# Swing ActiveX Controls

User's Manual

Jazz Soft

# 1 History

| Revision | Date | Description |
|---|---|---|
| 1.00 | May,27th,1999 | The first edition. |
| 2.71 | May,10th,2001 | Converted file format from HTML to Microsoft™ Word Document. |
| 3.10 | Feb,15th,2002 | Established 'Media Version'. |
| 3.13 | Oct,23rd,2002 | More faster processing. |
| 3.16 | Nov,13th,2003 | Added some functions for HSMS. |
| 3.17 | Jun,11th,2004 | Translated to English. |

# 2 What's New

- Process more faster for huge message.
- Media version now works on Windows XP.
- Support multi-byte character set of SECS-II.
- Added T7 time out.

# 3 Preface

Swing ActiveX Control Media version uses for protecting with hardware key, however Net version uses software password. Software password may disappear in case of hard ware crush or reinstallation of Windows O/S.

Feb, 15th, 2002
Hikaru Okada

# 4 Table Of Contents

# 5 System Requirements

It depends upon type of HASP key.

**For HASP USB**

- Windows 98 (USB Support), Windows Me, Windows 2000 or Windows XP.
- 32-bit ActiveX compatible developing language like Visual Basic or Visual C++.

**For HASP Parallel port**

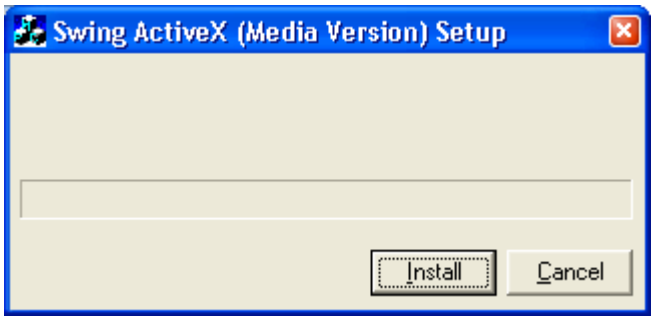
- Windows 95, Windows 98, Windows Me, WindowsNT 4.00, Windows 2000 or Windows XP。
- 32-bit ActiveX compatible developing language like Visual Basic or Visual C++.

# 6 Installation

All of the installation files for Swing are compressed as zip format and need to decompress. Some decompress tools for zip file are free to use. If you don't have it, search and download from appropriate web site,

## 6.1 To Installation Swing ActiveX Control

First, decompress zip file to some folder in your hard drive. To start installation, run setup.exe.




Press 'Install' button to start installation. You don't have to fill in the password any more.


When you are using evaluation version, following dialog box will be appear frequently. There are no differences between evaluation version and registered version.

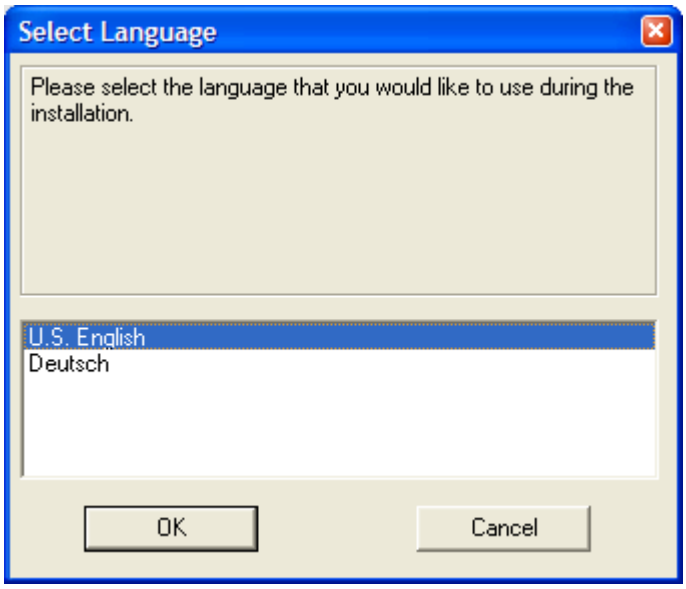

## 6.2 Install HASP Driver

If it is a new installation of Swing Media version, you must install HASP drivers as well.

(1) Close every running software.

(2) **Never insert HASP key in case of using USB type key! If you use Parallel port type key, insert HASP key at this time.** This procedure is very important, so do properly.[1]

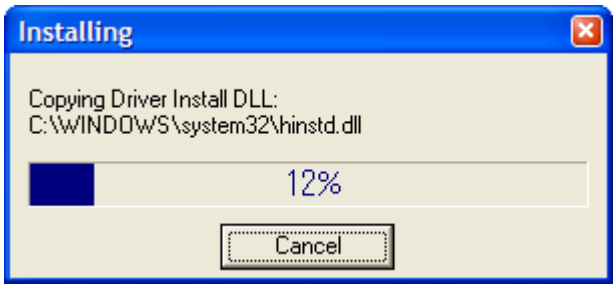
(3) Run hdd32.exe. Then following dialog box will be appeared.




(4) Select 'U.S. English', then press OK button.


(5) Press Next button to begin to copy driver files.

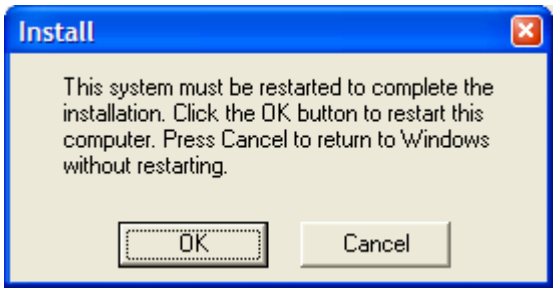
[1] If you made mistake, please try to install HASP driver again.




(6) If copy has been finished, the installation status dialog box will be appeared as follows.


(7) Press Next button to install HASP driver into Windows system. This process may take several minutes. If the installation has been finished, following dialog box will be appeared.


(8) If the system requires to restart Windows system, you can see following dialog box.



# 7 Architecture

The best language to develop application software cooperate with Swing is Visual Basic. It is very easy and intuitive to access 'properties', so there would be less bugs. Followings are the same procedure written in Visual Basic and written in Visual C++. This program opens Communication port #1 with 9600bps and show error message box in case of failure.

❑ Visual Basic

```
With SwingSecsI1
  .CommPort = 0
  .BaudRate = 9600
  .Active = True
  If Not .Active Then
    MsgBox "Error : Cannot open communication port!"
  End If
End With
```

❑ Visual C++

```
m_secs.SetCommPort(0);
m_secs.SetBaudRate(9600);
m_secs.SetActive(true);
if(!m_secs.GetActive())
  MessageBox("Error : Cannot open communication port!");
```

The line number of the source code to use Visual C++ is less than to use Visual Basic. But Visual Basic can use 'With' to omit object, so the code itself is a very good readability.

At this time, let's see the way to access to Active Property. It is completely different between 'set' value and 'get' value for this property. If you 'set' value, it means to open or close action. However, if you 'get' value, it means to inspect as opened or not. If you write codes in Visual C++, you must write very distinctively.

❑ Visual C++

```
m_secs.SetActive(true);                ' Open
if(!m_secs.GetActive())                ' Is it opened?
```

But it is ambiguous to write in Visual Basic.

❑ Visual Basic

```
.Active = True                         ' Open
If Not .Active Then                    ' Is it opened?
```

How can I distinct which is 'set' and which is 'get'? The answer is a very simple. If property description is the right hand of '=' mark, it means 'get', and left hand means 'set'. It must be very easy if we re-write as follows;

❑ Visual Basic

```
Dim bResult As Boolean
bResult = .Active
If Not bResult Then                    ' Is it opened?
```

There is no doubt that Active Property is placed in right hand of '='. This is very strict code, but it is better to realize every time which side the property is placed.

❑ Visual Basic

```
.Active = True                         ' Open
```

Besides this, this code looks as if put True in variable named Active. 'Open' is an action, not a variable, so it is very hard to catch image for conservative programmers.

Let's go back to 'real' world. Imagine a small cubic box called 'Communication Box'. This box can send out information automatically when it is connecting to PC. We can see a baud rate dial, master/slave switch and power switch. First, set each dials and switches to proper value, then turn on the power. How can we know that this box is running or not? It is very easy question -- just look at the power switch.

If we take this design concept to implement real communication box to virtual computer software, we have to eliminate invisible factor like 'function'. Swing ActiveX controls is made on this design concept.

# 8 Programming Guide for Visual Basic

This chapter, let's make a mini-host to describe programming technique using SwingSecsI Control and SwingSecsII Control. Next, we make very simple HSMS client using SwingHsms Control. We use Visual Basic for the development language.

## 8.1 Specification of mini-host

The specification of mini-host is as follows. Equipment that this mini-host can handle is a ordinary wafer inspection system.

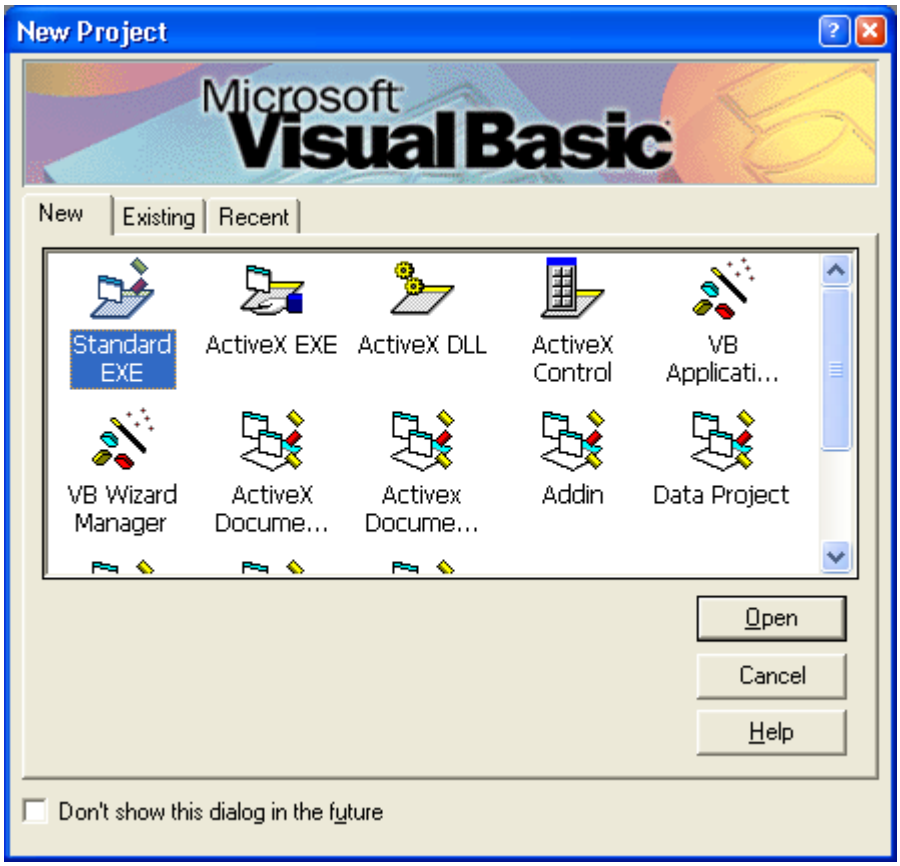
- It is available to handle 'Attempt Online', 'Select Recipe', 'Start Process Program' and 'Measurement Data'.
- Following messages can be sent:
  S1F13W
  S2F41W
  S6F12
- Following messages can be received:
  S1F14
  S2F42
  S6F11W
- It is assumed that equipment has already been initialized and ready to use.
- Do not concern about Stream 9 and Function 0.
- Do not watch T3 time out.

