



SIMONE
SOFTWARE

SIMONE API
COM Object
November 2023

Contents

1	Introduction.....	2
2	Concepts.....	2
2.1	SIMONE API COM Object Data Types	3
2.2	Routine Calling Conventions.....	4
3	SIMONE API COM Extensions	4
3.1	Additional Types and Constants	4
3.2	Additional SIMONE API Functions.....	6
3.2.1	simone_api_set_simone_path.....	6
3.2.2	simone_set_runmode	6
3.3	SIMONE API Array Functions	7
4	SIMONE Network Edit Extensions	8
4.1	Additional Types and Constants	8
4.2	Additional / Changed functions.....	8
4.2.1	simone_nw_start_edit	8
4.3	Example	9
5	Examples.....	10
5.1	Visual Basic 6.....	10
5.2	Visual Basic Net.....	11
5.3	C#	13

1 Introduction

On the MS Windows platform the SIMONE API functions are also available as a standard COM Object. This allows all programming languages and environments capable of using COM objects to access all SIMONE API functions in a consistent manner.

Due to the nature of the underlying standard SIMONE API library, it is only possible to use a single instance of the SIMONE API COM Object inside an application independently. Even if it is possible to create multiple SIMONE API COM Objects do not expect to be able to use them independently. Any change of internal state of a COM Object will change the internal state of all other SIMONE API COM Objects as well. For example, it is not possible to select a network "NETWORK1" in one SIMONE API COM Object and another network "NETWORK2" in any other SIMONE API COM Object.

Additionally the SIMONE API COM Object - like the standard SIMONE API Library - does not allow for multithreading. .

2 Concepts

The most recent SIMONE API COM Object is implemented using 2 files

- COMSimoneApi.dll
- COMSimoneApi.tlb

There exists still the previous version of the SIMONE API COM Object which represents the state of older versions of the SIMONE API.

COMSimoneApi.dll contains all functionality as of Version 2.87 of the SIMONE API.

All versions of the SIMONE API COM Objects will be installed by the SIMONE Installation procedure into the "exe"-sub directory of the SIMONE installation directory. The SIMONE API COM Object needs to be registered manually in the system using the Windows tool regsvr32.

Due to the fact that you can have multiple SIMONE installations on a computer but only a single COM object of the same type, registering of a new SIMONE API COM Object from a new SIMONE Installation will overwrite the COM registration of any previous SIMONE installation.

But the SIMONE API COM Object allows specifying from which SIMONE installation the SIMONE API Library should be loaded using the function:

```
simone_api_set_simone_path (<path to SIMONE Installation Directory>)
```

The SIMONE API COM Object dynamically loads the simone_api.dll if a new COM Object is instantiated and either uses above information to locate the SIMONE API DLL or, if nothing was specified, relies on the operating systems search strategy to locate DLL's/shared libraries.

Additionally, to the path of the SIMONE installation to be used, the SIMONE API COM Object also allows to choose whether to use the LOCAL or REMOTE version of the SIMONE API by calling the API function:

```
simone_set_runmode (<Runmode>)
```

where <Runmode> can either be SIMONE_RUNMODE_LOCAL or SIMONE_RUNMODE_REMOTE.
 Above function has to be called prior to any other SIMONE API COM function.
 The default is to use the LOCAL SIMONE API.

2.1 SIMONE API COM Object Data Types

The SIMONE API COM Object uses slightly different data types than the original SIMONE API library especially to make it easier to use the SIMONE API from Visual Basic like languages.

The most important difference is that the SIMONE API COM Object expects values and variables of the COM type **Date** at any places where the SIMONE API description expects a **time_t** value and a **BSTR** (or **OLESTR**) where the API documentation claims a **char*** (a string).

The SIMONE API COM Object transparently translates **Date** and **BSTR** values into **time_t** and **char*** values when calling SIMONE API Routines and vice versa.

Due to the fact that the time interval a Date variable can hold exceeds the time interval a time_t variable can store the user should make sure to use only Date values in the range of 01/01/1970 until 01/19/2038 or otherwise the corresponding API routine may fail.

