged in the event viewer. If this trace is not active, the function fails and returns FALSE, else the function returns TRUE.
The value for this parameter is SVMGR_FLAG_BIT0 to SVMGR_FLAG_BIT31.

*pszFormat* [in]

Control format. See the printf documentation for the format specification.
If this parameter value is null, the returned value specifies if the trace flag specified as parameter is active or not.

*[argument]* [in]

Optional argument depending of the control format.

### Returns

| | |
|---|---|
| TRUE | The trace has been logged in the event viewer. |
| FALSE | The trace cannot be logged in the event viewer because the Trace flag specified as parameter is not active. |

### Note

In order to set the trace flag bit you must use a SCADA Basic instruction in the application
TRACE (Mode, ProcessNumber, FlagString);

| | | |
|---|---|---|
| Mode: | Value 0 or 1 (0 - disable the bit mask | 1 - enable the bit mask) |
| ProcessNumber: | Value from11 to 18 (11: MgrToolkit1 | 18: MgrToolkit8) |
| FlagString: | Hexadecimal value string | |

TRACE (1,11,”400”); // Enable in MgrToolkit 1 the trace for SVMGR_FLAG_BIT10
TRACE (0,11,”C00”); // Disable in the MgrToolkit1 the trace for SVMGR_FLAG_BIT10 and SVMGR_FLAG_BIT11
The source of the message will be the SV Manager Toolkit id (MgrToolkit1 to MgrToolkit8).
Be carefull this instruction IS NOT supported in a macro animation.

## 5.61 IAPIMgr::TxtVarWrite

```
SVMGRAPI2 IAPIMgr::TxtVarWrite (
        LPCSTR          pszVarName,
        LPCSTR          pszValue,
        ULONG           ulClientHandle
        );
```

### Description

Write a variable named *pszVarName*.

This function is asynchronous. When the pending write command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnWriteCompleted method, specifying the result of the command.

### Parameters

*pszVarName* [in]

Contains the name of the variable to unadvise.

pszValue [in]

Value to write.

*ulClientHandle* [in]

Associated client handle for this read.

### Returns

| | |
|---|---|
| TRUE | Write  is pending. |
| FALSE | Variable cannot be written. Probably not present in the real-time database. |

## 5.62 IAPIMgr::UnMaskVar

```
SVMGRAPI2 IAPIMgr::UnMaskVar (
        _svmgrLevelMask     MaskLevel,
        LPCSTR              pszVarName,
        LPCSTR              pszOperator,
        ULONG               ulClientHandle
        );
```

### Description
Unmask variable named *pszVarName*.

This function is asynchronous. When the pending unmask command is completed, the SV Manager Toolkit invoke the IUsrMgr::OnUnMaskVarCompleted method, specifying the result of the command.

### Parameters
*MaskLevel* [in]

Contains the mask level to desactivate, it can be one of the following values.

```
svmgr_lvlUSERPROG1          // program level 1
svmgr_lvlUSERPROG2          // program level 2
svmgr_lvlUSERPROG3          // program level 3
svmgr_lvlUSERPROG4          // program level 4
svmgr_lvlOPERATOR           // operator
svmgr_lvlINHIB              // inhibited
svmgr_lvlALARM          // alarm
```

*pszVarName* [in]

Contains the name of the variable.

*pszOperator* [in]

Contains the name of the operator.

*ulClientHandle* [in]

Associated client handle for this request.

### Returns

| | |
|---|---|
| TRUE | Unmask is pending. |
| FALSE | Variable will be not unmask. Probably not present in the real-time database. |

## 5.63 IAPIMgr::UnsubscribeOPCItem

**SVMGRAPI2 IAPIMgr:: UnsubscribeOPCItem (**
**ULONG ulGroupServerHandle,**
**LPCTSTR pszItemID,**
**ULONG ulClientHandle**
**);**

### Description

Unsubscribe an OPC Item.

This function is asynchronous. When the pending command is completed, the SV Manager Toolkit invokes the IUsrMgr::OnUnsubscribeOPCItemCompleted method, specifying the result of the command.