## 8.2 Let's make a mini-host application

### 8.2.1 Establish default program group

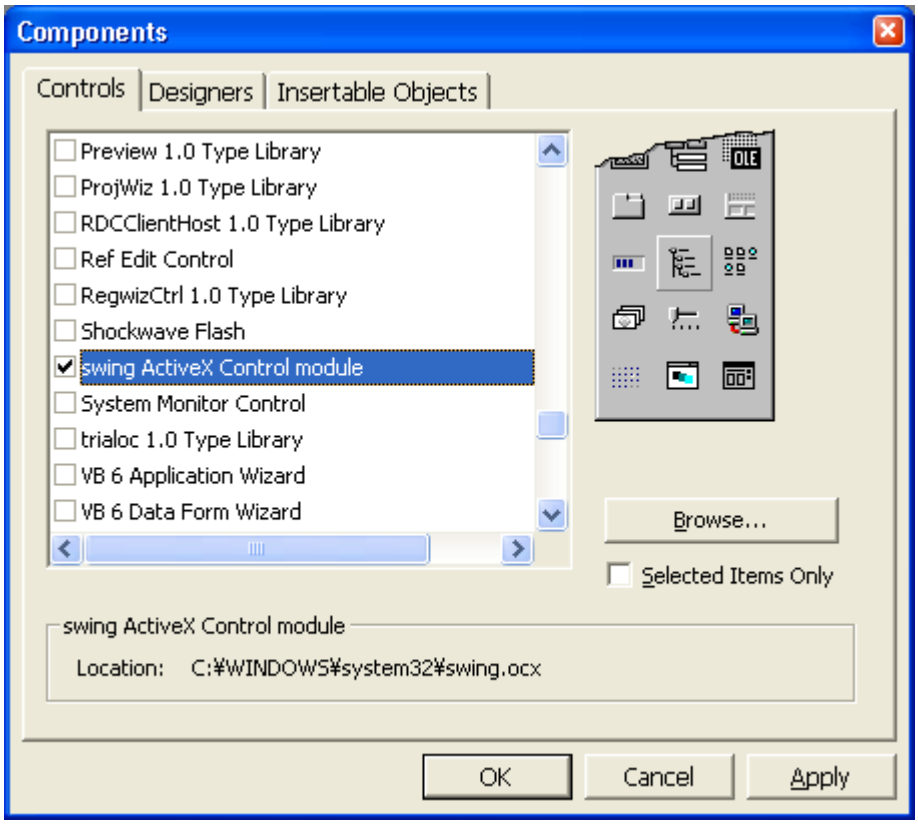
- Run Visual Basic

Run Visual Basic and establish default program group.

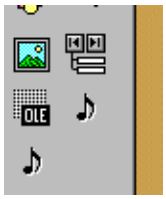



### 8.2.2 Insert Swing ActiveX Control

Select from the menu “Project” – “Components...”, check on “Swing ActiveX Control module”, then press OK button.

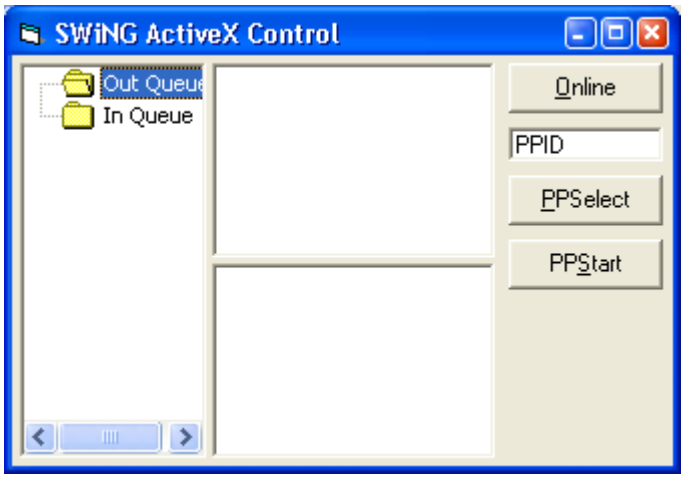



You will find that Swing ActiveX control has appeared in tool box.



### 8.2.3 Paste ActiveX Control

Paste SwingSecsI Control and SwingSecsII Control on the form. I recommend that you paste 2 SwingSecsII Controls for convenience. Because, you can treat incoming message and outgoing message separately.




In addition, buttons and textbox are put properly (see above). Also buttons can be array for better code.

I don't want to enforce you my programming style. These are all up to you and you can arrange by yourselves.

### 8.2.4 Open serial port

When you open serial port, let's write code in *FormLoad* Event. You can be ready to communicate right after the application comes up.

```
.Active = True
If Not .Active Then
  MsgBox "Error : Cannot open serial port!"
End If
```

This is a tiny sample code. I recommend that you put "Open" button and press it to open serial port, if you develop your system application. Sometimes it can not be opened properly, sometimes customer may not want to open from the beginning.

### 8.2.5 Send 'Attempt Online' message

If you send 'Attempt online' command, use Send Method in button pressed event handler function. We are going to use arrayed SwingSecsII Control indexed by 0 for incoming message and 1 for outgoing message. If mini-host send S1F13 to equipment, it must be empty list.

```
.List = "s1f13w{}"
SwingSecsI1.Send .Msg
```

Send Method in SwingSecsI Control requires Msg Property of SwingSecsII Control.

### 8.2.6 Send 'Select Recipe' command

To specify recipe, send PP-SELECT of S2F41.

```
.List = "s2f41w{<a'PP-SELECT'>{{<a'PPID'><a'" + Text1.Text + "'>}}}"
SwingSecsI1.Send .Msg
```

### 8.2.7 Start inspection

To start inspection, send START of S2F41.

```
.List = "s2f41w{<A'START'>{{}}}"
SwingSecsI1.Send .Msg
```

### 8.2.8 Catch event from equipment

When you receive a message, it is reported as Read Event. At the very beginning of this event handler procedure, set received message to SwingSecsII Control. After this procedure, you can browse other properties.

```
.Msg = pszMsg
```

You will receive S6F11 from equipment. We handle only following CEIDs.

```
10   Recipe selected.
11   Recipe selection failure.
20   Measurement completed.
30   Measurement data.
```

CEID is located at node '2'. First, we check this value, then do appropriate procedure.

```
' CEID
.Pointer = "2"
Select Case CInt(.Value)
Case 10
  MsgBox "Recipe selected."
Case 11
  MsgBox "Recipe selection failure"
Case 20
  MsgBox "Measurement completed"
Case 30
  ' Measurement data
End Select
```

S6F11 is expecting for reply, so send binary 0 for secondary message.

```
.List = "s6f12<b 0>"
.Reply pszMsg
SwingSecsI1.Send .Msg
```

Besides this, you will receive S1F14 and S2F42, you can write the same kind of code.

### 8.2.9 Run application

Making mini-host has been completed. This is very simple but still has a minimum functionality as a host system. But entire source code are less than 70 lines. It is very clear that it is more easier to develop software using Swing ActiveX control than without using it, if you write same kind of software from the beginning.

```
Option Explicit

Private Sub Command1_Click(Index As Integer)
  With SwingSecsII1(1)
    Select Case Index
    Case 0
      ' Attempt online
      .List = "s1f13w{}"
    Case 1
      ' Select recipe
      .List = "s2f41w{<a'PP-SELECT'>{{<a'PPID'><a'" + Text1.Text +
"'>}}}"
    Case 2
      ' Start measurement
      .List = "s2f41w{<a'START'>{{}}}"
    End Select

    SwingSecsI1.Send .Msg
  End With
End Sub

Private Sub Form_Load()
  With SwingSecsI1
    .Active = True
    If Not .Active Then
      MsgBox "Error : Cannot open serial port!"
    End If
  End With
End Sub

Private Sub SwingSecsI1_Read(ByVal pszMsg As String)
  With SwingSecsII1(0)
    .Msg = pszMsg
    Select Case .Stream
    Case 1
      Select Case .Fucntion
      Case 14
        ' S1F14
        MsgBox "You are online"
      End Select
    Case 2
      Select Case .Fucntion
      Case 42
        ' S2F42
      End Select
    Case 6
      Select Case .Fucntion
      Case 11
        ' S6F11
        With SwingSecsII1(1)
          .List = "s6f12<b 0>"
          .Reply pszMsg
```

```
        SwingSecsI1.Send .Msg
      End With
      ' CEID
      .Pointer = "2"
      Select Case CInt(.Value)
      Case 10
        MsgBox "Recipe selected"
      Case 11
        MsgBox "Recipe selection failure"
      Case 20
        MsgBox "Measurement completed"
      Case 30
        ' Measurement data
      End Select
    End Select
  End Select
 End With
End Sub
```

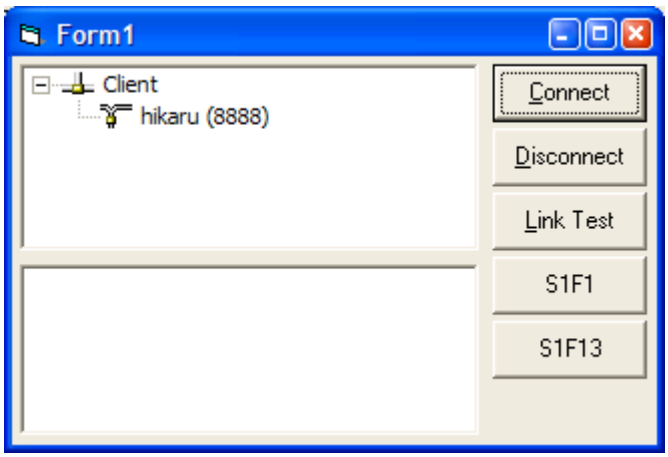
## 8.3 Let's make a simple HSMS client

### 8.3.1 Prepare for application

First, create default program group and insert ActiveX control following same procedure described in previous section.

### 8.3.2 Paste ActiveX Control

Paste SwingHsms Control and SwingSecsII Control on the form. In this case, we use only one SwingSecsII Control to simplify source code for explanation.




