Swing ActiveX Controls
ユーザーマニュアル

Jazz Soft

1 改訂履歴

版	日付	備考
1.00	May,27th,1999	初版
2.71	May,10th,2001	全面的に改定し HTML から Microsoft™ Word ドキュメントに変換。
3.10	Feb,15th,2002	メディアバージョンに伴う改訂。
3.13	Oct,23rd,2002	処理の高速化。
3.16	Nov,13th,2003	HSMS の機能強化
3.17	Jun,14th,2004	HSMS のサンプルを追加
3.20	Feb,23rd,2005	SwingComm コントロールを追加

2　新しい機能

- 巨大なメッセージの処理に時間がかかる場合があったのを、大幅に高速化しました。
- メディアバージョンが Windows XP に対応しました。
- SECS-II の 2 バイト文字列に対応
- T7 タイムアウトを追加。

3　はじめに

Swing ActiveX コントロールのメディアバージョンは従来より要望の多かった、ウィンドゥズの再インストール時の問題をハードウェアキーによるプロテクトで解決したものです。

2002 年 2 月 15 日
開発者

4 目次

5　使用環境

HASP のタイプにより異なります。

USB 版

- Windows 98(USB 対応版), Windows Me または Windows 2000 または Windows XP。
- Visual Basic や Visual C++などの Active X 対応の開発言語（32 ビット版）。

プリンタポート版

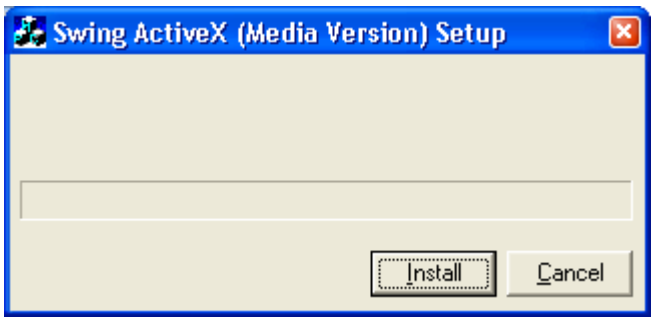
- Windows 95, Windows 98, Windows Me または WindowsNT 4.00, Windows 2000 または Windows XP。
- Visual Basic や Visual C++などの Active X 対応の開発言語（32 ビット版）。

6　インストール

SwingII のインストールファイルは zip 形式で圧縮されているため、まず解凍が必要です。Zip の解凍ツールは無償で利用できるものもインターネット上にありますので、無ければ入手してください。

6.1　Swing ActiveX コントロールのインストール

ハードディスクに適当なフォルダを作って解凍したら、いよいよインストールの開始です。インストールは専用のセットアッププログラムで行います。Setup.exe を起動してください。



『Install』ボタンをクリックするとインストールが始まります。従来バージョンのようにパスワードを入力する必要はありません。

試用版は画面に頻繁にダイアログボックスが表示される以外は製品版とまったく同じです。機能制限などは特にありません。

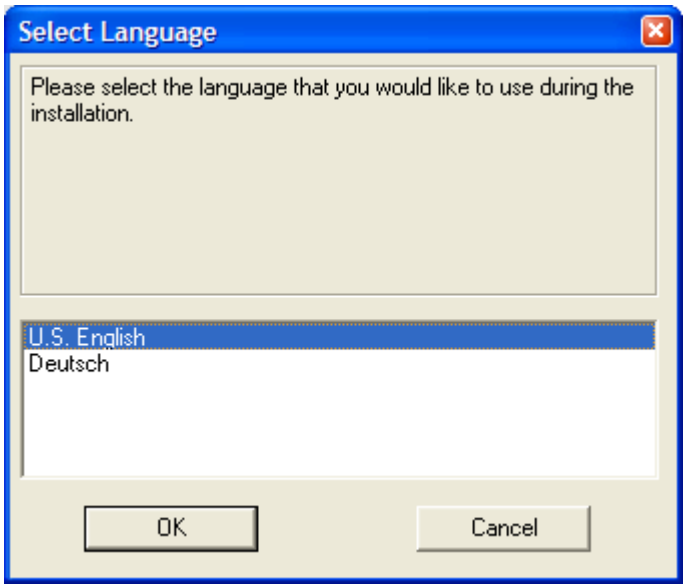
6.2　HASP ドライバのインストール

新規に Swing メディアバージョンをインストールした場合は、HASP ドライバもインストールする必要があります。

(1) 実行中のアプリケーションがある場合は、全て終了させます。

(2) **USB タイプの HASP を使う場合はキーを挿さないでください。プリンタポートタイプの HASP を使う場合はここでキーを挿してください。**この部分は非常に重要なので間違いのないよう行ってください。[1]