At any place the API documentation states to use the value "o" for specifying an "undefined" time value the user of the SIMONE API COM Object should supply a "o" Date value (Visual Basic Net requires the value #1/1/0100# which also works in VB6).

If the API returns an "undefined" time value "o" the SIMONE API COM Object returns the Date value #1/1/0100#.

All time/date values are interpreted as local time values!

Even if a BSTR is Unicode and can hold any Unicode character it will be translated by the SIMONE API COM Object into ANSI code to be able to supply the resulting string to the SIMONE API library. Therefore, a BSTR should only contain characters which can be translated to ANSI code.

At any place the API Documentation notes that an "empty or NULL" string can be supplied to omit the parameter the user of the SIMONE API COM Object should supply an empty BSTR ("") instead.

The following table shows which data types should be used in several programming languages and programming environments:

SIMONE API	Visual Basic 6	Visual Basic Net	C#
INT32	Long	Integer	int
float	Single	Single	Float
time_t	Date	Date	System.DateTime
char*	String	String	String

2.2 Routine Calling Conventions

The SIMONE API was initially developed to be called by the 'C' programming language and therefore the routine description is oriented to be called from 'C' programs. One prominent example for the API routine declaration oriented by the 'C' language is the fact that the length for output strings must be provided by the caller of a routine. This is not really required for a program using the SIMONE API COM Object and therefore the need for supplying any length parameters for output string parameters is eliminated.

For example the routine to retrieve a configuration item, `simone_get_config_item()` is declared in the API documentation as:

```
simone_get_config_item (section, name, value, value_len)
```

or in 'C' declaration:

```
simone_get_config_item (char *section, char *name, char *value,  
                        INT32 value_len)
```

but in the SIMONE API COM Object declares the routine simply as (Visual Basic Notation):

```
simone_get_config_item (Byval section as String, Byval Name as String,  
                        Byref value as String)
```

The SIMONE API COM Object itself makes sure to provide character buffers of appropriate size to the underlying SIMONE API Library.

Special attention needs the SIMONE API routines dealing with arrays of input and/or output. See chapter 3.3 for details.

3 SIMONE API COM Extensions

The following lists additional functions available in the SIMONE API COM Object but not in the standard SIMONE API Library.

3.1 Additional Types and Constants

The SIMONE API COM Object defines the following new types as aggregation of simple defines in the standard SIMONE API:

- ApiStatus - List of SIMONE API error codes
- OpenMode - modes for `simone_open()`
- SimoneFlags - list of most used SIMONE API flags
- ConditionFlags - specialised condition flags
- IncludeFlags - specialised flags in `simone_include()` ..
- ExecuteFlags - specialised flags for execute routines
- ArchiveFlags - flags for archive routines
- LicenseServerMode - valid values for license server modes
- ObjectTypes - list of defined object types in SIMONE

- RunTypes - list of available run types of scenarios
- SimoneUnit - list of predefined unit codes
- UnitTypes - list of defined unit types
- FunctionTypes - list of valid function types
- SimDataTypes - list of defined data types in SIMONE
- Interpolation - Interpolation modes for profiles
- Severity - severity codes for execution run messages
- Runmodes - defined API library run modes
- InitOptions - currently available init options
- RouteFlags - flags for routine `simone_make_path_from_object_set()`
- LogLevel - available log levels
- NetArchiveListInfo - field info identifier for `simone_net_archive_list_info()`
- PSIFlags - flags for function: `simone_prepare_system:info()`
- ReadMode - flags for function: `simone_set_readmode()`
which modifies the behaviour of the simone read functions.
- FilterFlags - flags for function: `simone_get_entry_set_filter()`
which allow to control the enumeration of scenario statements.
- FlagsVarIDEx - flags for function: `simone_var_id_ex()`
which allows to give hints about the type of the Simone object in the input string.
- FlagsExtinfo - flags for function: `simone_varid_info_ex()`

In Visual Basic 6 you can define variables of one the types above simply by using the type name like:

```
Dim status as ApiStatus
```

In Visual Basic Net or C# you have to prefix the type name with the identifier "Simone" like:

```
Dim status as Simone.ApiStatus
```

The second definition also works in VB6

The SIMONE API COM Object also defines some constants as `SIMONE_OPT_QT_YES` like the standard API Library. These constants have to be prefixed with "Simone.ApiConst" like in:

```
Dim f as Single
f = Simone.ApiConst.SIMONE_OPT_QT_YES
```

where "Simone" again can be omitted in VB6 and VBA.