### 8.3.3 Connect to server

To establish connection, let's write code in *FormLoad* Event. You can be ready to communicate right after the application comes up.

```
'Connect
.Active = True
```

### 8.3.4 Prepare to send

Message header of HSMS is not the same with the one of SECS-I. So first, we prepare Send function to send HSMS messages.

```
Private Sub Send (bDataMessage As Boolean)
  With SwingSecsII1
    If bDataMessage Then
      .SessionID = 0
      .SType = 0
    Else
      .SessionID = &HFFFF
    End If

    .PType = 0
    .EBit = False
    SwingHsms.Send .Msg
  End With
End Sub
```

### 8.3.5 Send 'Select.req' message

In HSMS-SS (Single Session), client (active entity) should send 'Select Request' message to server (passive entity).

```
  'Connected successfully?
  If .Active Then
    ' Send Select.Req message
    SwingSecsII1.List = "SelectReq"
    Send False
  End If
```

### 8.3.6 Receive HSMS message

It is defined HSMS message type as follows:

| Value | Description |
|---|---|
| 0 | Data message |
| 1 | Select.Req |
| 2 | Select.Rsp |
| 3 | Deselect.Req |
| 4 | Deselect.Rsp |
| 5 | LinkTest.Req |
| 6 | LinkTest.Rsp |
| 7 | Reject.Req |
| 9 | Separate Req |

In Read Event Handler function, incoming message should be checked each time.

```
  With SwingSecsII1
    .Msg = pszMsg

    Select Case .SType
```

```
    Case 0
      'Data message
    Case 1
      'Select.Req
    Case 2
      'Select.Rsp
    Case 3
      'Deselect.Req
    Case 4
      'Deselect.Rsp
    Case 5
      'LinkTest.Req
    Case 6
      'LinkTest.Rsp
    Case 7
      'Reject.Req
    Case 9
      'Separate.Req
    End Select
  End With
```

### 8.3.7 Reply 'LinkTest.rsp' message

When you received 'Link Test Request' message, you must reply immediately. To reply this message, send as follows:

```
  .List = "LinkTestRsp"
  .Reply pszMsg
  .SType = 6
  .Stream = 0
  .Fucntion = 0
  Send False
```

### 8.3.8 Run application

Making simple HSMS client has been completed. You can modify and add functionality to it to be a complicated system. But entire source code are still less than 100 lines. It is very clear that it is more easier to develop software using Swing ActiveX control than without using it, if you write same kind of software from the beginning.

```
Option Explicit

Private Sub Send (bDataMessage As Boolean)
  With SwingSecsII1
    If bDataMessage Then
      .SessionID = 0
      .SType = 0
    Else
      .SessionID = &HFFFF
    End If
    .PType = 0
    .EBit = False
    SwingHsms1.Send .Msg
  End With
End Sub
```

```
Private Sub Command1_Click(Index As Integer)
  With SwingHsms1
    Select Case Index
    Case 0
      'Connect
      .Active = True
      If .Active Then
        SwingSecsII1.List = "SelectReq"
        Send False
      End If
    Case 1
      'Disconnect
      .Active = False
    Case 2
      'Link test
      SwingSecsII1.List = "LinkTestReq"
      Send False
    Case 3
      'S1F1
      SwingSecsII1.List = "s1f1w"
      Send True
    Case 4
      'S1F13
      SwingSecsII1.List = "s1f13w{}"
      Send True
    End Select
  End With
End Sub

Private Sub Form_Load()
  Command1_Click 0
End Sub

Private Sub SwingHsms1_Read (ByVal pszIPAddress As String, ByVal
lPortNumber As Long, ByVal pszMsg As String)
  With SwingSecsII1
    .Msg = pszMsg
    Select Case .SType
    Case 0
      'Data message
      If (.Fucntion Mod 2) = 1 And .WBit Then
        'Send default reply message
        .List = "<b 0>"
        .Reply pszMsg
        Send True
      End If
    Case 1
      'Select.Req
      .List = "SelectRsp"
      .Reply pszMsg
      .SType = 2
      .Stream = 0
      .Fucntion = 0
      Send False
    Case 2
      'Select.Rsp
    Case 3
      'Deselect.Req
      .List = "DeselectRsp"
      .Reply pszMsg
      .SType = 4
      .Stream = 0
      .Fucntion = 0
      Send False
```

```
    Case 4
      'Deselect.Rsp
    Case 5
      'LinkTest.Req
      .List = "LinkTestRsp"
      .Reply pszMsg
      .SType = 6
      .Stream = 0
      .Fucntion = 0
      Send False
    Case 6
      'LinkTest.Rsp
    Case 7
      'Reject.Req
    Case 9
      'Separate.Req
    End Select
  End With
End Sub
```

# 9 SwingSecsI

SwingSecsI Control is an ActiveX control which is implemented SECS-I protocol. You can send or receive SECS-II messages very easily using this component. Normal case, it is used cooperate with SwingSecsII Control.

## 9.1 Reference

First of all, I will explain some special words.

- Persistent Property

If you specify some value for this type of property at design time, ActiveX control saves its property into resource (persistency). So if you don't have to set each property at run time. For example, set Show Property to False at design time, if you want to hide ActiveX control.

### 9.1.1 Active

If this property is set to True, ActiveX control will open serial port using the condition specified by CommPort Property and BaudRate Property. To know port has been opened successfully or not, check Active Property is True or False. If this property is set to False, then port will be closed. If you forget to close serial port when you terminate application, don't worry, ActiveX control automatically close serial port.

- Visual Basic

```
.CommPort = 0                          ' Com1
.BaudRate = 9600                       ' 9600 bps
.Active = True                         ' Open
If Not .Active then
  MsgBox "Error : Cannot open serial port!"
End If
```

- Visual C++

```
m_secs.SetCommPort(0);                 // Com1
m_secs.SetBaudRate(9600);              // 9600 bps
m_secs.SetActive(true);                // Open
if(m_secs.GetActive())
  MessageBox("Error : Cannot open serial port!");
```

### 9.1.2 Appearance

Specify the appearance of window. If this property is 1, the window is shown as sunken style. This property is a persistent property.

- Visual Basic

```
.Appearance = 0                        ' flat
.Appearance = 1                        ' sunken
```

❑ Visual C++

```
m_secs.SetAppearance(0);                // flat
m_secs.SetAppearance(1);                // sunken
```

### 9.1.3 BaudRate

Specify baud rate of serial port. Set proper baud rate number directly. This property is a persistent property.

❑ Visual Basic

```
.BaudRate = 9600                        ‘ 9600bps
.BaudRate = 4800                        ‘ 4800bps
.BaudRate = 2400                        ‘ 2400bps
```

❑ Visual C++

```
m_secs.SetBaudRate(9600);               // 9600bps
m_secs.SetBaudRate(4800);               // 4800bps
m_secs.SetBaudRate(2400);               // 2400bps
```

### 9.1.4 BorderStyle

Specify the border style of window. If this property is 1, the window has border line. This property is a persistent property.

❑ Visual Basic

```
.BorderStyle = 0                        ‘ no border
.BorderStyle = 1                        ‘ border
```

❑ Visual C++

```
m_secs.SetBorderStyle(0);               // no border
m_secs.SetBorderStyle(1);               // border
```

### 9.1.5 CommPort

Specify the serial port number. This value begins from 0. For example, Com #1 is 0, Com #2 is 1. This property is a persistent property.

❑ Visual Basic

```
.CommPort = 0                           ‘ Com1
.CommPort = 1                           ‘ Com2
```

❑ Visual C++

```
m_secs.SetCommPort(0);                  // Com1
m_secs.SetCommPort(1);                  // Com2
```

### 9.1.6 DeviceID

Specify the Device ID in SECS header. This property is a persistent property.

❑ Visual Basic

```
.LoadIni
SwingSecsII1.DeviceID = .DeviceID
```

❑ Visual C++

```
m_secs.LoadIni();
m_msg.SetDeviceID(m_secs.GetDeviceID ());
```

Device ID is one of the part in the message, so it must belong to SECS II. That is why SwingSecsII Control also has DeviceID Property.

☞ One Point
What happens if these 2 properties are not the same? SwingSecsI Control automatically change Device ID in sending message whenever it is not correct. DeviceID Property in the SwingSecsII Control is ignored.

### 9.1.7 IniFile

All the properties are saved in this .ini file. If you specify full path name, ActiveX control saves .ini file in specified folder. However, if you specify only file name without folder name, ActiveX control saves .ini file in Windows system directory. This property is a persistent property.

❑ Visual Basic

```
.IniFile = "tty.ini"                    ' C:\Windows\tty.ini
.IniFile = "C:\a.x"                     ' C:\a.x
```

❑ Visual C++

```
m_secs.SetIniFile("tty.ini");           // C:\Windows\tty.ini
m_secs.SetIniFile("C:\a.x");            // C:\a.x
```

The extension of file name should not always be ini, but traditionally it is better to use ini.

### 9.1.8 IniSection

All the properties are saved in this section in .ini file. If one project contains 2 or more SwingSecsI Control, it can be used separate section name to save in a single .ini file. This property is a persistent property.

❑ Visual Basic

```
.IniSection = "Host"                    ' [Host] section
```

❑ Visual C++

```
m_secs.SetIniSection("Host");           // [Host] section
```

### 9.1.9 Log

Enable or disable log file function. If this property is True, then ActiveX control writes to log file specified by LogFile Property. If this property is False, ActiveX control doesn't write to log file. However, this log file contains very primitive SECS-I protocol only. If you need to write message body as well, please use SwingSecsII Control. This property is a persistent property.

❑ Visual Basic

```
.Log = True                             ' write to log file
.Log = False                            ' do not write to log file
```

❑ Visual C++

```
m_secs.SetLog(true);                    // write to log file
m_secs.SetLog(false);                   // do not write to log file
```

### 9.1.10 LogFile

If Log Property is True, then ActiveX writes to log file specified by LogFile Property. This property is a persistent property.

❑ Visual Basic

```
.LogFile = "C:\Log\Secs.log"
```

❑ Visual C++

```
m_secs.SetLogFile("C:\Log\Secs.log");
```

If you don't use full path name, then ActiveX control writes into its current folder (not a application's current folder). In default installation, it is Windows system folder.

### 9.1.11 Master

If this property is True, then ActiveX control acts as master. If this property is False, it means slave. This property is a persistent property.

❑ Visual Basic

```
.Master = True                          ‘ master
.Master = False                         ‘ slave
```

❑ Visual C++

```
m_secs.SetMaster(true);                 // master
m_secs.SetMaster(false);                // slave
```

### 9.1.12 MSEC

SwingSecsI Control also supports MSEC protocol which is used in Mitsubishi only. If this property is True, it means MSEC, and if False, then SECS. This property is a persistent property.

❑ Visual Basic

```
.MSEC = True                            ‘ MSEC
.MSEC = False                           ‘ SECS
```

❑ Visual C++

```
m_secs.SetMSEC(true);                   // MSEC
m_secs.SetMSEC(false);                  // SECS
```

### 9.1.13 Retry

Specify how many times ActiveX control retry at send failure. Default value is 3 times. This property is a persistent property.

❑ Visual Basic

```
.Retry = 3                              ‘ retry for 3 times
```

❑ Visual C++

```
m_secs.SetRetry(3);                     // retry for 3 times
```

### 9.1.14 Show

If this property is True, then ActiveX control displays queue status. If False, it displays LED like picture. This property is a persistent property.

❑ Visual Basic

```
  .Show = True                              ‘ show queue
  .Show = False                             ‘ show LED like picture
```

❑ Visual C++

```
  m_secs.SetShow(true);                     // show queue
  m_secs.SetShow(false);                    // show LED like picture
```

### 9.1.15 T1

T1 timeout in milliseconds. Default value is 0.5 milliseconds. This property is a persistent property.

❑ Visual Basic

```
  .T1 = 500                                 ‘ 500 milliseconds
```

❑ Visual C++

```
  m_secs.SetT1(500);                        // 500 milliseconds
```

### 9.1.16 T2

T2 timeout in milliseconds. Default value is 10 seconds. This property is a persistent property.

❑ Visual Basic

```
  .T2 = 10000                               ‘ 10 seconds
```

❑ Visual C++

```
  m_secs.SetT2(10000);                      // 10 seconds
```

The value of T2 and T4 timeouts should be set under following guide line.

```
  T4 > T2 * ( Retry + 1 )
```

### 9.1.17 T3

T3 timeout in milliseconds. Default value is 45 seconds. This property is a persistent property.

