# *Sentinel SuperPro Microsoft COM Interface*

RAINBOW TECHNOLOGIES

October, 2002

**RAINBOW TECHNOLOGIES, INC.**
50 Technology Drive, Irvine, CA 92618
Telephone: (949) 450-7300, (800) 852-8569 Fax: (949) 450-7450

**RAINBOW TECHNOLOGIES LTD.**
4 The Forum, Hanworth Lane, Chertsey, Surrey KT16 9JX, United Kingdom
Telephone: (44) 1932 579200 Fax: (44) 1932 570743

**RAINBOW TECHNOLOGIES**
122, Avenue Charles de Gaulle, 92522 Neuilly-sur-Seine Cedex, France
Telephone: (33) 1 41 43 29 02 Fax: (33) 1 46 24 76 91

**RAINBOW TECHNOLOGIES GMBH**
Streiflacher Str. 7, Germering, D 82110, Germany
Telephone: (49) 89 32 17 98 15 Fax: (49) 89 32 17 98 50

Additional offices and distributors are located worldwide.

## *International Quality Standard Certification*

Rainbow Technologies, Inc. Irvine, CA facility has been issued the ISO 9001 certification, the globally recognized standard for quality, by Det Norske Veritas as of March 2002.
Certificate Number CERT-02982-2000-AQ-HOU-RABR2

# Table of Contents

# About This Document

This document contains information on using the Sentinel SuperPro Microsoft COM interface. It describes the interface requirements, build example, SuperPro client library APIs and their status codes.

## Conventions Used in This Document

Please note the following conventions regarding text this document:

| Convention | Meaning |
|---|---|
| `COURIER` | Denotes syntax, prompts and code examples. If bold, denotes text you type. |
| *Italics* | Text in italics denotes the parameter names, file names and directories, or for emphasis in notes and tips. |
| **Bold Lettering** | In procedures, words in boldface type represent keystrokes, menu items, window names or mouse commands. |

# Suggested References

Refer to the following documentation for more detailed information on Sentinel SuperPro.

| Document | What's in it ? |
|---|---|
| *Sentinel SuperPro 6.1 Developer's Guide*. | The detailed information about the product features and APIs. |
| *Sentinel SuperPro 6.3 Documentation Addendum* | Contains information about the product changes done since the 6.2 release. |

# Getting Help

If you have questions, need additional assistance, or encounter a problem, please contact Rainbow Technologies Technical Support using one of the methods listed in the following table:

**Rainbow Technologies Technical Support Contact Information**

| **Corporate Headquarters North America and South America** | |
|---|---|
| | Rainbow Technologies North America |
| Internet | http://www.rainbow.com/support.html |
| E-mail | techsupport@irvine.rainbow.com |
| Telephone | (800) 959-9954 (Monday – Friday, 6:00 a.m. – 6:00 p.m. PST) |
| Fax | (949) 450-7450 |
| **Australia and New Zealand** | |
| E-mail | techsupport@au.rainbow.com |
| Telephone | (61) 3 9820 8900 |
| Fax | (61) 3 9820 8711 |
| **China** | |
| E-mail | sentinel@isecurity.com.cn |
| Telephone | (86) 10 8266 3936 |
| Fax | (86) 10 8266 3948 |
| **France** | |
| E-mail | EuTechSupport@rainbow.com |
| Telephone | (33) 1 41.43.29.00 |
| Fax | +44 (0) 1932 570743 |
| **Germany** | |
| E-mail | EuTechSupport@rainbow.com |
| Telephone | 0183 RAINBOW (7246269) |
| Fax | +44 (0) 1932 570743 |
| **Taiwan and Southeast Asia** | |
| E-mail | techsupport@tw.rainbow.com |
| Telephone | (886) 2 2570-5522 |
| Fax | (886) 2 2570-1988 |
| **United Kingdom** | |
| E-mail | EuTechSupport@rainbow.com |
| Telephone | 0870 7529200 |
| Fax | +44 (0) 1932 570743 |
| **Countries not listed above** | |
| Please contact your local distributor for assistance. | |

# Interface Requirements

This section contains information on what is required to use this interface.

## Compiler Compatibility

This interface is compatible with the following compiler:

- Microsoft Visual C++ 6.0.

## Specific Requirements

The interface requires the following:

- Sentinel System Driver 5.41 or higher
- Sentinel SuperPro Server version 6.3 or higher (for network operations only; not required for cases where *direct-to-driver* communication takes place. Also note that in case of the *direct-to-driver* communication, the network APIs—RNBOsproGetSubLicense, RNBOsproSetProtocol and RNBOsproSetHeartBeat—will return an error whenever called.)

## Testing

This interface has been tested on the following platforms:

- Windows 98
- Windows NT
- Windows 2000
- Windows ME
- Windows XP (32-bit)

# Build Example Information

The following information can be used for building the example program given with this interface.

