

# ***Sentinel SuperPro Microsoft .NET (C#) Interface***



© Copyright 2002, Rainbow Technologies, Inc.  
All rights reserved.  
<http://www.rainbow.com>

All attempts have been made to make the information in this document complete and accurate. Rainbow Technologies, Inc. is not responsible for any direct or indirect damages or loss of business resulting from inaccuracies or omissions. The specifications contained in this document are subject to change without notice.

Sentinel SuperPro is a trademark of Rainbow Technologies, Inc. All other product names referenced herein are trademarks or registered trademarks of their respective manufacturers.

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

<b>About This Document</b>	<b>vi</b>
Conventions Used in This Document	vi
<b>Suggested References</b>	<b>vi</b>
<b>Getting Help</b>	<b>vii</b>
<b>Interface Requirements</b>	<b>1</b>
Compiler Compatibility	1
Specific Requirements	1
Testing	1
<b>Build Example Information</b>	<b>1</b>
Evaluation Program Files	1
Build Example Instructions	2
<b>Sentinel SuperPro Interface APIs</b>	<b>2</b>
The API Packet	2
RNBOsproFormatPacket	3
RNBOsproInitialize	3
RNBOsproSetProtocol	4
RNBOsproSetContactServer	4
RNBOsproFindFirstUnit	5
RNBOsproGetContactServer	6
RNBOsproSetHeartBeat	6
RNBOsproFindNextUnit	7
RNBOsproRead	7
RNBOsproExtendedRead	8
RNBOsproWrite	9
RNBOsproOverwrite	9
RNBOsproDecrement	10
RNBOsproActivate	11
RNBOsproQuery	12
RNBOsproGetVersion	13
RNBOsproGetHardLimit	14
RNBOsproGetKeyInfo	15
RNBOsproGetFullStatus	15
RNBOsproGetSubLicense	16
RNBOsproReleaseLicense	16
RNBOsproEnumServer	17
<b>Data Types, Constants and Structure Definitions</b>	<b>18</b>
Data Types Defined in C Interface	18
Constants	18
Query Definition	19
Monitoring Information Structure Definition	19
Server Information Structure Definition	20
<b>API Status Codes</b>	<b>20</b>

# About This Document

---

This document contains information on using the Sentinel SuperPro Microsoft .NET (C#) interface. It describes the interface requirements, build examples, 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.
<i>Italics</i>	Text in italics denotes the parameter names, file names and directories, or for emphasis in notes and tips.
<b>Bold Lettering</b>	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 about Sentinel SuperPro.

Document	What's in it ?
<i>Sentinel SuperPro 6.1 Developer's Guide.</i>	The detailed information about the product features and APIs.
<i>Sentinel SuperPro 6.3 Documentation Addendum</i>	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**

<b>Corporate Headquarters North America and South America</b>	
	Rainbow Technologies North America
Internet	<a href="http://www.rainbow.com/support.html">http://www.rainbow.com/support.html</a>
E-mail	<a href="mailto:techsupport@irvine.rainbow.com">techsupport@irvine.rainbow.com</a>
Telephone	(800) 959-9954 (Monday – Friday, 6:00 a.m. – 6:00 p.m. PST)
Fax	(949) 450-7450
<b>Australia and New Zealand</b>	
E-mail	<a href="mailto:techsupport@au.rainbow.com">techsupport@au.rainbow.com</a>
Telephone	(61) 3 9820 8900
Fax	(61) 3 9820 8711
<b>China</b>	
E-mail	<a href="mailto:sentinel@isecurity.com.cn">sentinel@isecurity.com.cn</a>
Telephone	(86) 10 8266 3936
Fax	(86) 10 8266 3948
<b>France</b>	
E-mail	<a href="mailto:EuTechSupport@rainbow.com">EuTechSupport@rainbow.com</a>
Telephone	(33) 1 41.43.29.00
Fax	+44 (0) 1932 570743
<b>Germany</b>	
E-mail	<a href="mailto:EuTechSupport@rainbow.com">EuTechSupport@rainbow.com</a>
Telephone	0183 RAINBOW (7246269)
Fax	+44 (0) 1932 570743
<b>Taiwan and Southeast Asia</b>	
E-mail	<a href="mailto:techsupport@tw.rainbow.com">techsupport@tw.rainbow.com</a>
Telephone	(886) 2 2570-5522
Fax	(886) 2 2570-1988
<b>United Kingdom</b>	
E-mail	<a href="mailto:EuTechSupport@rainbow.com">EuTechSupport@rainbow.com</a>
Telephone	0870 7529200
Fax	+44 (0) 1932 570743
<b>Countries not listed above</b>	
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 Studio 7.0 IDE (.NET)