❑ Visual Basic

```
  .T3 = 45000                               ‘ 45 seconds
```

❑ Visual C++

```
m_secs.SetT3(45000);                    // 45 seconds
```

If T3 timeout occurred, ActiveX control doesn't send S9F9 automatically. So user should implement proper code in application.

### 9.1.18 T4

T4 timeout in milliseconds. Default value is 45 milliseconds. This property is a persistent property.

❑ Visual Basic

```
.T4 = 45000                             ' 45 seconds
```

❑ Visual C++

```
m_secs.SetT4(45000);                    // 45 seconds
```

If this timeout occur, ActiveX control discard message block in queue (garbage collection). The value of T2 and T4 timeouts should be set under following guide line.

```
T4 > T2 * ( Retry + 1 )
```

### 9.1.19 Config

Show modal dialog box to edit properties.

❑ Visual Basic

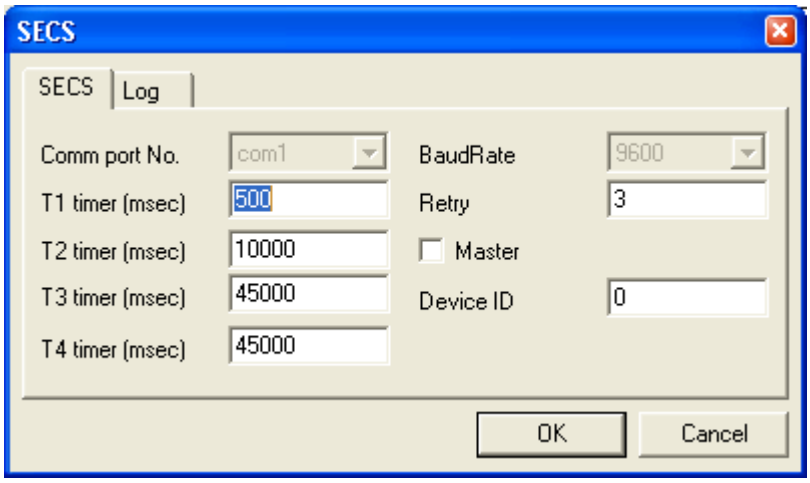
```
Function Config(pszTitle As String) As Boolean
```

❑ Visual C++

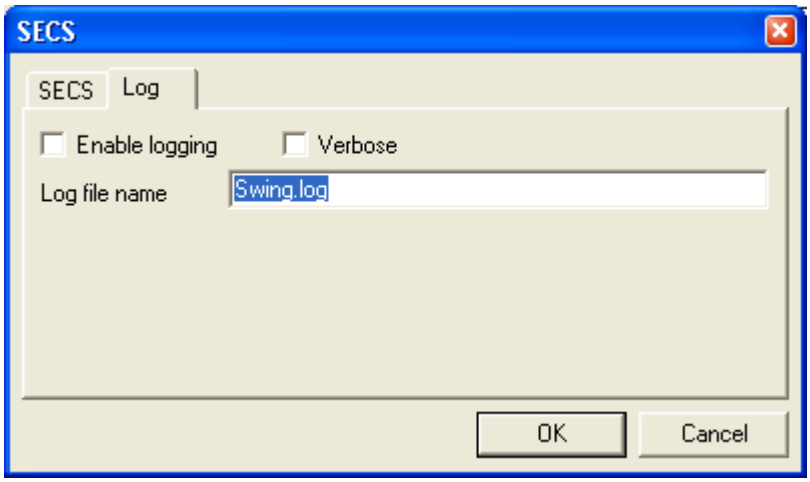
```
BOOL Config(LPCTSTR pszTitle)

pszTitle        Caption title of dialog box.
```

If user clicks OK button to close dialog box, then ActiveX control saves all the properties to ini file specified by IniFile Property and IniSection Property. If saving procedure is successfully completed, then ActiveX returns true. To load and set properties from .ini file, call LoadIni Method.




If serial port has already been opening (Active Property is True), it is impossible to edit CommPort Property and BaudRate Property (see above).




❑ Visual Basic

```
.Config "SECS Configuration"
```

❑ Visual C++

```
m_secs.Config("SECS Configuration");
```

If argument is NULL, IniSection Property is used.

### 9.1.20 LoadIni

Load properties setting from ini file specified by IniFile Property and IniSection Property. If it has failed to load from ini file, ActiveX control sets default values specified in original resource setting at the design time.
This method retrieves all the properties saved by Config Method, so it is recommended to be called at very beginning of application.

❑ Visual Basic

```
Private Sub Form_Load()
  SwingSecsI1.LoadIni
End Sub
```

❑ Visual C++

```
void CxxxView::OnInitialUpdate()
{
  ...
  m_secs.LoadIni();
  ...
}
```

### 9.1.21 Send

Send SECS-II message.

❑ Visual Basic

```
Sub Send(pszMsg As String)
```

❑ Visual C++

```
void Send(LPCTSTR pszMsg)
```

*pszMsg* Message to send.

At this time, message is posted at the end of send queue. If send procedure has been finished successfully, then Written Event occurs.

❑ Visual Basic

```
  .Send SwingSecsII1.Msg
```

❑ Visual C++

```
  m_secs.Send m_msg.GetMsg();
```

### 9.1.22 Errors

This event kicked when communication failure occurred.

❑ Visual Basic

```
Sub Errors(sError As Integer, pszInfo As String)
```

❑ Visual C++

```
void OnErrors(short sError, LPCTSTR pszInfo);


sError          Error code.  It should be one of the following reasons.

                SecsErrorUnexpectedChar          Received unexpected
                                                 character. (=1)
                SecsErrorBadLength               Length byte is not
                                                 correct. (=2)
                SecsErrorSum                     Check sum error. (=3)
                SecsErrorParity                  Parity error. (=4)
                SecsErrorBadMultiBlockNumber     Block number in multi-
                                                 block message is not
                                                 correct. (=5)
                SecsErrorBlockNumber             Block number is
                                                 neither 0 nor 1. (=6)
                SecsErrorNak                     Received NAK. (=7)
                SecsErrorRetry                   Retry over. (=8)
                SecsErrorT4                      T4 time out. (=9)
                SecsErrorT2                      T2 time out. (=10)
                SecsErrorT1                      T1 time out. (=11)
                SecsErrorT3                      T3 time out. (=12)
                SecsErrorDeviceID                Device ID is not
                                                 matched. (=13)

pszInfo         Additional information.  Currently, this value is not
                available.
```

### 9.1.23 Read

This event is invoked when ActiveX control receives message successfully. In case of multi-block message, it is kicked after final block. Received messages will be automatically deleted after T4 timeout has been occurred (garbage collection).

❑ Visual Basic

```
Sub Read(pszMsg As String)
```

❑ Visual C++

```
void OnRead(LPCTSTR pszMsg);


pszMsg          Received message.  Header of multi-block are the same,
                so they are combined into single message.
```

### 9.1.24 SelMsg

This event will be kicked when user clicks and selects list item in ActiveX control.

❑ Visual Basic

```
Sub SelMsg(pszMsg As String)
```

❑ Visual C++

```
void OnSelMsg(LPCTSTR pszMsg);


pszMsg          Selected message.
```

### 9.1.25 Written

This event is kicked when message has been sent successfully or has been failed to send. It is kicked after final block in case of multi-block message. All the messages are deleted when this event has been invoked, whatever the result of send process is succeeded or failed (garbage collection). Send order is from older one to newer one (FIFO=First In, First Out), so event order is the same with calling order of Send Method.

❑ Visual Basic

```
Sub Written(pszMsg As String, bOK As Boolean)
```

❑ Visual C++

```
void OnWritten(LPCTSTR pszMsg, BOOL bOK);


pszMsg          Hexadecimal formatted ASCII string message.
bOK             If send process is succeeded, ActiveX returns True(=1).
                If failure, then False(=0) will be set.  In case of
                False, it means ActiveX has already tried to send for
                retry count times.
```

# 10 SwingSecsII

SwingSecsII Control is the ActiveX control to implement SECS-II structure. Message is analyzed and displayed as tree node. It is very useful to cooperate with SwingSecsI Control or SwingHsms Control.

## 10.1 Reference

First of all, I will explain some special words.

- Persistent Property

If you specify some value for this type of property at design time, ActiveX control saves its property into resource (persistency). So if you don't have to set each property at run time. For example, set Show Property to False at design time, if you want to hide ActiveX control.

- Node

Node consists of '/' (slash mark) and node number. If node is '' (empty), it means root node. Generally speaking, root node is a list type, however, all of the types are available. If root node contains child node, then it must be a list type.

```
{
  <a'Kelly'>
  {
    <a'Brenda'>
    {
      <a'Donna'>
    }
  }
  <a'Valerie'>
  {
    {
      {
        <a'Andrea'>
      }
    }
    <a'Beverly Hills 90210'>
  }
}
```

Followings are example to specify Kelly, Brenda, Donna, Valerie and Andrea as node string.

```
Kelly             1
Brenda            2/1
Donna             2/2/1
Valerie           3
Andrea            4/1/1/1
```

There are no limitations for nesting level of nodes.

### 10.1.1 Appearance

Specify the appearance of window. If this property is 1, the window is shown as sunken style. This property is a persistent property.

❑ Visual Basic

```
.Appearance = 0                          ‘ flat
.Appearance = 1                          ‘ sunken
```

❑ Visual C++

```
m_msg.SetAppearance(0);                  // flat
m_msg.SetAppearance(1);                  // sunken
```

### 10.1.2 Array

This is the number of items which specified node contains. If the node is not array, it must be 1. If it is a list item, it is the number of child nodes.

❑ Visual Basic

```
.Pointer = “”
.Add SecsTypeList, “”
.Pointer = “99”
.Add SecsTypeList, “”
Text1.Text = “Count = “ + Format$(.Array)
```

❑ Visual C++

```
m_msg.SetPointer(“”);
m_msg.Add(SecsTypeList,“”);
m_msg.SetPointer(“99”);
m_msg.Add(SecsTypeList,“”);
m_text1.Format(“Count = %d”,m_msg.GetArray());
```

This is a read-only property. You can not set a value to it.

### 10.1.3 BlockNumber

Block number for SECS-I. If you received multi-block message, this number is the block number of final block. So you can realize in how many blocks the message was sent. It is very important, when you send SECS-I message, you always have to set this property to 1.

❑ Visual Basic

```
.BlockNumber = 1                         ‘ Block number = 1
```

❑ Visual C++

```
m_msg.SetBlockNumber(1);                 // Block number = 1
```

### 10.1.4 BorderStyle

Specify the border style of window. If this property is 1, the window has border line. This property is a persistent property.

❑ Visual Basic

```
.BorderStyle = 0                    ‘ no border
.BorderStyle = 1                    ‘ border
```

❑ Visual C++

```
m_msg.SetBorderStyle(0);            // no border
m_msg.SetBorderStyle(1);            // border
```

### 10.1.5 DeviceID

Specify the Device ID in SECS header. If you call Init Method, then the message is initialized using this Device ID.

❑ Visual Basic

```
.DeviceID = 0                       ‘ Device ID is zero
```

❑ Visual C++

```
m_msg.SetDeviceID(0);               // Device ID is zero
```

This property is a persistent property.。

### 10.1.6 EBit

End bit for SECS-I. Final block is set to True. SwingSecsI Control generate event after it has received final block, so this property value is always True.