## Evaluation Program Files

| File Name | Description |
|---|---|
| **Files Used for the COM Component** | |
| *Build.bat* | Batch file used for building the COM library. |
| *RainbowSSP.dsp* | The Microsoft Visual C++ 6.0 project file for the COM library. |
| *RainbowSSP.dsw* | The Microsoft Visual C++ 6.0 project workspace for *RainbowSSP.dll.* |

| File Name | Description |
|---|---|
| *RainbowSSP.cpp* | Source file of the exported methods. |
| *RainbowSSP.h* | Contains the definitions for the interface. |
| *RainbowSSP.idl* | IDL for *RainbowSSP.dll.* |
| *RainbowSSP.mak* | Make file for the COM library. |
| *RainbowSSP.tlb* | Type library file. |
| *RainbowSSP_i.c* | Contains the actual definitions of the CLIDs and IIDs. |
| *Resource.h* | The resource file. |
| *Superpro.cpp* | Source file of *Csuperpro class.* |
| *Superpro.h* | Declaration of *Csuperpro class.* |
| *Superpro.rgs* | Registraton script file. |
| *Dlldata.c* | C file for a proxy DLL. |
| *Dlldatax.c* | wrapper for dlldata.c |
| *Dlldatax.h* | Header file for procy/stuff. |
| *RainbowSSP.def* | Contains the list of exported functions. |
| *RainbowSSPps.def* | Contains the list of exported functions for proxy/server DLL. |
| *RainbowSSP.rc* | Resource file for SSP COM. |
| *RainbowSSP_p.c* | File contains the proxy stub code. |
| *RainbowSSPCP.h* | Header file for proxy stub. |
| *RainbowSSPps.mk* | Make file for making the proxy/stuff DLL. |
| *Spromeps.h* | An include file for all the SuperPro APIs. |
| *StdAfx.h* | include file for standard system include files. |
| *StdAfx.cpp* | source file that includes just the standard includes. |
| *SuperproContainer.h* | interface for the CSuperproContainer class. |
| *SuperproContainer.cpp* | implementation of the CSuperproContainer class. |
| *Spromeps.lib* | The Sentinel SuperPro static link library interface. Ensure that the library is made with multi-threaded DLL. |
| *RainbowSSP.dll* | The Sentinel SuperPro COM library. |
| *SuperPro Microsoft COM Interface.pdf* | (this document) |
| **Files Used for the COM Client** | |
| *Visual Basic Client\FrmEnSrvr.frm* | A Visual Basic form for EnumServer. |
| *Visual Basic Client\FrmEnsrvr.frx* | A binary file for *frmEnSrvr.frm.* |
| *Visual Basic Client\FrmGetKeyInfo.frm* | A Visual Basic form for GetKeyInfo. |
| *Visual Basic Client\FrmGetKeyInfo.frx* | A binary file for *frmGetKeyInfo.frm.* |
| *Visual Basic Client\frmSetconserver.frm* | A Visual Basic form for SetContactServer. |
| *Visual Basic Client\frmSetconserver.frx* | A binary file for the *frmSetconserver.frm* form. |
| *Visual Basic Client\FrmSetProt.frm* | A Visual Basic form for the SetProtocol. |

| File Name | Description |
|---|---|
| *Visual Basic Client\FrmSetProt.frx* | A binary file for *frmSetProt.frm.* |
| *Visual Basic Client\Rnbo.ico* | An icon file for *sproeval.exe.* |
| *Visual Basic Client\SPROEVAL.frm* | The main form of the COM client. |
| *Visual Basic Client\SPROEVAL.frx* | A binary file for *Sproeval.frm.* |
| *Visual Basic Client\SPROEVAL.mak* | The make file used for building *sproeval.exe*. |
| *Visual Basic Client\SPROEVAL.vbp* | A Visual Basic project file for the COM client example program. |
| *Visual Basic Client\SPROEVAL.vbw* | A Visual Basic workspace file for the COM client example program. |
| *Visual Basic Client\Spromeps.bas* | The declare statements file required by Visual Basic to interface with the COM DLL. |

## Build Instructions for the COM Component

To build the COM Component use the input files listed in the "Files Used for the COM Component" section.

### Using the Batch File

1. Edit the batch file *build.bat* to modify the *VCVARS_PATH* variable and set the location of *vcvars32.bat*.
2. Type `build` on the command prompt to build the COM Component.

#### Output Files/Libraries

- *RainbowSSP.dll*
- *RainbowSSP.h*
- *RainbowSSP_i.c*

### Using the Project Workspace

1. Open *RainbowSSP.dsw* in Microsoft Visual C++ 6.0 IDE.
2. Click **Build RainbowSSP.dll** under the **Build** menu to build the example program**.**

***Note** : Unicode libraries will be required for building the COM Component with the Unicode Project setting.*

## Building VB Client for the COM Component

To build the VB Client use the input files listed in the “Files Used for the COM Client“ section.

***Note:*** *For your application ensure that the COM Component is registered prior to building the VB Client.*

1. Open the *sproeval.vbw* project file in the Visual Basic 6.0 IDE.
2. Click the **Make sproeval.exe** option from the **File** menu to build the executable.

# Sentinel SuperPro Interface APIs

## The API Packet

The COM methods acts as wrappers around the SuperPro APIs. The COM objects acts as intermediary for passing data back and forth so that one can use a compiler and communicate to the standard C- based library. Since the COM object internally handles the packet, the COM methods do not require the packet to be passed each time a call is made. Developers need not define the APIPACKET in their program code. The packet as defined below now resides in the COM layer.

### Packet Definition

```
typedef RB_DWORD[SPRO_APIPACKET_SIZE/sizeof(RB_DWORD)] RB_SPRO_APIPACKET;
typedef RBP_VOID RBP_SPRO_APIPACKET;
```

### Return Code

All the COM methods return S_OK on success. If you wish to obtain the error code returned by the lastly called API, use the get_LastError(short *pretVal) method as explained below. A list of SuperPro APIs error codes is given in the end of this document for reference.