## Specific Requirements

The interface also 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
<i>Activate.cs</i>	The source file for the Activate form.
<i>Activate.resx</i>	The Activate form.
<i>AssemblyInfo.cs</i>	The source file for the sproeval assembly information.
<i>EnumServer.cs</i>	The source file for the EnumServer form.
<i>EnumServer.resx</i>	The EnumServer form.

File Name	Description
<i>KeyInfo.cs</i>	The source file for the KeyInfo form.
<i>KeyInfo.resx</i>	The KeyInfo form.
<i>Password.cs</i>	The source file for the Password form.
<i>Password.resx</i>	The Password form.
<i>Rnbo.ico</i>	An icon file for <i>sproeval.exe</i> .
<i>Sproeval.csproj</i>	The project file.
<i>Sproeval.sln</i>	The project solution file.
<i>Sproeval_MainForm.cs</i>	The source file for the main form.
<i>Sproeval_MainForm.resx</i>	The main form.
<i>SuperPro.cs</i>	The source code that implements the calls to the SuperPro APIs.
<i>sx32w.dll</i>	The Sentinel SuperPro dynamic link library .
<i>Build.bat</i>	The batch file to build <i>sproeval.exe</i> .
<i>SuperPro Microsoft .NET (C#) Interface.pdf</i>	(this document).

## Build Example Instructions

1. Open the *Sproeval.sln* file in the Microsoft Visual Studio .NET IDE.
2. Select **Rebuild Solution** option given under the **Build** menu to build the executable.
3. Run *sproeval.exe* from its location. Ensure that *sx32w.dll* exists under the same directory as *sproeval.exe*.

---

**Tip!** In your application you need to use the *superpro.cs* and *sx32w.dll* files to call the SuperPro APIs.

---

## Sentinel SuperPro Interface APIs

### The API Packet

All APIs require an APIPACKET structure. It is an 4112-byte array whose internal structure is known only by the SuperPro client library. The SuperPro driver uses the data in the APIPACKET to communicate with the SuperPro key. A programmer should never modify the data in the APIPACKET.

#### Packet Definition

```
public const uint SPRO_APIPACKET_SIZE = 4112;
public static byte[] packet = new byte[SPRO_APIPACKET_SIZE];
```

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 structures, constants and data types used, refer to the section on “Data Type, Constant and Structure Definitions.”

---

## RNBOsproFormatPacket

This API initializes and validates the API packet based on its size.

---

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

---

### Format

```
public static extern ushort RNBOsproFormatPacket(byte[] packet,
                                                uint packetSize);
```

### Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.
<i>packetSize</i>	IN	uint	The size of the APIPACKET.

### Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

### Equivalent C Interface API

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

## RNBOsproInitialize

This API initializes the packet and sets the SuperPro server to be contacted, if anything is set in the NSP\_HOST environment variable.

### Format

```
public static extern ushort RNBOsproInitialize(byte[] packet);
```

### Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.

### Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

## Equivalent C Interface API

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

## RNBOsproSetProtocol

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

This API will not work if the API packet already has a license; in that case it will return an SP\_INVALID\_OPERATION error code.

### Format

```
public static extern ushort RNBOsproSetProtocol (byte[] packet,  
                                                int    protocol);
```

### Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.
<i>protocol</i>	IN	int	The protocol chosen by the client for communication with the SuperPro server. The valid values are:  NSPRO_TCP_PROTOCOL = 1 NSPRO_IPX_PROTOCOL = 2 NSPRO_NETBEUI_PROTOCOL = 3 NSPRO_SAP_PROTOCOL = 8 <sup>†</sup>

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

### Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

## Equivalent C Interface API

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

## RNBOsproSetContactServer

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 address, IPX address, NetBEUI name or the name of the machine.

This API will not work if the API packet already has a license; in that case it will return an SP\_INVALID\_OPERATION error code.