❑ Visual Basic

```
If .EBit = False Then
  ‘ Never comes here
End If
```

❑ Visual C++

```
if(!m_msg.GetEBit())
{
  // Never comes here
}
```

Besides this, HSMS doesn't use EBit Property. This area is used for P Type. So EBit Property is not always True for SwingHsms Control.

### 10.1.7 Fucntion

Function number.

❑ Visual Basic

```
If .Stream = 2 And .Fucntion = 42 Then
  ‘ s2f42
  ...
```

❑ Visual C++

```
if(m_msg.GetStream()==2 && m_msg.GetFucntion()==42)
{
  // s2f42
  ...
}
```

### 10.1.8 List

If you read this property, you can obtain message structure as tree format which control is handling. You can set tree format to this property. In case of setting, you can use line-feeds, spaces and tabs in grammar freely.

If you use List Property, it is more simple to use this property than to use Add Method. It is very clear to compare with the code using Add Method Following 2 examples generate exactly the same message in different way.

❑ Using Add Method

❑ Visual Basic

```
.Init
.Stream = 1
.Fucntion = 13
.WBit = True
.Pointer = “”
.Add SecsTypeList, ””
.Pointer = “2”
.Add SecsTypeAscii, “Swing”
.Add SecsTypeAscii, “Ver 3.17”
```

❑ Visual C++

```
m_msg.Init();
m_msg.SetStream(1);
m_msg.SetFucntion(13);
m_msg.SetWBit(true);
m_msg.SetPointer(“”);
m_msg.Add(SecsTypeList,””);
m_msg.SetPointer(“2”);
m_msg.Add(SecsTypeAscii,“Swing”);
m_msg.Add(SecsTypeAscii,“Ver 3.17”);
```

❑ Using List Property

❑ Visual Basic

```
.List = "s1f13w{<A'Swing'><A'Ver 3.17'>}"
```

❑ Visual C++

```
m_msg.SetList("s1f13w{<A'Swing'><A'Ver 3.17'>}");
```

The grammar of the string to set List Property is as follows;

❑ Common Notice

White space (space, tab, carriage return and line feed) is treated as a separator only. So, you can use them to improve readability of source code. But you must be careful that you use in comment or string text, it is treated as character.

From aster (*) to the end of line are treated as comment, except aster in string text.

Integer consists of number '0' to '9' and minus (-) flag. If you want to write in hexadecimal expression put '0x' in front of the value. In this case, you can also use 'a' to 'f' and 'A' to 'F'. For decimal part of the number, you can omit front '0' like '0.9' as '.9'. You can also use exponential expression. There are reserved words like 'true' (=1) and 'false' (=0).

String is surrounded by single quotation mark ('). It is not be able to contain new-line and single quotation itself. So if you do want to fill these kind of characters in string, use hexadecimal expression like 0x0a.

Bold letter portion in explanation means to describe character itself. These characters may also be uppercase or lowercase letter fundamentally. Refer to each explanation for an italic character. Moreover, the portion surrounded by brackets ([]) can be omitted.

❑ Grammar

[**S***xx***F***yy*[w]] *Body*

| | |
|---|---|
| *xx* | Stream number. Don't fill space between 'S' and 'F'. |
| *yy* | Function number. Don't fill space 'f' and 'w'. |
| w | Wait bit. If you want to specify it, add 'w'. It can be omitted. |
| *Body* | Message body. |

In order to recognize stream, function, and wait-bit as 1 lump, don't put neither space nor new-line code among these. All of stream and function can be omitted and only message body can also be described.

❑ Message body

Message body is hierarchy structure.

#### 10.1.8.1 List

```
{[l [Number]]Body}
<[l [Number]]Body>

Number          Number of list.  This is only for compatibility of
                SECSIM.  Swing ActiveX control ignore this number.
```

```
Body            Message body.  You can put other items here.
```

#### 10.1.8.2 Ascii string

```
<a [Strings]>

Strings         ASCII string of text.  Long string can be divided
                separately.  Moreover, you can write character code as
                well, for example:

                <a ‘ABC’ ‘DEF’ ‘012’ 0x33 ‘4’ 53 54 ‘789’>

                could be written as

                <a ‘ABCDEF0123456789’>

                These are exactly the same.
```

#### 10.1.8.3 2-byte character string

```
<a2 [Strings]>

Strings         2-byte character string for far east complicated
                language.  This version of Swing ActiveX control can
                handle only MBCS (Multi Byte Character Set).
```

#### 10.1.8.4 JIS8 string

```
<j [Strings]>[2]

This is treated as same kind of string with ASCII string.

Strings         JIS-8 string of text for Japanese ‘katakana’.  Long
                string can be divided separately.  Moreover, you can
                write character code as well, for example:

                <a ‘ABC’ ‘DEF’ ‘012’ 0x33 ‘4’ 53 54 ‘789’>

                could be written as

                <a ‘ABCDEF0123456789’>

                These are exactly the same.
```

#### 10.1.8.5 Integer

```
<i1[Numbers]>
<i2[Numbers]>
<i4[Numbers]>
<i8[Numbers]>
<u1[Numbers]>
```

[2] I have never seen this type.

```
<u2[Numbers]>
<u4[Numbers]>
<u8[Numbers]>

Numbers          Integer.  It is one of these.

                 i1              8-bit integer
                 i2              16-bit integer
                 i4              32-bit integer
                 i8              64-bit integer
                 u1              8-bit unsigned integer
                 u2              16-bit unsigned integer
                 u4              32-bit unsigned integer
                 u8              64-bit unsigned integer

                 You can write some numbers.  This means array, for
                 example:

                 <i1 1 0x02 3>

This version of Swing ActiveX control can not handle very big value in
i8 and u8.
```

10.1.8.6 Floating point

```
<f4[FNumbers]>
<f8[FNumbers]>

Fnumbers         Floating point value.  It is one of these.

                 f4              32-bit floating point number
                 f8              64-bit floating point number

                 For example:

                 <f4 0 1.0 3.14>
```

10.1.8.7 Binary

```
<b [Numbers]>

Numbers          Binary value.  For example:

                 <b 0xff 0x3e 255 0>
```

10.1.8.8 Boolean

```
<bool [Numbers]>
<boolean [Numbers]>

Numbers          Boolean value.  For example:

                 <bool true false 1 0>
```

### 10.1.9 Msg

If received message is set to this property in Read Event handler function of SwingSecsI Control, each property will be updated and shown as tree. Contents of this property -- taking out (to 'get') -- string is created from each property.

❑ Visual Basic

```
Private Sub SwingSecsI1_Read(ByVal pszMsg As String)
  With SwingSecsII1
    .Msg = pszMsg
    Select Case .Stream
    Case 1
      Select Case .Fucntion
      Case 1
        's1f1
        ...
```

❑ Visual C++

```
Private Sub m_secs_Read(ByVal pszMsg As String)
  m_msg.SetMsg(pszMsg);
  switch(m_msg.GetStream())
  {
  case 1:
    switch(m_msg.GetFucntion())
    {
    case 1:
      // s1f1
      ...
```

### 10.1.10 Pointer

The node used as candidate for operation. Node consists of '/' (slash), node number, and '[]' (bracket). Node number is the number which begins from 1. It is considered that root was specified when node is empty. Node can be compared to the directory structure of MS-DOS (I think it may be easy to understand that the description of 'node' in explanation is replaced with a 'directory').

To create node item, first specify node then add command to it.

❑ Visual Basic

```
  .Pointer = "1"
  .Add SecsTypeAscii, "abc"
```

❑ Visual C++

```
  m_msg.SetPointer("1");
  m_msg.Add(SecsTypeAscii, "abc");
```

You can put 'slush' in front of the node string. It will be ignored.

❑ Visual Basic

```
  .Pointer = ""
  .Pointer = "/"                          ' This is the same

  .Pointer = "1/2"
  .Pointer = "/1/2"                       ' This is also the same
```

❑ Visual C++

```
  m_msg.SetPointer("");
  m_msg.SetPointer("/");                  // This is the same

  m_msg.SetPointer("1/2");
  m_msg.SetPointer("/1/2");               // This is also the same
```

Node can be made under the other node. In this case, you have to make parent node ahead of child node. The order to make nodes may differ from the order of actual SECS-II buffer.

```
1 :  {
2 :    {
3 :      <A'abc'>
4 :    }
5 :    {
6 :      <A'def'>
7 :    }
8 :  }
```

The order to add nodes are no matter and flexible whether 1→2→3→5→6, 1→2→5→3→6 or 1→2→5→6→3.

❑ 1→2→3→5→6

❑ Visual Basic

```
  .Pointer = ""
  .Add SecsTypeList, ""                   ' 1
  .Pointer = "1"
  .Add SecsTypeList, ""                   ' 2
  .Pointer = "1/1"
  .Add SecsTypeAscii, "abc"               ' 3
  .Pointer = "2"
  .Add SecsTypeList, ""                   ' 5
  .Pointer = "2/1"
  .Add SecsTypeAscii, "def"               ' 6
```

❑ Visual C++

```
  m_msg.SetPointer("");
  m_msg.Add(SecsTypeList,"");             // 1
  m_msg.SetPointer("1");
  m_msg.Add(SecsTypeList,"");             // 2
  m_msg.SetPointer("1/1");
  m_msg.Add(SecsTypeAscii,"abc");         // 3
  m_msg.SetPointer("2");
  m_msg.Add(SecsTypeList,"");             // 5
  m_msg.SetPointer("2/1");
  m_msg.Add(SecsTypeAscii,"def");         // 6
```

❑ 1→2→5→3→6

❑ Visual Basic

```
.Pointer = “”
.Add SecsTypeList, “”                ‘ 1
.Pointer = “1”
.Add SecsTypeList, “”                ‘ 2
.Pointer = “2”
.Add SecsTypeList, “”                ‘ 5
.Pointer = “1/1”
.Add SecsTypeAscii, “abc”            ‘ 3
.Pointer = “2/1”
.Add SecsTypeAscii, “def”            ‘ 6
```

❑ Visual C++

```
m_msg.SetPointer(“”);
m_msg.Add(SecsTypeList,“”);          // 1
m_msg.SetPointer(“1”);
m_msg.Add(SecsTypeList,“”);          // 2
m_msg.SetPointer(“2”);
m_msg.Add(SecsTypeList,“”);          // 5
m_msg.SetPointer(“1/1”);
m_msg.Add(SecsTypeAscii,“abc”);      // 3
m_msg.SetPointer(“2/1”);
m_msg.Add(SecsTypeAscii,“def”);      // 6
```

❑ 1→2→5→6→3

❑ Visual Basic

```
.Pointer = “”
.Add SecsTypeList, “”                ‘ 1
.Pointer = “1”
.Add SecsTypeList, “”                ‘ 2
.Pointer = “2”
.Add SecsTypeList, “”                ‘ 5
.Pointer = “2/1”
.Add SecsTypeAscii, “def”            ‘ 6
.Pointer = “1/1”
.Add SecsTypeAscii, “abc”            ‘ 3
```

❑ Visual C++

```
m_msg.SetPointer(“”);
m_msg.Add(SecsTypeList,“”);          // 1
m_msg.SetPointer(“1”);
m_msg.Add(SecsTypeList,“”);          // 2
m_msg.SetPointer(“2”);
m_msg.Add(SecsTypeList,“”);          // 5
m_msg.SetPointer(“2/1”);
m_msg.Add(SecsTypeAscii,“def”);      // 6
m_msg.SetPointer(“1/1”);
m_msg.Add(SecsTypeAscii,“abc”);      // 3
```

When you read node value, first specify node then read.