### Parameters

*ulGroupServerHandle* [in]

Handle given by function OnAddOPCGroupCompleted.

*ulClientHandle* [in]

Associated client handle for this request. Related at IAPIMgr::SubscribeOPCItem.

### Returns

| | |
|---|---|
| TRUE | Successful. |
| FALSE | Unscription of variable failed |

### Example

```
void StopProject()
{
        if (g_ulGroupServerHandle /*linked into OnAddOPCGroupCompleted*/ != 0)
        {
                if (g_pSvAPI->UnsubscribeOPCItem(g_ulGroupServerHandle, 555 /*the same used
                in SubscribeOPCItem, is related at the item created*/) == FALSE)
                {
                        LogMessage("Unsubscribe OPC Item failed");
                }
         }
  }
```

## 5.64 IAPIMgr::VarAdvise

```
SVMGRAPI2 IAPIMgr::VarAdvise (
        LPCSTR          pszVarName,
        ULONG           ulClientHandle
        );

SVMGRAPI2 IAPIMgr::VarAdvise (
        LPCSTR                          pszVarName,
        _svmgrAdviseOptions &           sAdviseOptions,
        ULONG                           ulClientHandle
        );
```

### Description
Advise a variable named *pszVarName*. On value and status change of this variable, the SV Manager Toolkit invokes the IUsrMgr::OnDataChange or IUsrMgr::OnDataChange2 interface method with the variable change and the *ulClientHandle* associated to this advice.

### Parameters
*pszVarName* [in]

Contains the name of the variable to advise.

*sAdviseOptions* [in]

Advise options. Must be initialize by IAPIMgr::InitAdviseOptionsStructure and free by IAPIMgr::FreeAdviseOptionsStructure.

*ulClientHandle* [in]

Associated client handle for this advice.

### Returns
| | |
|---|---|
| TRUE | Variable is advised |
| FALSE | Variable is not advised. Probably not present in the real-time database. |

## Example

This example shows the way to advise 2 variables using a deadband of 1%.

```
void StartProject()
{
	_svmgrAdviseOptions options;

	// Initialize the options structure
	if ( g_pSvAPI->InitAdviseOptionsStructure( sizeof(_svmgrAdviseOptions), options))
	{
		// Fix the deadband at 1%
		options.dDeadbandValue = 1;

		// Advise variables with these options
		g_pSvAPI->VarAdvise( "TEST1", options, 1);
		g_pSvAPI->VarAdvise( "TEST2", options, 1);

		// Free the options structure
		g_pSvAPI->FreeAdviseOptionsStructure( options);
	}
}
```

## 5.65 IAPIMgr::VarEnum

```
SVMGRAPI2 IAPIMgr::VarEnum (
        HANDLE              hEnum,
        ULONG               ulNbRequestedVar,
        _svmgrVarEnum *     peVar,
        ULONG *             pulNbVar
        );
```

### Description

Get an enumerator of the real-time database variables.

At the first call of this function, you get the *ulNbRequestedVar* first variables, at each successive call, you get the next *ulNbRequestedVar* variables.

### Parameters

*hEnum* [in]
    Contains the enumerator handle.

*ulNbRequestedVar* [in]
    Indicates the maximum number of variables to enumerate in one call.

*peVar* [out]
    Contains the enumerated variables.
    For each enumerated variable, the associated item of the *peVar* array contains its name, its extended attributes and its type.

*pulNbVar* [out]
    Contains the number of enumerated variables.

### Returns

| | |
|---|---|
| TRUE | The enumeration isn't finished. There are variables in the real-time database that is not enumerated. |
| FALSE | The enumeration is finished. There are no more variables in the real-time database that is not enumerated. |

### Note

*The caller must allocate peVar* and its size must be coherent with *ulNbRequestedVar*. At each call of this function, you must free the *peVar* with the function IAPIMgr::ClearVarEnum.