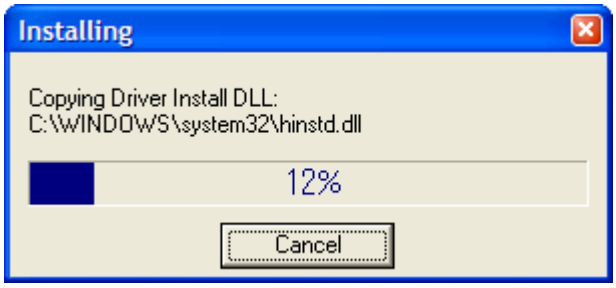
(3) hdd32.exe を実行します。以下のような確認画面が表示されます。



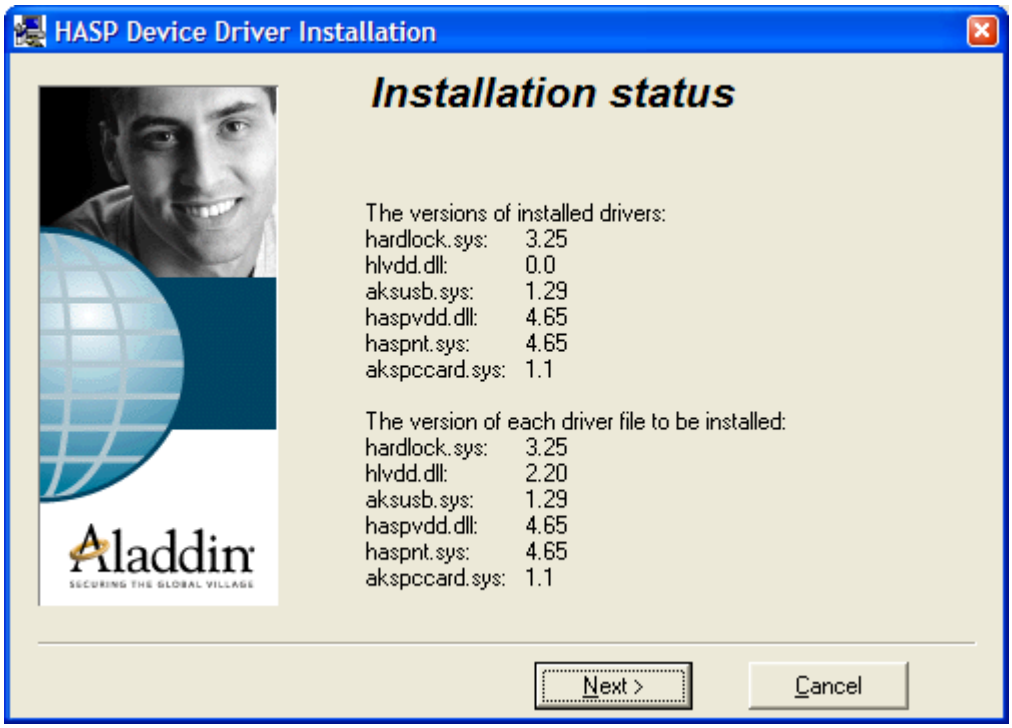
(4) U.S. English（英語）を選択して OK ボタンを押します。

[1]間違えてしまった場合はもう一度 HASP ドライバのインストールを行ってください。

(5) Next ボタンを押します。ドライバファイルのコピーが始まります。

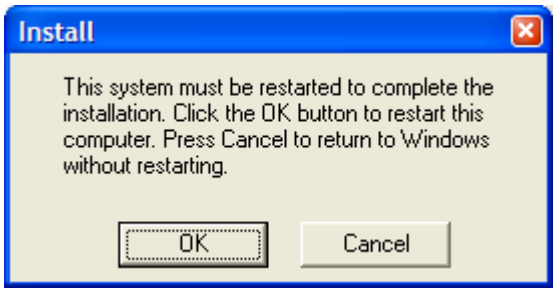


(6) コピーが完了すると以下のような画面が表示されます。



(7) Next ボタンを押すと HASP のインストールが始まります。この処理には数分かかる場合があります。終了すると以下のような画面が表示されます。

(8) 再起動が必要な場合は以下のような画面が表示されます。


Install
This system must be restarted to complete the installation. Click the OK button to restart this computer. Press Cancel to return to Windows without restarting.
OK
Cancel