### Format

```
public static extern ushort RNBOsproSetContactServer (byte[] packet,  
                                                     [MarshalAs(UnmanagedType.LPStr)] string serverName );
```

## Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.
<i>serverName</i>	IN	string	Any of the following reserved strings: <ul style="list-style-type: none"><li>▪ RNBO_STANDALONE</li><li>▪ RNBO_SPN_DRIVER</li><li>▪ RNBO_SPN_LOCAL</li><li>▪ RNBO_SPN_BROADCAST</li><li>▪ RNBO_SPN_ALL_MODES</li><li>▪ RNBO_SPN_SERVER_MODES</li><li>▪ no-net<sup>†</sup></li></ul> Or, name of the contact server (Servername/IP address/IPX address <sup>††</sup> )

<sup>†</sup> The no-net mode is deprecated. See the Sentinel SuperPro 6.3 Documentation Addendum for details.

<sup>††</sup> The IPX address should be represented in the “xx-xx-xx-xx,xx-xx-xx-xx-xx” format, for example 12-34-56-78,9A-BC-DE-F0-12-34.

## Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

## Equivalent C Interface API

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

## RNBOsproFindFirstUnit

This API finds the first key attached to the SuperPro server with the specified developer ID and gets a license from the key. If RNBOsproFindFirstUnit is called with an API packet, which already has a license, then a SP\_INVALID\_OPERATION error code is returned.

## Format

```
public static extern ushort RNBOsproFindFirstUnit(byte[] packet,
                                                  ushort developerID);
```

## Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.
<i>developerID</i>	IN	Ushort	The developer ID of the SuperPro key.

## Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

## Equivalent C Interface API

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

## RNBOsproGetContactServer

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

### Format

```
public static extern ushort RNBOsproGetContactServer(byte[] packet,
                                                    byte[] serverNameBuf,
                                                    uint serverNameBufSz);
```

### Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.
<i>serverNameBuf</i>	OUT	byte[]	A buffer in which the SuperPro server name is copied. A developer needs to ensure that the buffers have sufficient memory.
<i>serverNameBufSz</i>	IN	uint	The length of the buffer. The maximum length recommended is up to 64 bytes.

### Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

### Equivalent C Interface API

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

## RNBOsproSetHeartBeat

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

1. To 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 RNBOsproReleaseLicense is received by the server for this client.
2. To 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

```
public static extern ushort RNBOsproSetHeartBeat(byte[] packet,
                                                  int heartBeatValue);
```

### Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.
<i>heartBeatValue</i>	IN	int	A value that represents time in seconds.

## Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

## Equivalent C Interface API

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

## RNBOsproFindNextUnit

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

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

To re-initialize the APIPACKET, use RNBOsproFindFirstUnit and optionally, RNBOsproFindNextUnit depending upon the number of SuperPro keys found and the one your program accesses.

## Format

```
public static extern ushort RNBOsproFindNextUnit(byte[] packet);
```

## Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.

## Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

## Equivalent C Interface API

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

## RNBOsproRead

This API reads a word at the specified address of the SuperPro key identified by the APIPACKET.

On success, the data variable will contain information from the SuperPro key. If the error code returned is SP\_ACCESS\_DENIED, an attempt was made to read a non-readable (algorithm) word. For security reasons, algorithm words cannot be read.

## Format

```
public static extern ushort RNBOsproRead(byte[] packet,
                                       ushort address,
                                       ref    ushort data);
```

## Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.
<i>address</i>	IN	ushort	The cell address to be read.
<i>data</i>	OUT	ushort	Contains the data read from the key.

## Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

## Equivalent C Interface API

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

## RNBOsproExtendedRead

This API reads the word and access code at the specified address of the SuperPro key identified by the APIPACKET.

On success, the data variable will contain the information from the SuperPro key and the access code variable will contain the access code. If the error code returned is SP\_ACCESS\_DENIED, an attempt was made to read a non-readable (algorithm) word. For security reasons, algorithm words cannot be read.

## Format

```
public static extern ushort RNBOsproExtendedRead(byte[] packet,  
                                                  ushort  address,  
                                                  ref ushort data,  
                                                  ref byte  accessCode);
```

## Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.
<i>address</i>	IN	ushort	The cell address to be read.
<i>data</i>	OUT	ushort	Contains the data read from the key.
<i>accessCode</i>	OUT	byte	The associated access code returned.

## Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

## Equivalent C Interface API

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



## RNBOsproWrite

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

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 error code is SP\_ACCESS\_DENIED, 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

```
public static extern ushort RNBOsproWrite(byte[] packet,
                                         ushort writePassword,
                                         ushort address,
                                         ushort data,
                                         byte accessCode);
```

### Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.
<i>writePassword</i>	IN	ushort	The write password of the key.
<i>address</i>	IN	ushort	The cell address to be written.
<i>data</i>	IN	ushort	Contains the word to write in the key.
<i>accessCode</i>	IN	byte	Contains the access code associated with the word to write.

### Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

### 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);
```

## RNBOsproOverwrite

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

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 the error code returned is SP\_ACCESS\_DENIED, 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

```
public static extern ushort RNBOsproOverwrite(byte[] packet,
                                             ushort writePassword,
                                             ushort overwritePassword1,
                                             ushort overwritePassword2,
                                             ushort address,
                                             ushort data,
                                             byte accessCode);
```

## Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.
<i>writePassword</i>	IN	ushort	The write password for the key.
<i>overwritePassword1</i>	IN	ushort	The word 1 of the overwrite password.
<i>overwritePassword2</i>	IN	ushort	The word 2 of the overwrite password.
<i>address</i>	IN	ushort	The cell address to be written.
<i>data</i>	IN	ushort	Contains the word to write in the key.
<i>accessCode</i>	IN	byte	Contains the access code associated with the word to write.

## Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

## 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);
```

## RNBOsproDecrement

This API decrements the counter at the specified address of the SuperPro key identified by the APIPACKET. 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

Address	Data
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

```
public static extern ushort RNBOsproDecrement(byte[] packet,
                                              ushort writePassword,
                                              ushort address);
```

## Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.
<i>writePassword</i>	IN	ushort	The write password for the key.
<i>address</i>	IN	ushort	The cell address of the counter to decrement.

## Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

## Equivalent C Interface API

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

## RNBOsproActivate

This API is used to activate an inactive algorithm at the specified address of the SuperPro key identified by the APIPACKET.

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

```
public static extern ushort RNBOsproActivate(byte[] packet,
                                             ushort writePassword,
                                             ushort activatePassword1,
                                             ushort activatePassword2,
                                             ushort address);
```

## Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte()	The APIPACKET defined earlier.
<i>writePassword</i>	IN	ushort	The write password for the SuperPro key.
<i>activatePassword1</i>	IN	ushort	The first word of the activate password.
<i>activatePassword2</i>	IN	ushort	The second word of the activate password.
<i>address</i>	IN	ushort	The cell address of the first word of an algorithm to activate.

## Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

## 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);
```

## RNBOsproQuery

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

The address should be the first word of an active algorithm. The query data variable 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 variable. 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 have 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

```
public static extern ushort RNBOsproQuery(byte[] packet,
                                           ushort address,
                                           byte[] queryData,
                                           byte[] response,
                                           ref uint response32,
                                           ushort length);
```

## Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The APIPACKET defined earlier.
<i>address</i>	IN	ushort	The cell address of the word to query.
<i>queryData</i>	IN	byte[]	The query data sent to the key.
<i>response</i>	OUT	byte[]	The response information received from the key.
<i>response32</i>	OUT	uint	The last 4 bytes of the query response.
<i>length</i>	IN	ushort	The number of query bytes to be sent to an active algorithm and also the length of the response buffer.

## Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

## 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);
```

## RNBOsproGetVersion

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

## Format

```
public static extern ushort RNBOsproGetVersion(byte[] packet,
                                                ref byte majVer,
                                                ref byte minVer,
                                                ref byte rev,
                                                ref byte osDrvrType);
```

## Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.
<i>majVer</i>	OUT	byte	The major version number returned.
<i>minVer</i>	OUT	byte	The minor version number returned.
<i>rev</i>	OUT	byte	The revision level returned.

Name	Direction	Parameter Type	Description
<i>osDrvrType</i>	OUT	byte	The operating system driver type. Currently defined types are: <ol style="list-style-type: none"> <li>1. DOS local driver</li> <li>2. Windows 3.x local driver</li> <li>3. Windows Win32s local driver</li> <li>4. Windows 3.x system driver</li> <li>5. Windows NT system driver</li> <li>6. OS/2 system driver</li> <li>7. Windows 95 system driver</li> <li>8. NetWare local driver</li> <li>9. QNX local driver</li> </ol>

### Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

### 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);
```

## RNBOsproGetHardLimit

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

### Format