### Example

**Example : How to use variable enumeration ?**

In this example, we get the enumeration of all the real-time database, and display in the output window of Microsoft Visual Studio the name of variables that have the first character of the 4th attributes set to 'E'.

```
	HANDLE hEnum;			// Handle of the enumerator.

	IAPIMgr *g_pSvAPI;	// SV Manager Toolkit interface pointer, provided by
svmgrExchangeInterface

	// First, we must create the enumerator.
	g_pSvAPI->CreateVarEnum( hEnum);

	_svmgrVarEnum peVar[ 100];	// Buffer of enumerated variables

	ULONG		ulNbVar;		// Number of enumerated variables

	BOOL		bResult;		// if FALSE, no more variables to
							// enumerate

	do
	{
		// we ask for the next 100 variables
		bResult = g_pSvAPI->VarEnum( hEnum, 100, peVar, &ulNbVar);

		// we check all the enumerated variables...
		for ( ULONG ulCmpt = 0; ulCmpt < ulNbVar; ulCmpt++)
		{
			// ...for the first character of the 4th attributs
			if ( *(peVar[ ulCmpt].szAttrib[ 1]) == 'E')
			{
				TRACE(peVar[ ulCmpt].szName) ;
			}
		}

		// Before asking the next 100 variables or quit the loop,
		// we must clear the memory used for the current enum.
		g_pSvAPI->ClearVarEnum( ulNbVar, peVar);
	}
	// continue while bResult is different from FALSE,
	// meaning that there are no more variables to enumerate
	while ( bResult == TRUE);

	// Last, we must destroy the enumerator.
	g_pSvAPI->CloseVarEnum( hEnum);
```

## 5.66 IAPIMgr::VarGetType

```
SVMGRAPI2 IAPIMgr::VarGetType (
        LPCSTR                  pszVarName
        _svmgrVarType &         vt
        );
```

### Description

Check the existence of a variable. The existence of a variable is the fact it is present in the real-time database.

### Parameters

*pszVarName* [in]

Contains the name of the variable to get the type.

*vt* [out]

If the function succeeds, contains the type of the requested variable. List and signification of values:

| | |
|---|---|
| - svmgr_vtLOG : | The requested variable is a logical type variable. |
| - svmgr_vtANA : | The requested variable is a analogic type variable. |
| - svmgr_vtTXT : | The requested variable is a text type variable. |
| - svmgr_vtALARM : | The requested variable is an alarm type variable. |

### Returns

| | |
|---|---|
| TRUE | The variable named *pszVarName* exists.<br>The parameter *vt* contains the type of the requested variable. |
| FALSE | The variable named *pszVarName* doesn’t exist.<br>The parameter *vt* is not consistent. |

## 5.67 IAPIMgr::VarIsExist

```
SVMGRAPI2 IAPIMgr::VarIsExist (
        LPCSTR        pszVarName
        );
```

### Description

Check the existence of a variable. The existence of a variable is the fact it is present in the real-time database.

### Parameters

*pszVarName* [in]

Contains the name of the variable to check the existence.

### Returns

| | |
|---|---|
| TRUE | The variable named *pszVarName* exists. |
| FALSE | The variable named *pszVarName* doesn’t exist. |

## 5.68 IAPIMgr::VarRead

```
SVMGRAPI2 IAPIMgr::VarRead (
        LPCSTR                        pszVarName,
        ULONG                         ulClientHandle
        _svmgrDataSource              dsSource = svmgr_dsCACHE
        );
```

### Description

Read a variable named *pszVarName*.

The **svmgrVarRead** function is asynchronous. When the pending read is completed, the SV Manager Toolkit invokes the IUsrMgr::OnReadCompleted or IUsrMgr::OnReadCompleted2 methods, specifying the result of this read command, and if successful, the value and its quality.

### Parameters

*pszVarName* [in]

Contains the name of the variable to unadvise.

*ulClientHandle* [in]

Associated client handle for this read.

*dsSource* [in]

Indicates what source to read: it can be **svmgr_dsCACHE** or **svmgr_dsDEVICE**. The data source **svmgr_dsCACHE** indicates the value returned in the IUsrMgr::OnReadComplete is the value in the real-time database. And, the data source **svmgr_dsDEVICE** indicates that a read command will be full executed: for example, if a variable is an equipment variable, the **svmgrVarRead** function will result in a read command on the device.