If you observe the error:

error CS1752: Interop type 'APIConst' cannot be embedded.

trying to compile an application using constants from `APIConst`, change the properties of the referenced *Simone COM Object* and set **Embed Interop Types** to false!

3.2 Additional SIMONE API Functions

3.2.1 simone_api_set_simone_path

`simone_api_set_simone_path (simone_path)`

Parameters:

<code>simone_path</code>	- path to SIMONE installation	IN
--------------------------	-------------------------------	----

Return values

<code>simone_status_ok</code>	if SIMONE API Library was successfully loaded
-------------------------------	---

This routine allows specifying which SIMONE installation to be used. The SIMONE API COM Object will try to load the standard SIMONE API Library from the supplied installation if possible. If the supplied path was correct and the SIMONE API library was successfully loaded the routine returns `simone_status_ok`, otherwise `simone_status_not_found`.

If the routine returns `simone_status_not_found` calling of any other SIMONE API routine will fail with an exception.

This routine only works if it is called prior to any other SIMONE API routine with the exception of `simone_set_runmode()`. If it is called after any other API routine the routine will return `simone_status_ok` signalling success but does not change the SIMONE installation used.

If the routine is not called before any other API routine the SIMONE API COM Object tries to load the SIMONE API Library using the operating system standards for locating a DLL. If loading of the SIMONE API DLL fails in this case the called function raises an exception.

3.2.2 simone_set_runmode

`simone_set_runmode (runmode)`

Parameters:

<code>runmode</code>	- either <code>SIMONE_RUNMODE_LOCAL</code> or <code>SIMONE_RUNMODE_REMOTE</code>
----------------------	--

Return values:

<code>simone_status_ok</code>	if runmode was set correctly
<code>simone_status_badpar</code>	if the supplied parameter is invalid

This routine allows to specify whether the LOCAL or REMOTE version of the SIMONE API should be used. If called with SIMONE_RUNMODE_REMOTE the remote SIMONE API library is loaded allowing to connect to a SIMONE API Server anywhere in the network and calling SIMONE API functions transparently over the network.

The LOCAL SIMONE API used by default.

This routine can only be called once and needs to be called prior to all other SIMONE API functions especially `simone_api_set_simone_path()`. A second call to the routine is silently ignored.

3.3 SIMONE API Array Functions

Other than the SIMONE API Library the SIMONE API COM Object does not need explicit information about the size of supplied arrays for input or output. Instead, the supplied array(s) for input define their size implicitly. The size of the output arrays returned is automatically adjusted to the size of the input array.

The routine `simone_varid_array()` for example expects the size of the following array of variables as the first parameter. The SIMONE API COM Object version of `simone_varid_array()` omits the size parameter and directly starts with variable array. The supplied array implicitly defines their size. The output arrays for object ids and extension ids do not need to be dimensioned before explicitly. The size of the arrays is adjusted by the SIMONE API COM object to match the size of the input variable array.

The input arrays do not have to start with fixed first item at position 0 or 1. In fact the starting position can be chosen as appropriate. The layout of the output arrays will always be adapted to the layout of the input array(s).

All input arrays which should have the same size as stated in the SIMONE API documentation should have the same size and layout also in the SIMONE API COM Object.