The following APIs are supported by this interface:

***Note****: The equivalent C interface APIs are also included to provide information about the parameters passed. For information on the classes, constants and data types used, refer to the section on “Data Type, Constant and Class Definitions.”*

## FormatPacket

This API validates the packet based on its size.

***Note****: This API must be called before any other RNBOspro function.*

### Format

```
HRESULT __STDCALL FormatPacket();
```

### Parameters

The COM object maintains the packet internally. Hence, the packet will not be passed for any API as an argument.

### Equivalent C Interface API

```
SP_STATUS SP_API RNBOsproFormatPacket(RBP_SPRO_APIPACKET  packet,
                                      RB_WORD             packetSize );
```

## Initialize

This API initializes the packet.

### Format

```
HRESULT __STDCALL Initialize();
```

### Parameters

None.

### Equivalent C Interface API

```
SP_STATUS SP_API RNBOsproInitialize(RBP_SPRO_APIPACKET  packet);
```

## SetProtocol

This API registers the communication protocol of a client with the SuperPro server. It is called after initializing the packet and before the RNBOsproFindFirst API. If this API is not used, the default protocol remains TCP/IP.

This API will not work if the packet already has a license; and will return an SP_INVALID_OPERATION error.

### Format

```
HRESULT __STDCALL SetProtocol ( VARIANT protocol);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *protocol* | IN | VARIANT | The protocol chosen by a client for communication with the server.The valid values are:<br>NSPRO_TCP_PROTOCOL 1<br>NSPRO_IPX_PROTOCOL 2<br>NSPRO_NETBEUI_PROTOCOL 4<br>NSPRO_SAP_PROTOCOL 8† |

*† Service Advertising Protocol (SAP) is used for finding the key plugged in the Novell server through broadcast only.*

### Equivalent C Interface API

```
SP_STATUS SP_API  RNBOsproSetProtocol(RBP_SPRO_APIPACKET  packet,
                                      PROTOCOL_FLAG       protocol);
```

## SetContactServer

This API is used to set the SuperPro server to be contacted for a particular API packet. The contact server can be set as RNBO_STANDALONE, RNBO_SPN_DRIVER, RNBO_SPN_LOCAL, RNBO_SPN_BROADCAST, RNBO_SPN_ALL_MODES, RNBO_SPN_SERVER_MODES or as an IP or IPX address, NetBEUI name or the name of the computer.

This API will not work if the packet already has a license; and will return an SP_INVALID_OPERATION error.

### Format

```
HRESULT __STDCALL SetContactServer(VARIANT serverName);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *serverName* | IN | VARIANT | Any of the following reserved strings:<br>▪ RNBO_STANDALONE<br>▪ RNBO_SPN_DRIVER<br>▪ RNBO_SPN_LOCAL<br>▪ RNBO_SPN_BROADCAST<br>▪ RNBO_SPN_ALL_MODES<br>▪ RNBO_SPN_SERVER_MODES<br>▪ no-net†<br>▪ Or, name of the contact server (Servername/IP address/IPX address††) |

† *The no-net mode is deprecated. See the* Sentinel SuperPro 6.3 Documentation Addendum *for details.*

†† *The IPX address should be represented in the "xx-xx-xx-xx,xx-xx-xx-xx-xx-xx" format, for example 12-34-56-78,9A-BC-DE -F0-12-34.*

### Equivalent C Interface API

```
SP_STATUS SP_API RNBOsproSetContactServer(RBP_SPRO_APIPACKET packet,
                                          char               *serverName);
```

## FindFirstUnit

This API finds the first SuperPro key with the specified developer ID and gets a license.

### Format

```
HRESULT __STDCALL FindFirstUnit(VARIANT developerId);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *developerID* | IN | VARIANT | The developer ID of the SuperPro key. |

### Equivalent C Interface API

```
SP_STATUS SP_API RNBOsproFindFirstUnit(RBP_SPRO_APIPACKET packet,
                                       RB_WORD            developerID);
```

## GetContactServer

This API is used to return the SuperPro server set for a particular API packet.

### Format

```
HRESULT __STDCALL GetContactServer(VARIANT *pServerName);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *pServerName* | IN/OUT | VARIANT | A buffer in which the server name is copied. Developer must provide a buffer with sufficient memory. |

### Equivalent C Interface API

```
SP_STATUS SP_API RNBOsproGetContactServer (RBP_SPRO_APIPACKET  packet,
                                           char              *serverNameBuf,
                                           RB_WORD          serverNameBufSz);
```

## SetHeartBeat

This API customizes the heartbeat of a client. It has to be called only after FindFirst is called. It can be used to:

1. Set an infinite heartbeat for a client by setting the time to INFINITE_HEARTBEAT. In this case, the SuperPro server will not release the license acquired by a client, until ReleaseLicense is received by the server for this client.
2. Set the heartbeat to any value between MIN_HEARTBEAT to MAX_HEARTBEAT in multiples of 1 second.

If the API is not used, the default heartbeat setting is 120 seconds.