### Returns

| | |
|---|---|
| TRUE | Read is pending. |
| FALSE | Variable will be not read. Probably not present in the real-time database. |

### Note

Because the advising mode is more efficient (event mode), prefer use IAPIMgr::VarAdvise rather than IAPIMgr::VarRead (polling mode).

## 5.69 IAPIMgr::VarRecords

```
SVMGRAPI2 IAPIMgr::VarRecords (
        ULONG                ulSerieSize,
        _svmgrVarRecord *    pVarRecords
        );
```

### Description
Force the record (value, timestamp and quality) of a set of internal variables directly into the historical manager.

### Parameters
*ulSerieSize* [in]
  Contains the size of the array *pVarRecords*. This value cannot be greater than the value returns by the function IAPIMgr::GetVarRecordsMaxSize.

*pVarRecords* [in]
  Array of variables to record.

### Returns
| | |
|---|---|
| TRUE | The variable settings are pending. |
| FALSE | The variable settings have been refused. For example, array size is too large. |

### See Also
IAPIMgr::GetVarRecordsMaxSize

## 5.70 IAPIMgr::VarUnadvise

```
SVMGRAPI2 IAPIMgr::VarUnadvise (
        LPCSTR          pszVarName,
        ULONG           ulClientHandle
        );
```

### Description

Unadvise an advised variable named *pszVarName*.

### Parameters

*pszVarName* [in]

Contains the name of the variable to unadvise.

*UlClientHandle* [in]

Contains the client handle for the advice to abort. It's the same value that the *ulClientHandle* parameter of the IAPIMgr::VarAdvise function.

### Returns

| | |
|---|---|
| TRUE | Variable is unadvised |
| FALSE | Variable is not unadvised. Probably not present in the real-time database, or no advise currently associated with this variable. |

## 5.71 IAPIMgr::WriteOPCItem

**SVMGRAPI2 IAPIMgr:: WriteOPCItem (**
**ULONG ulGroupServerHandle,**
**LPCTSTR pszItemID,**
**const VARIANT & value,**
**ULONG ulClientHandle**
**);**

### Description

Write an OPC Item named *pszItemID* with the value *value*.

### Parameters

*ulGroupServerHandle* [in]
Handle given by function OnAddOPCGroupCompleted.

*pszItemID* [in]
Contains the name of the variable (TAG). Example "Device1.Tag1"

*value* [in]
Contains the value of the variable. The type is VARIANT.
Example:

```
VARIANT value;
VariantInit(&value);

V_VT(&value) = VT_I2; //Type 2 bytes signed integer
V_I2(&value) = (short)i;  // “i” as integer
```

*ulClientHandle* [in]
Associated client handle for this request.

### Returns

| | |
|---|---|
| TRUE | Successful. |
| FALSE | Write failed |

### Example

```
VARIANT v;
VariantInit(&v);

V_VT(&v) = VT_I2;
V_I2(&v) = (short)i; // “i” as integer

g_pSvAPI->WriteOPCItem(g_ulGroupServerHandle, "Device1.WriteTag2", v,1234);
```

# 6 Functions index

## 6.1 IUsrMgr functions index

## 6.2 IAPIMgr functions index

IAPIMgr::SetVariableAttribute
IAPIMgr::SetTimer
IAPIMgr::SqlConnectonStart
IAPIMgr::SqlConnectionStop
IAPIMgr::SqlConnectionTestConnection
IAPIMgr::SqlCmdCancelRequest
IAPIMgr::SqlCmdExecuteDataReader
IAPIMgr::SqlCmdExecuteNonQuery
IAPIMgr::SqlCmdExecuteScalar
IAPIMgr::SubscribeOPCItem
IAPIMgr::Trace
IAPIMgr::TxtVarWrite
IAPIMgr::UnMaskVar
IAPIMgr::UnsubscribeOPCItem
IAPIMgr::VarAdvise
IAPIMgr::VarEnum
IAPIMgr::VarGetType
IAPIMgr::VarIsExist
IAPIMgr::VarRead
IAPIMgr::VarUnadvise
IAPIMgr::VarRecords
IAPIMgr::WriteOPCItem