The function `simone_varid_array()` for example might be called as follows (in Visual Basic 6 notation)

```
Dim varid (1 to 3) as String
Dim oids() As Long
Dim eids() As Long
Dim stats() As Long
Dim ss as ApiStatus
Dim api as Simone.Api6

varid(1) = "INP1.p"
varid(2) = "OUT9.p"
varid(3) = "OUT4.P"

ss = api.simone_varid_array (varid, SIMONE_NO_FLAG, oids, eids, stats)
```

The SIMONE API COM Object then will prepare and fill the arrays `oids`, `eids` and `stats` from index 1 up to index 3.

For C# arrays need to be defined as `System.Array` compatible. The above example could be written in C# as follows:

```
Array varid = new String [3] { "INP1.P", "OUT9.P", "OUT4.P"};
Array oids;
Array eids;
Array stats;
Simone.ApiStatus ss; Simone.Api2 api;
```

```
ss = api.simone_varid_array (varid, Simone.SimoneFlags.SIMONE_NO_FLAG,
oids, eids, stats);
```

4 SIMONE Network Edit Extensions

The SIMONE API COM Object V7 now supports all functions of the Network Edit Extensions described in the dedicated manual.

All functions of the Network Edit Extensions are encapsulated in a new COM Object named ICOMSimoneNetApi which has to be created using the function `simone_nw_start_edit()`. All functions of the Network Edit Extensions have to be called from this object.

There is only a single ICOMSimoneNetApi object and user should call the routine `simone_nw_start_edit()` only once!

The use of the Network Edit Extensions requires the same SIMONE license as the SIMONE Extended API.

4.1 Additional Types and Constants

The Network Edit Extensions has introduced the following new types compared to the standard SIMONE API:

- | | |
|--|------------------------------------|
| • NWMetricReferenceConditions | Metric Reference Conditions |
| • NWImperialReferenceConditions | Imperial Reference Conditions |
| • NWBTUReferenceConditions | BTU Reference Conditions |
| • NWHandleType | Type of Handle identifying objects |
| • NWElementParameters | Available Element Parameters |
| • NWElementPipeWidthMode | Mode used to described thickness |
| • of Element connection lines when drawing | |
| • NWGasQualityTrackingStyle | Gas Quality Tracking Style |
| • NWGasQualityType | Gas Quality Types |
| • NWLayerVisibilityMode | Modes for Layer Visibility |
| • settings | |
| • NWLabelType | Type of available Labels |
| • NWLocationType | Selects Coordinate System for |
| • Labels | |
| • NWLabelZOrder | Fore-/Background Attribute of |
| • Labels | |
| • NWLabelReferencePoint | Position of Label Reference Point |
| • NWLabelActionEvents | Available Action Events of Labels |
| • NWTextLabelLineAlignment | Alignment of Text Label Lines |

4.2 Additional / Changed functions

4.2.1 simone nw start edit

Compared to the standard SIMONE API the routine `simone_nw_start_edit()` has a different signature allowing to return the new `ICOMSimoneNetApi` object required to call all other functions of the Network Edit Extensions:

`simone_nw_start_edit (netapi, flags)`

Parameters:

Netapi	- ICOMSimoneNetApi Object	OUT
Flags	- reserved for future use	IN

Return values:

<code>simone_status_ok</code>	if ICOMSimoneApi object was created successfully
<code>simone_status_nolicense</code>	if no sufficient license was used.

4.3 Example

The following example using C# shows how to get an instance of the `ICOMSimoneNetApi` object and query some statistics for the active network model. Error checks are omitted due to readability.

```
api = new Simone.Api9();
if (api != null) {
    Simone.ApiStatus ss;
    Simone.SimoneFlags flags = Simone.SimoneFlags.SIMONE_NO_FLAG;

    ss = api.simone_init ("C:\\simone\\sys\\user.ini");
    ss = api.simone_select ("SIMPLE");

    Simone.ICOMSimoneNetApi netapi;
    ss = api.simone_nw_start_edit (out netapi, flags);

    int hnw;
    int node_count;
    String owner;

    ss = netapi.simone_nw_load_active (out hnw, flags);
    ss = netapi.simone_nw_node_get_count (hnw, out node_count, flags);
    ss = netapi.simone_nw_statistics_get_owner (hnw, out owner, flags);

    // enumerate all nodes
    int hnode;
    String name;
    ss = netapi.simone_nw_node_get_first (hnw, out hnode, flags);
    while (ss == Simone.ApiStatus.simone_status_ok) {
        ss = netapi.simone_nw_node_get_name (hnode, out name, flags);

        ss = netapi.simone_nw_handle_release (hnode);

        ss = netapi.simone_nw_node_get_next (hnw, out hnode, flags);
    }

    ss = netapi.simone_nw_handle_release (hnw, flags);
    ss = api.simone_end();
}
```

```
}
```