❑ Visual Basic

```
.Pointer = “1”
If .Value = “abc” Then
  ...
End If
```

❑ Visual C++

```
m_msg.SetPointer(“1”);
if(m_msg.GetValue()==“abc”)
{
  ...
}
```

If node is array, they are returned separated with space code.

❑ Visual Basic

```
.Pointer = “1”
.Add SecsTypeBool, “1”
.Add SecsTypeBool, “0”
...
.Pointer = “1”
Dim strBuf As String
strBuf = .Value                          ‘ “1 0” is returned
```

❑ Visual C++

```
m_msg.SetPointer(“1”);
m_msg.Add(SecsTypeBool,“1”);
m_msg.Add(SecsTypeBool,“0”);
...
m_msg.SetPointer(“1”);
CString strBuf=m_msg.GetValue();       // “1 0” is returned
```

If you extract one of the item in array, use bracket. Index of array begins with 0.

❑ Visual Basic

```
.Pointer = “1[1]”
Dim strBuf As String
strBuf = .Value                          ‘ “0” is returned
```

❑ Visual C++

```
m_msg.SetPointer(“1[1]”);
CString strBuf=m_msg.GetValue();       // “0” is returned
```

So if root node is array, it becomes strange format -- only bracket.

❑ Visual Basic

```
.List = “<f4 1.414 3.14 2.236>”
.Pointer = “[1]”
```

```
  Dim strBuf As String
  strBuf = .Value                          ' "3.14" is returned
```

❑ Visual C++

```
  m_msg.SetList("<f4 1.414 3.14 2.236>");
  m_msg.SetPointer("[1]");
  Cstring strBuf=m_msg.GetValue();       // "3.14" is returned
```

Bracket is ignored in ASCII, JIS-8 and List.

### 10.1.11 PType

P type (presentation type). Almost unexceptional, it is used SECS-II message in HSMS, so this value should be 0.

❑ Visual Basic

```
  If .PType <> 0 Then
    MsgBox "Invalid P-type!"
  End If
```

❑ Visual C++

```
  if(m_msg.GetPType()!=0)
    MessageBox("Invalid P-type!");
```

### 10.1.12 RBit

Reverse bit.

❑ Visual Basic

```
  If .RBit Then
    MsgBox "Invalid reverse-bit!"
  End If
```

❑ Visual C++

```
  if(m_msg.GetRBit())
    MessageBox("Invalid reverse-bit!");
```

### 10.1.13 SessionID

Session ID (AKA Device ID).

❑ Visual Basic

```
  If .SessionID <> &HFFFF Then
```

```
    MsgBox “Invalid Session ID!”
  End If
```

❑ Visual C++

```
  if(m_msg.GetSessionID()!=0xffff)
    MessageBox(“Invalid Session ID!”);
```

### 10.1.14 Show

If this property is True, it displays on screen whether port is opened or not. If this property is False, ActiveX control is not drawn only by applying in a background color. This property is a persistent property.

If node addition is performed in large quantities when Show Property is True, it takes very long time to redraw tree. Probably, it will be good to perform node addition after setting Show Property to False. If addition is completed, set Show Property to True.

❑ Visual Basic

```
  .Show = False
  .Pointer = “1”
  .Add SecsTypeList, “”
  Dim nCnt as Integer
  For nCnt = 1 to 100
    .Pointer = “1/” + Format$(nCnt)
    .Add SecsTypeAscii, “Something”
  Next nCnt
  .Show = True
```

❑ Visual C++

```
  m_msg.SetShow(false);
  m_msg.SetPointer(“1”);
  m_msg.Add(SecsTypeList,“”);
  for(int nCnt=0;nCnt<100;nCnt++)
  {
    CString strBuf;
    strBuf.Format(“1/%d”,nCnt+1);
    m_msg.SetPointer(strBuf);
    m_msg.Add(SecsTypeAscii,“Something”);
  }
  m_msg.SetShow(true);
```

### 10.1.15 SourceID

Source ID.

❑ Visual Basic

```
  Send.SourceID = Receive.SourceID
```

❑ Visual C++

```
m_send.SetSourceID(m_receive.GetSourceID());
```

### 10.1.16 Stream

Stream.

❑ Visual Basic

```
Select Case .Stream
Case 6
  Select Case .Fucntion
  Case 11
    ‘s6f11
    ...
```

❑ Visual C++

```
switch(m_msg.GetStream())
{
case 6:
  switch(m_msg.GetFucntion())
  {
  case 11:
    // s6f11
    ...
```

### 10.1.17 SType

S type (session type). Currently, it is defined as follows:

| Value | Description |
|---|---|
| 0 | Data message |
| 1 | Select.Req |
| 2 | Select.Rsp |
| 3 | Deselect.Req |
| 4 | Deselect.Rsp |
| 5 | LinkTest.Req |
| 6 | LinkTest.Rsp |
| 7 | Reject.Req |
| 8 | (Unused) |
| 9 | Separate.Req |
| 10 | (Unused) |
| 11～127 | (Reserved) |
| 128～255 | (Reserved) |

❑ Visual Basic

```
If .SType = 9 Then
  MsgBox “Received Separate.Req!”
End If
```

❑ Visual C++

```
if(m_msg.GetSType()==9)
  MessageBox("Received Separate.Req!");
```

### 10.1.18 SystemBytes

System bytes. It is 4-byte value and consists of source ID and transaction ID. The value of reply message should be exactly the same with primary message.

❑ Visual Basic

```
Send.SystemBytes = Receive.SystemBytes
```

❑ Visual C++

```
m_send.SetSystemBytes(m_receive.GetSystemBytes());
```

### 10.1.19 TransactionID

Transaction ID.

❑ Visual Basic

```
Send.TransactionID = Receive.TransactionID
```

❑ Visual C++

```
m_send.SetTransactionID(m_receive.GetTransactionID());
```

### 10.1.20 Type

Node type. This is one of following value.

```
SecsTypeInvalid              0      (Invalid)
SecsTypeList                 1      List
SecsTypeBinary               2      Binary
SecsTypeBoolean              3      Boolean
SecsTypeAscii                4      ASCII
SecsTypeJis                  5      JIS-8
SecsTypeLong8                6      64-bit integer
SecsTypeChar                 7      8-bit integer
SecsTypeShort                8      16-bit integer
SecsTypeLong                 9      32-bit integer
SecsTypeDouble              10      64-bit floating point
SecsTypeFloat               11      32-bit floating point
SecsTypeDWord8              12      64-bit unsigned integer
SecsTypeByte                13      8-bit unsigned integer
SecsTypeWord                14      16-bit unsigned integer
SecsTypeDWord               15      32-bit unsigned integer
```

This property is read only. You can not set value.

### 10.1.21 Value

Node value in decimal.

❑ Visual Basic

```
If Cint(.Value) = 201 Then
  Text1.Text = “CEID is 201”
End If
```

❑ Visual C++

```
if(::atoi(m_msg.GetValue())==201)
  m_text1=“CEID is 201”;
```

If node is array, the result is separated by space code.

❑ Visual Basic

```
.List = “{<u2 10 20 30 40>}”
Dim strResult As String
.Pointer = “1”
strResult = .Value                    ‘ “10 20 30 40” is returned
```

❑ Visual C++

```
m_msg.SetList(“{<u2 10 20 30 40>}”);
m_msg.SetPointer(“1”);
CString strBuf=m_msg.GetValue();      // “10 20 30 40” is returned
```

If you want to extract one of array, use bracket. Index of array begins with 0.

❑ Visual Basic

```
.List = “{<u2 10 20 30 40>}”
Dim strResult As String
.Pointer = “1[2]”
strResult = .Value                    ‘ “30” is returned
```

❑ Visual C++

```
m_msg.SetList(“{<u2 10 20 30 40>}”);
m_msg.SetPointer(“1[2]”);
CString strBuf=m_msg.GetValue();      // “30” is returned
```

This property is read only. You can not set value.

### 10.1.22 ValueHex

This property indicates node value as hexadecimal.

❑ Visual Basic

```
If .ValueHex = “ff” Then
  Text1.Text = “Value is 0xff”
End If
```

❑ Visual C++

```
if(m_msg.GetValueHex()==“ff”)
  m_text1=“Value is 0xff”;
```

If node is array, the result is separated by space code.

❑ Visual Basic

```
.List = “{<u2 0x10 0x20 0x30 0x40>}”
Dim strResult As String
.Pointer = “1”
strResult = .ValueHex                    ‘ “0010 0020 0030 0040”
```

❑ Visual C++

```
m_msg.SetList(“{<u2 0x10 0x20 0x30 0x40>}”);
m_msg.SetPointer(“1”);
CString strBuf=m_msg.GetValueHex();    // “0010 0020 0030 0040”
```

If you want to extract one of array, use bracket. Index of array begins with 0.

❑ Visual Basic

```
.List = “{<u2 0x10 0x20 0x30 0x40>}”
Dim strResult As String
.Pointer = “1[2]”
strResult = .ValueHex                    ‘“0030”
```

❑ Visual C++

```
m_msg.GetList(“{<u2 0x10 0x20 0x30 0x40>}”);
m_msg.SetPointer(“1[2]”);
CString strBuf=m_msg.GetValueHex();    // “0030”
```

This property is read only. You can not set value.

### 10.1.23 WBit

Wait bit. If reply message is required, this property is True.

❑ Visual Basic

```
If (.Fucntion Mod 2) And .WBit Then
```

```
    'Send default reply message
    ...
  End If
```

❑ Visual C++

```
  if(m_msg.GetFucntion()%2 && m_msg.GetWBit())
  {
    // Send default reply message
    ...
  }
```

### 10.1.24 Add

Add 1 node.

❑ Visual Basic

```
Function Add(nType As enumSecsType, pszValue As String) As Boolean
```

❑ Visual C++

```
BOOL Add(long nType,LPCTSTR pszValue)
```

| | |
|---|---|
| *sType* | Node type (☞Type Property). |
| *pszValue* | Node value specified by string. This argument is ignored in case of list item. |

### 10.1.25 Init

Remove all of nodes in message structure.

❑ Visual Basic

```
  .Init
  .Stream = 1
  .Fucntion = 13
  .WBit = True
```

❑ Visual C++

```
  m_msg.Init();
  m_msg.SetStream(1);
  m_msg.SetFucntion(13);
  m_msg.SetWBit(true);
```

### 10.1.26 Reply

Initialize header for reply message.

❑ Visual Basic

```
Sub Reply(pszMsg As String)
```

❑ Visual C++

```
void Reply(LPCTSTR pszMsg)
```

*pszMsg* Primary message.

❑ Visual Basic

```
  .List = “<b 0>”
  .Reply pszMsg
  SwingSecsI1.Send .Msg
```

❑ Visual C++

```
  m_msg.SetList(“<b 0>”);
  m_msg.Reply(pszMsg);
  m_secs.Send(m_msg.GetMsg());
```

# 11 SwingHsms

SwingHsms Control is the ActiveX control to implement HSMS protocol. It is very easy to develop HSMS communication software to use this ActiveX control. It is very useful to cooperate with SwingHsms Control.

## 11.1 Reference

First of all, I will explain some special words.