```
public static extern ushort RNBOsproGetHardLimit(byte[] packet,
                                                  out ushort hardLimit);
```

### Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.
<i>hardLimit</i>	OUT	ushort	A buffer that will hold the hard limit.

### Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

### Equivalent C Interface API

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

## RNBOsproGetKeyInfo

This API is used to get information about a key from a particular SuperPro server.

### Format

```
public static extern ushort RNBOsproGetKeyInfo (byte[] packet,
                                                ushort devId,
                                                ushort keyIndex,
                                                byte[] nsproMonitorInfo);
```

### Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.
<i>devId</i>	IN	ushort	The developer ID of the key at the position specified by the <i>keyIndex</i> parameter.
<i>keyIndex</i>	IN	ushort	The index of the key whose information is being sought.
<i>nsproMonitorInfo</i>	OUT	byte[]	A buffer that contains the key information, including the developerID, hard limit and license use. A developer needs to ensure that the buffer has sufficient memory.

### Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

### Equivalent C Interface API

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

## RNBOsproGetFullStatus

This function is used for obtaining the return code of the last called API. It is provided for support purposes only.

### Format

```
public static extern ushort RNBOsproGetFullStatus(byte[] packet);
```

### Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.

### Return Code

Returns an integer value that can be interpreted by Rainbow's technical support department.

### Equivalent C Interface API

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

## RNBOsproGetSubLicense

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

### Format

```
public static extern ushort RNBOsproGetSubLicense (byte[] packet,
                                                    ushort address);
```

### Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.
<i>address</i>	IN	ushort	The address of the cell to get the sublicense from.

### Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

### Equivalent C Interface API

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

## RNBOsproReleaseLicense

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

```
public static extern ushort RNBOsproReleaseLicense(byte[] packet,
                                                    ushort address,
                                                    ref ushort numSubLic);
```

### Parameters

Name	Direction	Parameter Type	Description
<i>packet</i>	IN	byte[]	The API packet defined earlier.
<i>address</i>	IN	ushort	The cell address of the sublicense. If a sublicense is to be released, specify the cell number, otherwise specify <i>zero</i> .
<i>numSubLic</i>	IN/OUT	ushort	A variable containing the number of sublicenses to be released. If the main license is to be released, this can be specified as <i>zero</i> .



## Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

## Equivalent C Interface API

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

## RNBOsproEnumServer

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

## Format

```
public static extern ushort RNBOsproEnumServer(int enumFlag,
                                                ushort developerId,
                                                byte[] nsproServerInfo,
                                                ref ushort numServerInfo);
```

## Parameters

Name	Direction	Parameter Type	Description
<i>enumFlag</i>	IN	int	The flag used for contacting. <ul style="list-style-type: none"><li>▪ The first-found server that has a license to offer (NSPRO_RET_ON_FIRST_AVAILABLE);</li><li>▪ or, the first-found server that may have licenses (NSPRO_RET_ON_FIRST);</li><li>▪ or, all the servers in the network (NSPRO_GET_ALL_SERVERS).</li></ul>
<i>developerId</i>	IN	ushort	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.
<i>nsproServerInfo</i>	OUT	byte[]	A buffer that contains the SuperPro server information, such as the server address and the number of licenses available. A developer needs to ensure that the buffer has sufficient memory.
<i>numServerInfo</i>	IN/OUT	ushort	An in-out parameter that contains the desired number of the SuperPro servers to be enumerated. When the function returns, this variable contains the actual number of servers found on the network.

## Return Code

On success, returns SP\_SUCCESS. Else, returns an error code as defined in the “API Status Codes” section at the end of this document.

## Equivalent C Interface API

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

# Data Types, Constants and Structure Definitions

This section provides information about the data types, constants and structures used in this document.

## Data Types Defined in C Interface

Definition	Data Type in C	Data Type in C# (.NET)
RB_DWORD	unsigned long int	Int32
ENUM_SERVER_FLAG	int	Int16
SP_STATUS	unsigned short int	ushort
PROTOCOL_FLAG	unsigned short int	ushort
RB_BYTE	unsigned char	byte
RBP_DWORD	unsigned long int*	--
RBP_WORD	unsigned short int*	--
RBP_BYTE	unsigned char*	--
RBP_VOID	void*	--

-- Not Applicable

## Constants

- public const int MAX\_NAME\_LEN = 64; /\* Maximum host name length \*/
- public const int MAX\_ADDR\_LEN = 32; /\* Maximum host address length \*/

## Protocol Constants