5 Examples

Following are short examples how to use the SIMONE API COM object in Visual Basic 6 and Visual Basic .Net. The examples assume that the SIMONE API COM Object is already installed and correctly registered in the Windows Registry as COM Object. If the COM Object is not correctly registered it will not be available as reference in VB6 or VBNET.

If the COM Object is not registered the registration can be done using the tool "regsvr.32" in the \windows\system32 directory.

5.1 Visual Basic 6

To be able to use the SIMONE API COM Object in Visual Basic 6 you have to add a reference to the "SIMONE-API COM Server" by selecting the menu entries "Project->References" and checking the list item "SIMONE-API COM Server".

After that create a new project, for example a standard exe project and paste the source code below into the Form_Load() routine.

The code below does not contain any error handling and has to be adjusted for the SIMONE installation path in `simone_api_set_simone_path()` and the configuration file to be used in `simone_init_ex()`. The network SAMPLE_CS and scenario LOADS should be available if the sample network were installed during the SIMONE installation.

Please note that the parameters supplied to routines of the SIMONE API COM Object must be typed exactly as defined by the COM Object. Therefore you should use separate `DIM` statement for any variable because the following defines one variable of type String (s2) and one of type Variant (s1)

```
Dim s1, s2 As String
```

Whereas

```
Dim s1 as String
Dim s2 as string
```

or

```
Dim s1 as String, s2 as String
```

defines both variables as Strings.

```
Dim api As Simone.api9
Dim status As Simone.ApiStatus
```

```
Dim version As Long
Dim rootdir As String
Dim objid As Long
Dim extid As Long
Dim value As Single
Dim msg As String
Dim d As Date

Set api = New Simone.Api9

If Not api Is Nothing Then

    ' select simone installation
    status = api.simone_api_set_simone_path("c:\simone ")

    ' get version of SIMONE API
    version = api.simone_api_version()

    ' initialise the SIMONE API
    status = api.simone_init_ex("c:\simone\sys\COM_API",
                               SIMONE_NO_FLAG)

    ' get configuration information
    status = api.simone_get_config_item("", "simone_root", rootdir)

    ' select a network
    status = api.simone_select("SAMPLE_CS")

    ' select a scenario
    status = api.simone_open("LOADS", SIMONE_MODE_READ)

    ' execute the scenario
    status = api.simone_execute(msg)

    ' get objid/extension for a variable
    status = api.simone_varid("INP1.P", objid, extid)

    ' read values for all times
    d = 0
    Do
        status = api.simone_set_next_rtime(d)
        If (status = simone_status_invtime) Then
            Exit Do
        Else
            status = api.simone_read(objid, extid, SIMONE_DEFAULT_UNIT,
                                    value)
        End If
    Loop

    ' close scenario
    status = api.simone_close()

    ' terminate API usage
    status = api.simone_end()
End If
```

5.2 Visual Basic Net

To use the SIMONE API COM Object in Visual Basic lets assume you created a new VBNet Console application project in Visual Studio.

After that you have to add a reference to the SIMONE-API COM Server by selecting "Project->Add Reference" and selecting the "SIMONE-API COM Server" from the tab strip labelled COM.