### Format

```
HRESULT __STDCALL SetHeartBeat(VARIANT heartBeatValue);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *heartBeatValue* | IN | VARIANT | A value that represents time in seconds. |

### Equivalent C Interface API

```
SP_STATUS SP_API  RNBOsproSetHeartBeat(RBP_SPRO_APIPACKET packet,
                                       RB_DWORD           heartBeatValue);
```

## FindNextUnit

This API finds the next SuperPro key based on the developer ID maintained in the APIPACKET. This API should not be called, unless FindFirstUnit has returned a successful value or, if the licenses available with the SuperPro server are exhausted (SP_NO_LICENSE_AVAILABLE).

If this API returns success, the system will release the license obtained by FindFirstUnit API call and will contain the data for the next SuperPro key. However, if this API returns an error value, the packet will be marked invalid.

To re-initialize the API packet, use FindFirstUnit and optionally, FindNextUnit depending upon the number of SuperPro keys found and the one your program accesses.

### Format

```
HRESULT __STDCALL FindNextUnit();
```

### Parameters

None.

### Equivalent C Interface API

```
SP_STATUS SP_API RNBOsproFindNextUnit(RBP_SPRO_APIPACKET  packet);
```

## Read

This API reads a word at the specified address of the SuperPro key identified by the API packet. On success, the data variable will contain information from the SuperPro key.

If SP_ACCESS_DENIED error code is returned , an attempt was made to read a non-readable (algorithm) word. For security reasons, algorithm words cannot be read.

### Format

```
HRESULT __STDCALL Read(VARIANT address, VARIANT *pData);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *address* | IN | VARIANT | The cell address to be read. |
| *pData* | IN/OUT | VARIANT | Contains the data read from the key. |

### Equivalent C Interface API

```
SP_STATUS SP_API RNBOsproRead(RBP_SPRO_APIPACKET packet,
                              RB_WORD            address,
                              RBP_WORD           data);
```

## ExtendedRead

This API reads the word and access code at the specified address of the SuperPro key identified by the API packet. On success, the data variable will contain the information from the SuperPro key and the access code variable will contain the access code. If SP_ACCESS_DENIED error code is returned , an attempt was made to read a non-readable (algorithm) word. For security reasons, algorithm words cannot be read.

### Format

```
HRESULT __STDCALL ExtendedRead(VARIANT   address,
                               VARIANT  *pData,
                               VARIANT  *pAccessCode);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *address* | IN | VARIANT | The cell address to be read. |
| *pData* | IN/OUT | VARIANT | Contains the data read from the key. |
| *pAccessCode* | IN/OUT | VARIANT | The associated access code returned. |

### Equivalent C Interface API

```
SP_STATUS SP_API RNBOsproExtendedRead(RBP_SPRO_APIPACKET packet,
                                      RB_WORD            address,
                                      RBP_WORD           data,
                                      RBP_BYTE           accessCode);
```

## Write

This API is used to write a word and its associated access code to the SuperPro key identified by the API packet.

Writing to the SuperPro key requires a write password. The word data is placed in the *data* variable and its associated access code is placed in the access code variable.

On success, the data and its associated access code are written to the specified word on the SuperPro key. If SP_ACCESS_DENIED error code is returned, either the write password was incorrect or an attempt was made to write/overwrite a locked cell.

The write API can be used only to write/overwrite words with an access code of 0. To overwrite words with other access codes, use the RNBOsproOverwrite API.

### Format

```
HRESULT __STDCALL Write(VARIANT    writePassword,
                        VARIANT    address,
                        VARIANT    data,
                        VARIANT    accessCode);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *writePassword* | IN | VARIANT | The write password of the key. |
| *address* | IN | VARIANT | The cell address to be written. |
| *data* | IN | VARIANT | Contains the word to write in the key. |
| *accessCode* | IN | VARIANT | Contains an access code associated with the word to write. |

### Equivalent C Interface API

```
SP_STATUS SP_API RNBOsproWrite(RBP_SPRO_APIPACKET  packet,
                               RB_WORD             writePassword,
                               RB_WORD             address,
                               RB_WORD             data,
                               RB_BYTE             accessCode);
```

## Overwrite

This API writes a word and its associated access code to the SuperPro key identified by the API packet.

Overwriting to the SuperPro key requires the write and overwrite passwords. The word data is placed in the data variable and its associated access code is placed in the access code variable. On success, the data and its associated access code are written to the specified word on the SuperPro key. If SP_ACCESS_DENIED error code is returned, the write password and/or the overwrite passwords were incorrect.

This API can be used to overwrite any word on the SuperPro key with an exception of the words at addresses 0-7.

### Format

```
HRESULT __STDCALL Overwrite(VARIANT  writePassword,
                            VARIANT  owp1,
                            VARIANT  owp2,
                            VARIANT  address,
                            VARIANT  data,
                            VARIANT  accessCode);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *writePassword* | IN | VARIANT | The write password of the key. |
| *owp1* | IN | VARIANT | The word 1 of the overwrite password. |
| *owp2* | IN | VARIANT | The word 2 of the overwrite password. |
| *address* | IN | VARIANT | The cell address to be written. |
| *data* | IN | VARIANT | Contains the word to write in the key. |
| *accessCode* | IN | VARIANT | Contains the access code associated with the word to write. |

### Equivalent C Interface API