- Persistent Property

If you specify some value for this type of property at design time, ActiveX control saves its property into resource (persistency). So if you don't have to set each property at run time. For example, set Show Property to False at design time, if you want to hide ActiveX control.

### 11.1.1 Active

The Action of this property depends upon Server Property.

First, if Server Property is True, ActiveX control opens TCP port specified by LocalPortNumber Property and wait for connection from client. At this time, connection is not established -- when client connects server, it will have connected. Active Property becomes True when TCP port has been opened successfully, although connection has not established.

Besides this, if Server Property is False, ActiveX control attempts to connect to server specified by IPAddress Property and PortNumber Property. It is possible to use IP address or computer name (host name) for IPAddress Property. If server is running on the same computer, it is possible to specify IPAddress Property as "" (empty). TCP port number of your computer can be specified by LocalPortNumber Property, however it is possible to let server determine free TCP port number automatically. If TCP port number is fixed number, it may take very long time to establish connection, so auto numbering faculty is very useful. To use auto numbering, set LocalPortNumber Property 0. When connection has been successfully established, Active Property becomes True.

If you set this property False, then connection will be lost. Don't worry to forget to disconnect established connection at termination of application. ActiveX control automatically delete all of the connection. When Active Property on server side is set to False, then all the connections between each client are disconnected.

- Visual Basic

```
.IPAddress = "hsms_server"
.PortNumber = 5000
.LocalPortNumber = 0
.Server = False
.Active = True                          ' Open
If Not .Active then
  MsgBox "Cannot connect to server!"
End If
```

❑ Visual C++

```
m_hsms.SetIPAddress("hsms_server");
m_hsms.SetPortNumber(5000);
m_hsms.SetLocalPortNumber(0);
m_hsms.SetServer(false);
m_hsms.SetActive(true);                // Open
if(!m_hsms.GetActive())
  MessageBox("Cannot connect to server!");
```

If the other side of Active Property is set to False, connection will be disappeared (like spouse relationship). Please notice that Active Property will be set to False automatically at this time whatever it is client.

### 11.1.2 Appearance

Specify the appearance of window. If this property is 1, the window is shown as sunken style. This property is a persistent property.

❑ Visual Basic

```
.Appearance = 0                        ' flat
.Appearance = 1                        ' sunken
```

❑ Visual C++

```
m_hsms.SetAppearance(0);               // flat
m_hsms.SetAppearance(1);               // sunken
```

### 11.1.3 BorderStyle

Specify the border style of window. If this property is 1, the window has border line. This property is a persistent property.

❑ Visual Basic

```
.BorderStyle = 0                       ' no border
.BorderStyle = 1                       ' border
```

❑ Visual C++

```
m_hsms.SetBorderStyle(0);              // no border
m_hsms.SetBorderStyle(1);              // border
```

### 11.1.4 IniFile

All the properties are saved in this .ini file. If you specify full path name, ActiveX control saves .ini file in specified folder. However, if you specify only file name without folder name, ActiveX control saves .ini file in Windows system directory. This property is a persistent property.

❑ Visual Basic

```
.IniFile = “tty.ini”                    ‘ C:\Windows\tty.ini
.IniFile = “C:\a.x”                     ‘ C:\a.x
```

❑ Visual C++

```
m_hsms.SetIniFile(“tty.ini”);           // C:\Windows\tty.ini
m_hsms.SetIniFile(“C:\a.x”);            // C:\a.x
```

The extension of file name should not always be ini, but traditionally it is better to use ini.

### 11.1.5 IniSection

All the properties are saved in this section in .ini file. If one project contains 2 or more SwingHsms Controls, it can be used separate section name to save in a single .ini file. This property is a persistent property.

❑ Visual Basic

```
.IniSection = “Host”                    ‘ [Host] section
```

❑ Visual C++

```
m_hsms.IniSection(“Host”);              // [Host] section
```

### 11.1.6 IPAddress

Specify IP address to connect to. It is possible to set IPAddress Property decimal number of IP address or computer name (host name). If server is running on the same computer, it is possible to specify IPAddress Property as "" (empty).

❑ Visual Basic

```
.IPAddress = “”
.PortNumber = 5000
.LocalPortNumber = 0
.Server = False
.Active = True                          ‘ Open
If Not .Active then
  MsgBox “Cannot connect to server!”
End If
```

❑ Visual C++

```
m_hsms.SetIPAddress(“”);
m_hsms.SetPortNumber(5000);
m_hsms.SetLocalPortNumber(0);
m_hsms.SetServer(false);
m_hsms.SetActive(true);                 // Open
if(!m_hsms.GetActive())
```

```
   MessageBox("Cannot connect to server!");
```

### 11.1.7 LocalPortNumber

Local TCP port number of your computer. Server does not have to specify PortNumber Property but does local port number, however client must specify PortNumber Property. If you want server to determine free TCP port automatically, set LocalPortNumber to 0.

Before you use LocalPortNumber, you must check the number you want to use is not used by the other 'well known port' (like #80 for http server). Generally, it is recommended to use the number over 1024. This property is a persistent property.

❑ Visual Basic

```
  .LocalPortNumber = 0                    ' Auto determine
```

❑ Visual C++

```
  m_hsms.SetLocalPortNumber(0);           // Auto determine
```

### 11.1.8 Log

Sorry that faculty for this property has not been implemented yet.

### 11.1.9 LogFile

Sorry that faculty for this property has not been implemented yet.

### 11.1.10 MaxLength

Maximum message length to be able to handle with this ActiveX control. This property is a persistent property.

❑ Visual Basic

```
  .MaxLength = 10240                      ' 10KB
```

❑ Visual C++

```
  m_hsms.SetMaxLength(10240);             // 10KB
```

### 11.1.11 PortNumber

TCP port number. Server does not have to specify PortNumber Property but does local port number, however client must specify PortNumber Property. If you want server to determine free TCP port automatically, set LocalPortNumber to 0.

Before you use LocalPortNumber, you must check the number you want to use is not used by the other 'well known port' (like #80 for http server). Generally, it is recommended to use the number over 1024. This property is a persistent property.

❑ Visual Basic

```
.PortNumber = 5000
```

❑ Visual C++

```
m_hsms.SetPortNumber(5000);
```

### 11.1.12 Selected

The status whether ActiveX has already received Select.req. If ActiveX control receives Select.req, then SwingHsms Control straight to shift 'selected' status. If you don't want ActiveX to shift 'selected', then set Selected Property to False again. If you have received Select.rsp, you can judge whether system goes to 'Selected' or not using reason code. This property relates to T7 timeout.

❑ Visual Basic

```
SwingHsms1.Selected = False
```

❑ Visual C++

```
m_hsms.SetSelected(false);
```

### 11.1.13 Server

If this property is True, then it acts as passive entity. If it is false, it acts as active entity. This property is a persistent property.

❑ Visual Basic

```
.Server = True                          ' Server
.Server = False                         ' Client
```

❑ Visual C++

```
m_hsms.SetServer(true);                 // Server
m_hsms.SetServer(false);                // Client
```

### 11.1.14 Show

If this property is True, then ActiveX control shows connection status on screen. When server has connection between clients, it shows every connection on screen. It is very natural that client can connect with only one server. If this property is False, it shows Light Emitting Diode (LED) style bitmap. This property is a persistent property.

❑ Visual Basic

```
.Show = True                          ‘ Show connection status
.Show = False                         ‘ Show LED like picture
```

❑ Visual C++

```
m_hsms.SetShow(true);                 // Show connection status
m_hsms.SetShow(false);                // Show LED like picture
```

### 11.1.15 T3

T3 time out in milliseconds. Default value is 45 seconds. This property is a persistent property.

❑ Visual Basic

```
.T3 = 500                             ‘ 500 milliseconds
```

❑ Visual C++

```
m_hsms.SetT3(500);                    // 500 milliseconds
```

### 11.1.16 T5

Sorry that faculty for this property has not been implemented yet.

### 11.1.17 T6

T6 time out in milliseconds. Default value is 5 seconds. This property is a persistent property.

❑ Visual Basic

```
.T6 = 5000                            ‘ 5 seconds
```

❑ Visual C++

```
m_hsms.SetT6(5000);                   // 5 seconds
```

### 11.1.18 T7

T7 time out in milliseconds. Default value is 10 seconds. This property is a persistent property.

❑ Visual Basic

```
  .T7 = 10000                              ‘ 10 seconds
```

❑ Visual C++

```
  m_hsms.SetT7(10000);                     // 10 seconds
```

### 11.1.19 T8

T8 time out in milliseconds. Default value is 0.5 seconds. This property is a persistent property.

❑ Visual Basic

```
  .T8 = 500                                ‘ 500 milliseconds
```

❑ Visual C++

```
  m_hsms.SetT8(500);                       // 500 milliseconds
```

### 11.1.20 Config

Show dialog box to edit properties.

❑ Visual Basic

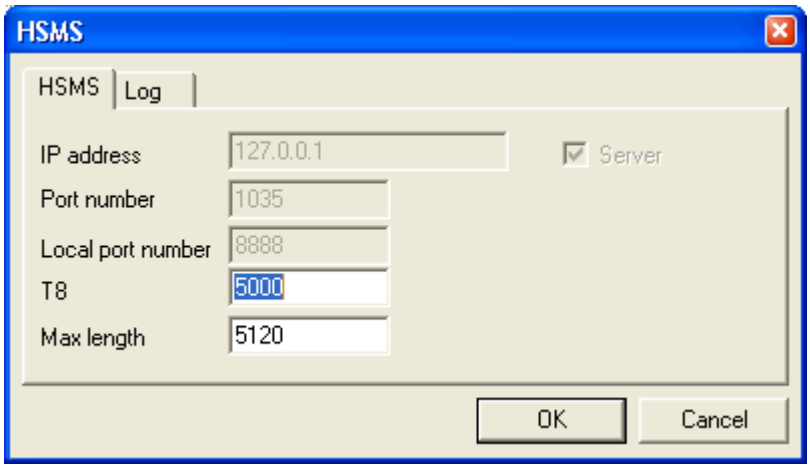
```
Function Config(pszTitle As String) As Boolean
```

❑ Visual C++

```
BOOL Config(LPCTSTR pszTitle)
```

*pszTitle* Caption title of dialog box. If you doesn’t specify, IniSection Property is used for caption title.

If user press OK button to close dialog box, ActiveX control will save properties to .ini file specified by IniFile Property and IniSection Property. After save procedure has been completed successfully, this method returns True. To retrieve saved information from .ini file, call LoadIni Method.




When port has already been opened (when Active Property is True), it is impossible to change IPAddress Property, PortNumber Prperty and LocalPortNumber Property.

❑ Visual Basic

```
.Config "HSMS Configuration"
```

❑ Visual C++

```
m_hsms.Config("HSMS Configuration");
```

### 11.1.21 ConvertIPAddress

Convert computer name (host name) to IP address.

❑ Visual Basic

```
Function ConvertIPAddress(pszName As String) As String
```

❑ Visual C++

```
CString ConvertIPAddress(LPCTSTR pszName)

pszName          Host name to convert.
```

❑ Visual Basic

```
.IPAddress = .ConvertIPAddress("")
```

❑ Visual C++

```
m_hsms.SetIPAddress(m_hsms.ConvertIPAddress(""));
```

### 11.1.22 Disconnect

Disconnect specified connection. This method is used for server side.