```
/* To set the communication protocol flags */  
  
public enum PROTOCOL_FLAG  
{  
    NSPRO_TCP_PROTOCOL      = 1,  
    NSPRO_IPX_PROTOCOL      = 2,  
    NSPRO_NETBEUI_PROTOCOL  = 4,  
    NSPRO_SAP_PROTOCOL      = 8  
}
```

## Heartbeat Constants

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

public const int MAX_HEARTBEAT          = 2592000;
public const int MIN_HEARTBEAT          = 60;
public const int INFINITE_HEARTBEAT     = -1;
```

## Enumeration Flag Constants

```
/* Flags to specify the way of enumerating servers */

public enum ENUM_SERVER_FLAG
{
    NSPRO_RET_ON_FIRST                = 1,
    NSPRO_GET_ALL_SERVERS             = 2,
    NSPRO_RET_ON_FIRST_AVAILABLE     = 4
}
```

## Access Modes

```
/* To set the access modes */

public const string RNBO_STANDALONE     = "RNBO_STANDALONE";
public const string RNBO_SPN_LOCAL      = "RNBO_SPN_LOCAL";
public const string RNBO_SPN_DRIVER     = "RNBO_SPN_DRIVER";
public const string RNBO_SPN_BROADCAST = "RNBO_SPN_BROADCAST";
public const string RNBO_SPN_ALL_MODES  = "RNBO_SPN_ALL_MODES";
public const string RNBO_SPN_SERVER_MODES = "RNBO_SPN_SERVER_MODES";
```

## Query Definition

```
public const int SPRO_MAX_QUERY_SIZE    = 56;
```

## Monitoring Information Structure Definition

```
/* Information about the key, used as a part of NSPRO_KEY_MONITOR_INFO
structure */
```

```
public class NSPRO_KEY_MONITOR_INFO
{
    public ushort developerId;
    public ushort hardLimit;
    public ushort inUse;
    public ushort numTimeOut;
    public ushort highestUse;
}
```

```
/* Provides information of the server with key details */
```

```
public class NSPRO_MONITOR_INFO
{
    public byte[]          serverName;
    public byte[]          serverIPAddress;
    public byte[]          serverIPXAddress;
    public byte[]          version;
    public ushort          protocol;
    public NSPRO_KEY_MONITOR_INFO sproKeyMonitorInfo;
```

```

public NSPRO_MONITOR_INFO()
{
    this.serverName           = new byte[Superpro.MAX_NAME_LEN];
    this.serverIPAddress      = new byte[Superpro.MAX_ADDR_LEN];
    this.serverIPXAddress     = new byte[Superpro.MAX_ADDR_LEN];
    this.version              = new byte[Superpro.MAX_NAME_LEN];
    this.protocol              = 0;
    this.sproKeyMonitorInfo    = new NSPRO_KEY_MONITOR_INFO();
    this.sproKeyMonitorInfo.developerId = 0;
    this.sproKeyMonitorInfo.hardLimit = 0;
    this.sproKeyMonitorInfo.inUse = 0;
    this.sproKeyMonitorInfo.numTimeOut = 0;
    this.sproKeyMonitorInfo.highestUse = 0;
}
}

```

## Server Information Structure Definition

```

/* Server information with the number of licenses available */

public class NSPRO_SERVER_INFO
{
    public byte[] serverAddress;
    public ushort numLicAvail;

    public NSPRO_SERVER_INFO()
    {
        serverAddress = new byte[Superpro.MAX_ADDR_LEN];
        numLicAvail    = 0;
    }
}

```

## API Status Codes

On success, all APIs discussed earlier return SP\_SUCCESS. Else, they return an error code defined below. This section enumerates all the recoverable API status codes with their description. However, if you receive any unknown error codes, please report it to Rainbow Technologies Technical Support.

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

Status Code (Decimal)	Description
0	<b>SP_SUCCESS</b> The fuction completed successfully.
1	<b>SP_INVALID_FUNCTION_CODE</b> 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.
2	<b>SP_INVALID_PACKET</b> 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.