```
SP_STATUS SP_API RNBOsproOverwrite(RBP_SPRO_APIPACKET packet,
                                   RB_WORD            writePassword,
                                   RB_WORD            overwritePassword1,
                                   RB_WORD            overwritePassword2,
                                   RB_WORD            address,
                                   RB_WORD            data,
                                   RB_BYTE            accessCode);
```

## Decrement

This API decrements the counter at the specified address of the SuperPro key identified by the API packet. If the API is successful, the counter is decremented by 1. Errors are returned if you try to decrement a locked or hidden word, the counter is already 0, the word at the address is not a counter or, the write password is incorrect.

If the counter is associated with an active algorithm and the counter is decremented to 0, the associated algorithm will be made inactive.

The counter and associated algorithm can appear in the SuperPro as:

| Address | Data |
|---|---|
| N – 2 | Counter |
| N – 1 | Counter |
| N | Algorithm Word 1 |
| N + 1 | Algorithm Word 2 |

If either or both counters exist, the counter is associated with the algorithm. This association will exist only for N = 0C, 14, 1C, 24, 2C, 34, 3C Hex. An algorithm can have both an associated password and associated counters. The counters can be used to make the algorithm inactive and the password can be used to make the algorithm active. See RNBOsproActivate.

### Format

```
HRESULT __STDCALL Decrement(VARIANT writePassword,
                            VARIANT address);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *writePassword* | IN | VARIANT | The write password of the key. |
| *address* | IN | VARIANT | The cell address of the counter to decrement. |

### Equivalent C Interface API

```
SP_STATUS SP_API RNBOsproDecrement(RBP_SPRO_APIPACKET packet,
                                   RB_WORD            writePassword,
                                   RB_WORD            address);
```

## Activate

This API is used to activate an inactive algorithm at the specified address of the SuperPro key identified by the API packet. If the API is successful, the algorithm is made active. Errors are returned if the write password is invalid, the activate password is invalid, or the address is not word 1 of an algorithm having an activation password.

The algorithm and associated password will appear in the SuperPro as:

| Address | Data |
|---|---|
| N | Algorithm Word 1 |
| N + 1 | Algorithm Word 2 |
| N + 2 | Activate Password 1 |
| N + 3 | Activate Password 1 |

The association will only exist for N = 08, 0C, 10, 14, 18, 1C, 20, 24, 28, 2C, 30, 34, 38, 3C Hex. An algorithm can have both an associated password and associated counters.The counters can be used to make an algorithm inactive and the password can be used to make an algorithm active. See RNBOsproDecrement.

### Format

```
HRESULT __STDCALL Activate(VARIANT  writePassword,
                           VARIANT  activatepw1,
                           VARIANT  activatepw2,
                           VARIANT  address);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *writePassword* | IN | VARIANT | The write password of the key. |
| *activatepw1* | IN | VARIANT | The first word of the activate password. |
| *activatepw2* | IN | VARIANT | The second word of the activate password. |
| *address* | IN | VARIANT | The cell address of the first word of an algorithm to activate. |

### Equivalent C Interface API

```
SP_STATUS SP_API RNBOsproActivate(RBP_SPRO_APIPACKET packet,
                                  RB_WORD            writePassword,
                                  RB_WORD            activatePassword1,
                                  RB_WORD            activatePassword2,
                                  RB_WORD            address);
```

## Query

This API is used to query an active algorithm at the specified address of the SuperPro key identified by the API packet.

The address should be the first word of an active algorithm. The query data parameter will point to the first byte of the data to be passed to an active algorithm. The length of the query data is specified in the length variable.

On success, the query response of the same length is placed in the buffer pointed by the response parameter. The last 4 bytes of the response will also be placed in the response32 variable.

Each query byte may contain any value varying from 0 to 255. Each response byte may also contain any value between 0-255. The length of the response will always be the same as the length of the query bytes. It is the programmer's responsibility to provide buffers with sufficient memory.

However, if the address is not the first word of an active algorithm, the return status will be SP_SUCCESS and the response buffer data will be the same as the query buffer data.

### Format

```
HRESULT __STDCALL Query(VARIANT   address,
                        VARIANT   queryData,
                        VARIANT  *pResponse,
                        VARIANT  *pResponse32,
                        VARIANT   length);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *address* | IN | VARIANT | The cell address of the word to query. |
| *queryData* | IN | VARIANT | A buffer containing the query string. |

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *pResponse* | IN/OUT | VARIANT | A buffer containing the returned response string. |
| *pResponse32* | IN/OUT | VARIANT | A variable containing the short response. |
| *length* | IN | VARIANT | The number of query bytes to be sent to an active algorithm and also the length of the response buffer. |

### Equivalent C Interface API

```
SP_STATUS SP_API RNBOsproQuery(RBP_SPRO_APIPACKET packet,
                               RB_WORD            address,
                               RBP_VOID           queryData,
                               RBP_VOID           response,
                               RBP_DWORD          response32,
                               RB_WORD            length);
```

## GetVersion

This API returns the driver's version number and type.