❑ Visual Basic

```
Function Disconnect(pszIPAddress As String, lPortNumber As Long) As
Boolean
```

❑ Visual C++

```
BOOL Disconnect(LPCTSTR pszIPAddress,long lPortNumber)
```

```
pszIPAddress     IP address to disconnect.
lPortNumber      TCP port number to disconnect.
```

❑ Visual Basic

```
  .Disconnect "", 5000
```

❑ Visual C++

```
  m_hsms.Disconnect("",5000);
```

### 11.1.23 GetHostName

Get my computer name.

❑ Visual Basic

```
Function GetHostName() As String
```

❑ Visual C++

```
CString GetHostName()
```

❑ Visual Basic

```
  .IPAddress = .GetHostName()
```

❑ Visual C++

```
  m_hsms.SetIPAddress(m_hsms.GetHostName());
```

### 11.1.24 LoadIni

Retrieve properties from .ini file specified by IniFile Property and IniSection Property. If it is impossible to read from .ini file, ActiveX control uses default value specified in resource settings.

It is recommended to call this method at very beginning of application's starting up for retrieving properties saved by Config Method.

❑ Visual Basic

```
Private Sub Form_Load()
  SwingHsms1.LoadIni
End Sub
```

❑ Visual C++

```
void CxxxView::OnInitialUpdate()
{
  ...
  m_hsms.LoadIni();
  ...
}
```

### 11.1.25 Send

Send one message.

❑ Visual Basic

```
Sub Send(pszMsg As String)
```

❑ Visual C++

```
void Send(LPCTSTR pszMsg)
```

*pszMsg* Message to send.

❑ Visual Basic

```
  SwingHsms1.Send SwingSecsII1.Msg
```

❑ Visual C++

```
  m_hsms.Send(m_msg.GetMsg());
```

### 11.1.26 Connected

This event occurs when connection has been established. On server side, connection has not been established whatever Active Property is True. It will have been established connection after client connects to server.

❑ Visual Basic

```
Sub Connected(pszIPAddress As String, lPortNumber As Long, bConnect As
                Boolean)
```

❑ Visual C++

```
void OnConnected(LPCTSTR pszIPAddress, long lPortNumber, BOOL bConnect)
```

| | |
|---|---|
| *pszIPAddress* | IP address of message sender. |
| *lPortNumber* | TCP port number of message sender. |
| *bConnect* | If True, it means established connection. If false, it means disconnected. |

### 11.1.27 Errors

This event occurs at communication error.

❑ Visual Basic

```
Sub Errors(pszIPAddress As String, lPortNumber As Long, sError As
                Integer, pszInfo As String)
```

❑ Visual C++

```
void OnErrors(LPCTSTR pszIPAddress, long lPortNumber, short sError,
                LPCTSTR pszInfo)
```

| | |
|---|---|
| *pszIPAddress* | IP address of message sender. |
| *lPortNumber* | TCP port number of message sender. |
| *sError* | Error code. It is one of the following value: |

❑ Errors from Swing

| | |
|---|---|
| -2 | Received message longer than MaxLength Proerty. |
| -4 | T8 time out. |
| -7 | T6 time out. |
| -8 | T7 time out. |

❑ Errors from WinSock

| | | |
|---|---|---|
| 10093 | WSANOTINITIALISED | Socket has not been initialized. |
| 10050 | WSAENETDOWN | Network subsystem error. |
| 10048 | WSAEADDRINUSE | Socket local address is in use. |
| 10014 | WSAEFAULT | Invalid user address (like invalid character). |
| 10036 | WSAEINPROGRESS | Service provider is in progress. |
| 10049 | WSAEADDRNOTAVAIL | Remote address can not be available. |
| 10047 | WSAEAFNOSUPPORT | Cannot use specified address family for this socket. |
| 10061 | WSAECONNREFUSED | Connection has been refused. |
| 10039 | WSAEDESTADDRREQ | ? |
| 10022 | WSAEINVAL | Invalid listening socket. |
| 10056 | WSAEISCONN | Already connected. |
| 10024 | WSAEMFILE | ? |
| 10051 | WSAENETUNREACH | Cannot reach to network. |
| 10055 | WSAENOBUFS | Buffer size is not enough. |
| 10038 | WSAENOTSOCK | Not a socket. |

```
   10060  WSAETIMEDOUT        Time out before established connection.
   10035  WSAEWOULDBLOCK      Cannot execute right now.

pszInfo          Additional information.  This argument can not be used
                 now.
```

### 11.1.28 Read

This event occurs when ActiveX control has received message successfully.

❑ Visual Basic

```
Sub Read(pszIPAddress As String, lPortNumber As Long, pszMsg As String)
```

❑ Visual C++

```
void OnRead(LPCTSTR pszIPAddress, long lPortNumber, LPCTSTR pszMsg)
```

```
pszIPAddress     IP address of message sender.
lPortNumber      TCP port number of message sender.
pszMsg           Received message.
```

### 11.1.29 SelConnection

This event occurs when user has changed selection of connection with mouse.

❑ Visual Basic

```
Sub SelConnection(pszIPAddress As String, lPortNumber As Long)
```

❑ Visual C++

```
void OnSelConnection(LPCTSTR pszIPAddress, long lPortNumber)
```

```
pszIPAddress     IP address of message sender.
lPortNumber      TCP port number of message sender.
```

# 12 Programming Hints

## 12.1 Don't use magic numbers

It is not a good idea to use 'magic' number directly, you had better to use constant value for readability. ActiveX control already contains following constants.

❑ Visual Basic

For Type Property of SwingSecsII

```
  SecsTypeInvalid                    0
  SecsTypeList                       1
  SecsTypeBinary                     2
  SecsTypeBoolean                    3
  SecsTypeAscii                      4
  SecsTypeJis                        5
  SecsTypeLong8                      6
  SecsTypeChar                       7
  SecsTypeShort                      8
  SecsTypeLong                       9
  SecsTypeDouble                     10
  SecsTypeFloat                      11
  SecsTypeDWord8                     12
  SecsTypeByte                       13
  SecsTypeWord                       14
  SecsTypeDWord                      15
```

For *sError* in Errors Event of SwingSecsI Control

```
  SecsUnexpectedChar                 1
  SecsBadLength                      2
  SecsBadSum                         3
  SecsBadParity                      4
  SecsBadMultiBlockNumber            5
  SecsBadBlockNumber                 6
  SecsNak                            7
  SecsRetry                          8
  SecsT4                             9
  SecsT2                             10
  SecsT1                             11
  SecsT3                             12
  SecsBadDeviceID                    13
```

For Visual C++ 6.0 can not use above constants, it is useful to write constants manually as follows:

❑ Visual C++

```
enum[3]
{
  SecsTypeInvalid=0,                 // No need "=0"
  SecsTypeList,
  SecsTypeBinary,
  SecsTypeBoolean,
```

[3] In Visual C++ .NET, not to name 'enum' doesn't make compilation error, but it becomes 'no name enum'.

```
  SecsTypeAscii,
  SecsTypeJis,
  SecsTypeLong8,
  SecsTypeChar,
  SecsTypeShort,
  SecsTypeLong,
  SecsTypeDouble,
  SecsTypeFloat,
  SecsTypeDWord8,
  SecsTypeByte,
  SecsTypeWord,
  SecsTypeDWord,
};
```

## 12.2 How to send S1F1?

When you make a message, it is more easier to use List Property than to use Add Method. For another one's convenience to read code, it is strongly recommended to use List Property.

❑ Visual Basic

```
  .List = "s1f1w"
```

❑ Visual C++

```
  m_msg.SetList("s1f1w");
```

It can be written separately as follows, however this code does not keep good readability.

❑ Visual Basic

```
  With SwingSecsII1
    .Init
    .Stream = 1
    .Fucntion = 1
    .WBit = True
    SwingSecsI1.Send .Msg
  End With
```

❑ Visual C++

```
  m_msg.Init();
  m_msg.SetStream(1);
  m_msg.SetFucntion(1);
  m_msg.SetWBit(true);
  m_secs.Send(m_msg.GetMsg());
```

## 12.3 When you send S1F13

Use List Property as follows:

❑ Visual Basic

```
.List = _
  “s1f13w” + _
  “{” + _
  “  <a‘Swing’>” + _
  “  <a‘ver 2.71 Copyright(C) 1996-2004 JazzSoft’>” + _
  “}”
```

❑ Visual C++

```
m_msg.SetList(
  “s1f13w”
  “{”
  “  <a‘Swing’>”
  “  <a‘ver 2.71 Copyright(C) 1996-2004 JazzSoft’>”
  “}”
  );
```

Of course, it is possible to write one line, but it is better to fill indents for readability.

## 12.4 Create more than one node

To make following message,

```
{
  <bool true>
  {
    <u2 201>
    <u2 92>
  }
}
```

Use List Property as follows:

❑ Visual Basic

```
.List = _
  “{” + _
  “  <bool true>” + _
  “  {” + _
  “     <u2 201>” + _
  “     <u2 92>” + _
  “  }” + _
  “}”
```

❑ Visual C++

```
m_msg.SetList(
  “{”
  “  <bool true>”
  “  {”
  “     <u2 201>”
  “     <u2 92>”
  “  }”
```

```
  "}"
  );
```

## 12.5 Analyze received message

When you analyze received message, first specify node then read value directly.

❑ Visual Basic

```
.Pointer = "3"
If .Type <> SecsTypeAscii Then
  MsgBox "Invalid type"
  Exit Sub
End If
If CInt(.Value) = 3 Then
  MsgBox "Received value is 3"
End If
```

❑ Visual C++

```
m_msg.SetPointer("3");
if(m_msg.GetType()!=SecsTypeAscii)
{
  MessageBox("Invalid type");
  return;
}
if(::atoi(m_msg.GetValue())==3)
  MessageBox("Received value is 3");
```

The value is returned as string format, so it is possible to read value without type checking. For example, sometimes receiver cannot determine the value in S2F37 whether it is U4 or U2. In this case, receiver ignore type checking and can use the value.

❑ Visual Basic

```
.Pointer = "3"
Dim nType As Integer
nType = .Type
If (nType <> SecsTypeDWord) And (nType <> SecsTypeWord) Then
  MsgBox "Invalid type"
  Exit Sub
End If
If .Value = "3" Then
  MsgBox "Received value is 3"
End If
```

❑ Visual C++

```
m_msg.SetPointer("3");
int nType=m_msg.GetType();
if(nType!=SecsTypeDWord && nType!=SecsTypeWord)
{
  MessageBox("Invalid type");
  return;
}
```

```
if(m_msg.GetValue()==“3”)
  MessageBox(“Received value is 3”);
```

## 12.6 Never show message box in event handler function

It is very dangerous to show modal dialog box (including message box) in event handler function. Let's see following code:

❑ Visual Basic

```
If .Value = “OK” Then
  .List = “<bool true>”
  .Reply pszMsg
  SwingSecsI1.Send .Msg
Else
  MsgBox “It was NG”
  .List = “<bool false>”
  .Reply pszMsg
  SwingSecsI1.Send .Msg
End If
```

If user forget to close message box for a long time, reply message will be delayed to send. So it may cause T3 time out. When T3 time out has occurred, S9F9 will be received and ActiveX invokes another event recursively.

To eliminate this problem, use modeless dialog box or put message line on screen without using dialog box.

❑ Visual Basic

```
  ErrorMsg.Caption = “It was NG”
  ErrorMsg.Blink = True
```

# 13 Index

## V



## W