Status Code (Decimal)	Description
3	<b>SP_UNIT_NOT_FOUND</b> 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	<b>SP_ACCESS_DENIED</b> 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	<b>SP_INVALID_MEMORY_ADDRESS</b> 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	<b>SP_INVALID_ACCESS_CODE</b> You specified an invalid access code. The access code must be 0 (read/write data), 1 (read only data), 2 (counter), or 3 (algorithm/hidden).
7	<b>SP_PORT_IS_BUSY</b> The port is busy in some other operation.
8	<b>SP_WRITE_NOT_READY</b> The write or decrement action could not be performed due to a momentary lack of sufficient power. Attempt the operation again.
9	<b>SP_NO_PORT_FOUND</b> No ports could be found on the workstation.
10	<b>SP_ALREADY_ZERO</b> You tried to decrement a counter or data word that already contains the value 0. If you are using the counter to control demo program executions, this condition may occur after corresponding algorithm descriptor has been reactivated with its activation password.
12	<b>SP_DRIVER_NOT_INSTALLED</b> 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	<b>SP_IO_COMMUNICATIONS_ERROR</b> The system device driver is having problems communicating with the unit. Please verify that the device driver is properly installed.
15	<b>SP_PACKET_TOO_SMALL</b> The API packet is too small.
16	<b>SP_INVALID_PARAMETER</b> The API packet contained an invalid parameter.
18	<b>SP_VERSION_NOT_SUPPORTED</b> The current system device driver is outdated. Please update the system device driver.
19	<b>SP_OS_NOT_SUPPORTED</b> The Operating System or environment is currently not supported by the client library. Please contact Rainbow Technical Support.
20	<b>SP_QUERY_TOO_LONG</b> The maximum length of a query string supported is 56 characters. Retry with a shorter string.
21	<b>SP_INVALID_COMMAND</b> An invalid SuperPro command was specified in the API call.
30	<b>SP_DRIVER_IS_BUSY</b> The system device driver is busy. Try the operation again.
31	<b>SP_PORT_ALLOCATION_FAILURE</b> Failed to allocate a parallel port through the Operating System's parallel port contention handler.

Status Code (Decimal)	Description
32	<b>SP_PORT_RELEASE_FAILURE</b> Failed to release a previously allocated parallel port through the Operating System's parallel port contention handler.
39	<b>SP_ACQUIRE_PORT_TIMEOUT</b> Failed to acquire access to a parallel port within the defined time-out.
42	<b>SP_SIGNAL_NOT_SUPPORTED</b> 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	<b>SP_INIT_NOT_CALLED</b> Failed to call the client library's initialize API. This API must be called prior to the API that generated this error.
58	<b>SP_DRV_TYPE_NOT_SUPPORTED</b> The type of driver access, either direct I/O or system driver, is not supported for the defined Operating System and client library.
59	<b>SP_FAIL_ON_DRIVER_COMM</b> The client library failed on communicating with a Rainbow system driver.
60	<b>SP_SERVER_PROBABLY_NOT_UP</b> Server is not responding and the client has been timed out.
61	<b>SP_UNKNOWN_HOST</b> Unknown server host. Server host does not seem to be on the network. Invalid hostname.
62	<b>SP_SENDTO_FAILED</b> Client was unable to send message to the server.
63	<b>SP_SOCKET_CREATION_FAILED</b> Client was unable to open network socket. Make sure the TCP/IP or IPX protocol stack is properly installed on the machine.
64	<b>SP_NORESOURCES</b> 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	<b>SP_BROADCAST_NOT_SUPPORTED</b> Broadcast is not supported by the network interface on the machine.
66	<b>SP_BAD_SERVER_MESSAGE</b> Could not understand message received from the server. An error occurred in decrypting(or decoding) a server message at the client end.
67	<b>SP_NO_SERVER_RUNNING</b> 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	<b>SP_NO_NETWORK</b> Unable to talk to the specified host. Network communication problems encountered.
69	<b>SP_NO_SERVER_RESPONSE</b> 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	<b>SP_NO_LICENSE_AVAILABLE</b> All licenses are currently in use. Server has no more licenses available for this request.
71	<b>SP_INVALID_LICENSE</b> The license is no longer valid. License expired due to timeout.
72	<b>SP_INVALID_OPERATION</b> 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.

Status Code (Decimal)	Description
73	<b>SP_BUFFER_TOO_SMALL</b> The size of the buffer is not sufficient to hold the expected data.
74	<b>SP_INTERNAL_ERROR</b> Internal error faced in licensing.
75	<b>SP_PACKET_ALREADY_INITIALIZED</b> The given APIPACKET has already been initialized.
76	<b>SP_PROTOCOL_NOT_INSTALLED</b> The protocol specified is not installed.