### Format

```
HRESULT __STDCALL GetVersion(VARIANT  *pMajor,
                             VARIANT  *pMinor,
                             VARIANT  *pRev,
                             VARIANT  *pOsType);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *pMajor* | OUT | VARIANT | The major version number returned. |
| *pMinor* | OUT | VARIANT | The minor version number returned. |
| *pRev* | OUT | VARIANT | The revision level returned. |
| *pOsType* | OUT | VARIANT | The operating system driver type. Currently defined types are:<br><br>1. DOS local driver<br>2. Windows 3.x local driver<br>3. Windows Win32s local driver<br>4. Windows 3.x system driver<br>5. Windows NT system driver<br>6. OS/2 system driver<br>7. Windows 95 system driver<br>8. NetWare local driver<br>9. QNX local driver |

### Equivalent C Interface API

```
SP_STATUS SP_API RNBOsproGetVersion(RBP_SPRO_APIPACKET  packet,
                                    RBP_BYTE            majVer,
                                    RBP_BYTE            minVer,
                                    RBP_BYTE            rev,
                                    RBP_BYTE            osDrvrType);
```

## GetHardLimit

This API is used to retrieve the maximum number of licenses supported by a key (the hard limit).

### Format

```
HRESULT __STDCALL GetHardLimit(VARIANT *pHardLimit);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *pHardLimit* | OUT | VARIANT | A buffer to hold the hard limit. |

### Equivalent C Interface API

```
SP_STATUS SP_API RNBOsproGetHardLimit(RBP_SPRO_APIPACKET packet,
                                      RBP_WORD           HardLimit);
```

## GetKeyInfo

This API is used to get information about the key attached to a SuperPro server.

### Format

```
HRESULT __STDCALL GetKeyInfo (VARIANT               developerId,
                              VARIANT               keyIndex,
                              NSPRO_MONITOR_INFO_T *pMonitorInfo);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *developerId* | IN | VARIANT | The developer ID of the key at the position specified by the *keyIndex* parameter. |
| *keyIndex* | IN | VARIANT | The index of the key whose information is being sought. |
| *pMonitorInfo* | IN/OUT | NSPRO_MONITOR_INFO_T | A variable of the class NSPRO_MONITOR_INFO_T that will hold information about the key. |

### Equivalent C Interface API

```
SP_STATUS SP_API RNBOsproGetKeyInfo(RBP_SPRO_APIPACKET  packet,
                                    RB_WORD             devId,
                                    RB_WORD             keyIndex,
                                    NSPRO_MONITOR_INFO *nsproMonitorInfo);
```

## GetFullStatus

This API is generally used for obtaining the status code of the last called API; however for COM client you need to call get_LastError to get the status code.

### Format

```
HRESULT __STDCALL GetFullStatus ();
```

### Parameters

None.

### Equivalent C Interface API

```
RB_WORD SP_API RNBOsproGetFullStatus(RBP_SPRO_APIPACKET  packet);
```

## GetSubLicense

This API is used to get a sublicense from the read-only data cell.

### Format

```
HRESULT __STDCALL GetSubLicense(VARIANT address);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *address* | IN | VARIANT | The address of the cell to get a sublicense from. |

### Equivalent C Interface API

```
SP_STATUS SP_API RNBOsproGetSubLicense(RBP_SPRO_APIPACKET packet,
                                      RB_WORD            address);
```

## ReleaseLicense

This API can be used in two ways:

1. To release the main license by specifying the cell address as zero.
2. To release the sublicense from a particular cell by specifying the cell address of the sublicensing cell as well as the number of sublicenses to be released.

### Format

```
HRESULT __STDCALL ReleaseLicense(VARIANT  address,
                                 VARIANT *pNumOfLic);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *address* | IN | VARIANT | If a sublicense is to be released, specify the sublicense cell number, otherwise specify 0. |
| *pNumOfLic* | IN/OUT | VARIANT | An integer array of length one whose first element will contain the number of sublicenses to be released. If the main license is to be released,  the first element can be set to zero. |

### Equivalent C Interface API

```
SP_STATUS SP_API  RNBOsproReleaseLicense(RBP_SPRO_APIPACKET packet,
                                        RB_WORD            address,
                                        RBP_WORD           numSubLic);
```

## EnumServer

This API is used to enumerate the number of SuperPro servers running in a subnet for the developer ID specified.

### Format

```
HRESULT __STDCALL EnumServer(VARIANT              eFlag,
                             VARIANT              devId,
                             NSPRO_SERVER_INFO_T *pServerInfo,
                             VARIANT             *pNumOfServers);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *eFlag* | IN | VARIANT | The flag used for contacting either:<br>- the first-found server that has licenses to offer (NSPRO_RET_ON_FIRST_AVAILABLE), or<br>- the first-found server that may have licenses (NSPRO_RET_ON_FIRST), or<br>- all the SuperPro servers in the network (NSPRO_GET_ALL_SERVERS). |
| *devid* | IN | VARIANT | The developer ID for the SuperPro device to find. Only the servers that have a key matching the developer ID will respond. If developer ID is specified as 0xFFFF, then all servers (for a specified protocol) in the subnet would respond. |
| *pServerInfo* | IN/OUT | NSPRO_SERVER_INFO_T | An array of structure NSPRO_SERVER_INFO_T that will hold the information about the server(s). |
| *pNumOfServers* | IN/OUT | VARIANT | An integer array of length one whose first statement will contain the number of servers to be found. The maximum value allowed is 10. |

### Equivalent C Interface API

```
SP_STATUS SP_API RNBOsproEnumServer(ENUM_SERVER_FLAG  enumFlag,
                                    RB_WORD           developerId,
                                    NSPRO_SERVER_INFO serverInfo,
                                    RBP_WORD          numServerInfo);
```

## get_LastError

It is used to get the status code of the the last SuperPro API call.