The code really looks comparable to the code for VB6 with the exception that constants like `simone_status_ok` and `SIMONE_NO_FLAG` has to be prefixed with the fully qualified source.

```
Module Module1
```

```
    Sub Main()
```

```
        Dim api As Simone.api9
        Dim status As Simone.ApiStatus
        Dim version As Long
        Dim rootdir As String
        Dim objid As Long
        Dim extid As Long
        Dim value As Single
        Dim msg As String
        Dim d As Date
```

```
        api = New Simone.api9
```

```
        If Not api Is Nothing Then
```

```
            ' select simone installation
            status = api.simone_api_set_simone_path("c:\simone")

            ' get version of SIMONE API
            version = api.simone_api_version()

            ' initialise the SIMONE API
            status = api.simone_init_ex("c:\simone\sys\COM_API",
                                       Simone.SimoneFlags.SIMONE_NO_FLAG)

            ' get configuration information
            status = api.simone_get_config_item("", "simone_root", rootdir)

            ' select a network
            status = api.simone_select("SAMPLE_CS")

            ' select a scenario
            status = api.simone_open("LOADS",
                                    Simone.OpenMode.SIMONE_MODE_READ)

            ' execute the scenario
            status = api.simone_execute(msg)

            ' get objid/extension for a variable
            status = api.simone_varid("INP1.P", objid, extid)

            ' read values for all times
            d = #1/1/0100#
            Do
                status = api.simone_set_next_rtime(d)
                If (status = Simone.ApiStatus.simone_status_invtime) Then
                    Exit Do
                Else
```

```

        status = api.simone_read(objid, extid,
                                Simone.SimoneUnit.SIMONE_UNIT_DEFAULT, value)
    End If
Loop

' close scenario
status = api.simone_close()

' terminate API usage
status = api.simone_end()
End If

End Sub

End Module

```

5.3 C#

To use the SIMONE API COM Object in C# lets assume you created a new C# Console application project in Visual Studio.

After that you have to add a reference to the SIMONE-API COM Server by selecting "Add Reference.." from the References item in the project explorer and selecting the "SIMONE-API COM Server" from the tab strip labelled COM.

```

using System;

namespace COMCSSample
{
    class COMSimoneApiSample
    {
        static void Main(string[] args)
        {
            Simone.Api9 api;
            Simone.ApiStatus status;
            int version;
            String rootdir;
            int objid, extid;
            float value;
            String msg;
            DateTime d;

            Console.WriteLine("SIMONE API COM Object test ...");

            api = new Simone.Api9();
            if (api != null) {

                // select simone installation
                status = api.simone_api_set_simone_path
                    ("c:\\simone ");

                // get version of SIMONE API
                version = api.simone_api_version();

                // initialise the SIMONE API

```

```

status = api.simone_init_ex
    ("c:\\simone\\sys\\COM_API",
    Simone.SimoneFlags.SIMONE_NO_FLAG);

// get configuration information
status = api.simone_get_config_item("", "simone_root",
    out rootdir);

// select a network
status = api.simone_select("SAMPLE_CS");

// select a scenario
status = api.simone_open("LOADS",
    Simone.OpenMode.SIMONE_MODE_READ);

// execute the scenario
status = api.simone_execute(out msg);

// get objid/extension for a variable
status = api.simone_varid("INP1.P", out objid, out extid);

// read values for all times
d = new DateTime(0);
do
{
    status = api.simone_set_next_rtime(ref d);
    if (status == Simone.ApiStatus.simone_status_ok)
    {
        status = api.simone_read(objid, extid,
            Simone.SimoneUnit.SIMONE_UNIT_DEFAULT,
            out value);
    }
} while (status == Simone.ApiStatus.simone_status_ok);

// close scenario
status = api.simone_close();

// terminate API usage
status = api.simone_end();
}

Console.WriteLine("press RETURN to finish");
Console.ReadLine();
}
}
}

```



LIWACOM
Informationstechnik GmbH

P.O. Box 102415

D-45024 Essen

Phone +49 2 01 17 03 8 - 0

Fax +49 2 01 17 03 8 - 0

E-mail postmaster@liwacom.de