### Format

```
HRESULT __STDCALL get_LastError (short *pVal);
```

### Parameters

| Name | Direction | Parameter Type | Description |
|---|---|---|---|
| *pVal* | OUT | short | This will contain the status code for the last SuperPro API call. The desciption for these SuperPro API status code are given at the end. |

# Data Type, Constant and Class Definitions

## Data Types Defined in C Interface

- `typedef unsigned long  int*  RBP_DWORD;`
- `typedef unsigned long  int   RB_DWORD;`
- `typedef int                  ENUM_SERVER_FLAG;`
- `typedef unsigned short int*  RBP_WORD;`
- `typedef unsigned short int   RB_WORD;`
- `typedef unsigned short int   SP_STATUS;`
- `typedef unsigned short int   PROTOCOL_FLAG;`
- `typedef unsigned char*       RBP_BYTE;`
- `typedef unsigned char        RB_BYTE;`
- `typedef void*                RBP_VOID;`

## Constants

- `#define SP_SUCCESS          0`
- `#define SP_API`
- `#define MAX_ADDR_LEN        32`
- `#define MAX_NAME_LEN        64`

### Protocol Flag Definition

```
/* Set protocol flags */

#define NSPRO_TCP_PROTOCOL      1
#define NSPRO_IPX_PROTOCOL      2
#define NSPRO_NETBEUI_PROTOCOL  4
#define NSPRO_SAP_PROTOCOL      8
```

### Enumeration Flag Definition

```
/* Enum server flags */

#define NSPRO_RET_ON_FIRST            1
#define NSPRO_GET_ALL_SERVERS         2
#define NSPRO_RET_ON_FIRST_AVAILABLE  4
```

### Heartbeat Definition

```
/* Making the license update time programmable*/

#define MAX_HEARTBEAT        2592000         /* 30*24*60*60 seconds */
#define MIN_HEARTBEAT        60              /* 60 seconds */
#define INFINITE_HEARTBEAT   0xFFFFFFFF      /* Infinite heartbeat */
```

**Access Modes Definition**

```
/* To set an access modes for the protected application */

#define RNBO_STANDALONE         __TEXT("RNBO_STANDALONE")
#define RNBO_SPN_DRIVER         __TEXT("RNBO_SPN_DRIVER")
#define RNBO_SPN_LOCAL          __TEXT("RNBO_SPN_LOCAL")
#define RNBO_SPN_BROADCAST      __TEXT("RNBO_SPN_BROADCAST")
#define RNBO_SPN_ALL_MODES      __TEXT("RNBO_SPN_ALL_MODES")
#define RNBO_SPN_SERVER_MODES   __TEXT("RNBO_SPN_SERVER_MODES")
```

## NSPRO_KEY_MONITOR_INFO_T Structure

```
/* Information about the key */

struct NSPRO_KEY_MONITOR_INFO_T {
  [helpstring("Developer ID")]              VARIANT developerId;
  [helpstring("Hard Limit")]                VARIANT hardLimit;
  [helpstring("License in Used")]           VARIANT inUse;
  [helpstring("Number of Time Out")]        VARIANT numTimeOut;
  [helpstring("Highest Used")]              VARIANT highestUse;
} NSPRO_KEY_MONITOR_INFO_T;}
```

## NSPRO_SERVER_INFO_T Structure

```
/* The SuperPro server information with the number of licenses available */

struct NSPRO_SERVER_INFO_T {
  [helpstring("Server ip name")]               VARIANT  serverAddress;
  [helpstring("Number of license available")]  VARIANT  numOfLicense;
} NSPRO_SERVER_INFO_T;
```

# SuperPro API Status Codes

All the COM methods return S_OK on SUCCESS. However, the get_LastError(short *pretVal) method can be used to get the error code of lastly called API. The description of the error codes is given in the following table.

If you receive any unknown error numbers, please report the error number (extended error number if possible) to Rainbow Technologies Technical Support.

***Note:*** *The API Status Codes not listed below are obsolete even though they appear in the* spromeps.h *header file.*

| Status Code (Decimal) | Description |
|---|---|
| 0 | **SP_SUCCESS**<br>The fuction completed successfully. |
| 1 | **SP_INVALID_FUNCTION_CODE**<br>An invalid function code was specified. See your language's include file for valid API function codes. Generally, this error should not occur if you are using a Rainbow-provided interface to communicate with the driver. |

| Status Code (Decimal) | Description |
|---|---|
| 2 | **SP_INVALID_PACKET**<br>A checksum error was detected in the command packet, indicating an internal inconsistency. The packet structure structure may have been tampered with. Generally, this error should not occur if you are using a Rainbow-provided interface to communicate the driver. |
| 3 | **SP_UNIT_NOT_FOUND**<br>The specific unit could not be found. Make sure you are sending the correct information to find the unit. This error is returned by other functions if the unit has disappeared or unplugged. |
| 4 | **SP_ACCESS_DENIED**<br>You attempted to perform an illegal action on a word. For example, you may have tried to read an algorithm/hidden word, write to a locked word, or decrement a word that is not a data nor a counter word. |
| 5 | **SP_INVALID_MEMORY_ADDRESS**<br>You specified an invalid Sentinel SuperPro memory address. Valid addresses are 0-63 decimal(0-3F hex). Cells 0-7 are invalid for many operations. Algorithm descriptors must be referenced by the first (even) address. |
| 6 | **SP_INVALID_ACCESS_CODE**<br>You specified an invalid access code. The access code must be 0 (read/write date), 1 (read only data), 2 (counter), or 3 (algorithm/hidden). |
| 7 | **SP_PORT_IS_BUSY**<br>The port is busy in some other operation. |
| 8 | **SP_WRITE_NOT_READY**<br>The write or decrement action could not be performed due to a momentary lack of sufficient power. Attempt the operation again. |
| 9 | **SP_NO_PORT_FOUND**<br>No ports could be found on the workstation. |
| 10 | **SP_ALREADY_ZERO**<br>You tried to decrement a counter or data word that already contains the value 0. If you are using the counter to control demo prgram executions, this condition may occur after corresponding algorithm descriptor has been reactivated with its activation password. |
| 12 | **SP_DRIVER_NOT_INSTALLED**<br>The system device driver was not installed or detected. Communication with the unit was not possible. Please verify that the device driver is properly loaded. |
| 13 | **SP_IO_COMMUNICATIONS_ERROR**<br>The system device driver is having problems communicating with the unit. Please verify that the device driver is properly installed. |
| 15 | **SP_PACKET_TOO_SMALL**<br>The API packet is too small. |
| 16 | **SP_INVALID_PARAMETER**<br>The API packet contained an invalid parameter. |
| 18 | **SP_VERSION_NOT_SUPPORTED**<br>The current system device driver is outdated. Please update the system device driver. |
| 19 | **SP_OS_NOT_SUPPORTED**<br>The Operating System or environment is currently not supported by the client library. Please contact Rainbow Technical Support. |
| 20 | **SP_QUERY_TOO_LONG**<br>The maximum length of a query string supported is 56 characters. Retry with a shorter string. |
| 21 | **SP_INVALID_COMMAND**<br>An invalid SuperPro command was specified in the API call. |

| Status Code (Decimal) | Description |
|---|---|
| **30** | **SP_DRIVER_IS_BUSY**<br>The system device driver is busy. Try the operation again. |
| **31** | **SP_PORT_ALLOCATION_FAILURE**<br>Failed to allocate a parallel port through the Operating System's parallel port contention handler. |
| **32** | **SP_PORT_RELEASE_FAILURE**<br>Failed to release a previously allocated parallel port through the Operating System's parallel port contention handler. |
| **39** | **SP_ACQUIRE_PORT_TIMEOUT**<br>Failed to acquire access to a parallel port within the defined time-out. |
| **42** | **SP_SIGNAL_NOT_SUPPORTED**<br>The particular machine does not support a signal line. For example, an attempt was made to use the ACK line on a NEC 9800 computer. The NEC 9800 does not have an ACK line. Therefore, this error is reported. |
| **57** | **SP_INIT_NOT_CALLED**<br>Failed to call the client library's initialize API. This API must be called prior to the API that generated this error. |
| **58** | **SP_DRVR_TYPE_NOT_SUPPORTED**<br>The type of driver access, either direct I/O or system driver, is not supported for the defined Operating System and client library. |
| **59** | **SP_FAIL_ON_DRIVER_COMM**<br>The client library failed on communicating with a Rainbow system driver. |
| **60** | **SP_SERVER_PROBABLY_NOT_UP**<br>Server is not responding and the client has been timed out. |
| **61** | **SP_UNKNOWN_HOST**<br>Unknown server host. Server host does not seem to be on the network. Invalid hostname. |
| **62** | **SP_SENDTO_FAILED**<br>Client was unable to send message to the server. |
| **63** | **SP_SOCKET_CREATION_FAILED**<br>Client was unable to open network socket. Make sure the TCP/IP or IPX protocol stack is properly installed on the machine. |
| **64** | **SP_NORESOURCES**<br>Could not locate enough licensing resources. Insufficient resources (such as memory) are available to complete the request. An error occurred in attempting to allocate memory needed by function. |
| **65** | **SP_BROADCAST_NOT_SUPPORTED**<br>Broadcast is not supported by the network interface on the machine. |
| **66** | **SP_BAD_SERVER_MESSAGE**<br>Could not understand message received from the server. An error occurred in decrypting(or decoding) a server message at the client end. |
| **67** | **SP_NO_SERVER_RUNNING**<br>Cannot talk to the server. Verify server is running. No server seems to be running. Server on specified host is not available for processing the client request. |
| **68** | **SP_NO_NETWORK**<br>Unable to talk to the specified host. Network communcation problems encountered. |
| **69** | **SP_NO_SERVER_RESPONSE**<br>No server responded to client broadcast. Either there is no server running in the subnet or no server in the subnet has a desired key attached. Also can be the case, when a particular server (the contact server for that client) is not responding back. |
| **70** | **SP_NO_LICENSE_AVAILABLE**<br>All licenses are currently in use. Server has no more licenses available for this request. |

| **Status Code (Decimal)** | **Description** |
|---|---|
| **71** | **SP_INVALID_LICENSE**<br>The license is no longer valid. License expired due to timeout. |
| **72** | **SP_INVALID_OPERATION**<br>The specified operation cannot be performed. A license has already been issued for the given APIPACKET. Trying to set contact server after obtaining a license for the given APIPACKET, or trying to make another findfirst call. |
| **73** | **SP_BUFFER_TOO_SMALL**<br>The size of the buffer is not sufficient to hold the expected data. |
| **74** | **SP_INTERNAL_ERROR**<br>Internal error faced in licensing. |
| **75** | **SP_PACKET_ALREADY_INITIALIZED**<br>The given APIPACKET has already been initialized. |
| **76** | **SP_PROTOCOL_NOT_INSTALLED**<br>The protocol specified is not installed. |