7　設計思想

SwingII を使ってアプリケーションを作成する場合にもっとも相性のいい言語は何といっても Visual Basic です。プロパティに対するアクセスが非常に直感的で分かりやすくなっているのでバグも少なくなります。同じ処理を Visual Basic と Visual C++で記述してみると一目瞭然です。この例では通信ポート#1 を 9600bps でオープンし、失敗したらメッセージを表示してみます。

❑ Visual Basic の場合

```
With SwingSecsI1
  .CommPort = 0
  .BaudRate = 9600
  .Active = True
  If Not .Active Then
    MsgBox “エラー：通信ポートをオープンできません！”
  End If
End With
```

❑ Visual C++の場合

```
m_secs.SetCommPort(0);
m_secs.SetBaudRate(9600);
m_secs.SetActive(true);
if(!m_secs.GetActive())
  MessageBox(“エラー：通信ポートをオープンできません！”);
```

行数では Visual C++の方が短いため一見シンプルに思えるかも知れません。しかしよく見てみると Visual Basic は対象オブジェクトを With 文で省略することができるため、コードはかなり見やすくなっています。

ここで注目すべき事は Active プロパティに対する操作です。このプロパティに関しては値をセットするか読み出すかでは本質的に意味が違います。値をセットする場合はオープン／クローズする操作を意味し、読み出す場合は現在オープンされているかを調べる目的で使います。Visual C++はこの意味では非常に忠実に記述しなければなりません。

❑ Visual C++の場合

```
m_secs.SetActive(true);                ‘オープンする
if(!m_secs.GetActive())                ‘オープンできたか？
```

一方 Visual Basic は一見しただけではこの点があいまいです。

❑ Visual Basic の場合

```
.Active = True                         ‘オープンする
If Not .Active Then                    ‘オープンできたか？
```

これを Visual Basic では代入演算子（＝）の右辺にあるか左辺にあるかで見分けることができます。If 文の記述を書き直すと以下のようになります。

❑ Visual Basic の場合

```
Dim bResult As Boolean
bResult = .Active
If Not bResult Then                    ‘オープンできたか？
```

これで Active プロパティが右辺にあることがはっきり分かるようになりました。まあここまで明示的に書く必要はありませんが、プロパティが右辺にあるか左辺にあるかは常に意識した方がいいでしょう。

ところでこのプログラムはちょっと見ただけでは Active という名前の変数に値を代入しているようにしか見えないのではないでしょうか？オープンというのは変数ではなく動作なので従来型のプログラマにとっては理解に苦しむ実装です。

現実の世界で考えてみましょう。ここに“SECS 通信ボックス”なる装置があったとします。これはパソコンに装着すると自動的に SECS メッセージの送受信ができるというものです。この装置にはボーレートダイヤルやマスター／スレーブスイッチなどがあり適切に設定してから電源スイッチを入れると動作を始めます。ではこの装置が動作しているかどうかを調べるにはどうしたらいいでしょうか？答えは簡単です。電源スイッチが入っているかどうかを見ればいいのです。

このデザイン思想で“SECS 通信ボックス”をソフトウェアに置き換えようとすると、少なくとも『関数』などという現実の世界で形として見えないものは排除しなければなりません。これがこのコントロールの設計思想に一貫してあらわれています。

8　プログラミングガイド（Visual Basic 編）

この章では SECS のミニホストを作成しながらプログラミングについて解説していくことにします。開発する言語は Visual Basic を使用します。

8.1 ミニホストの仕様

作成するミニホストの仕様は以下のようなものとし、通信相手は一般的なウェハ検査装置とします。

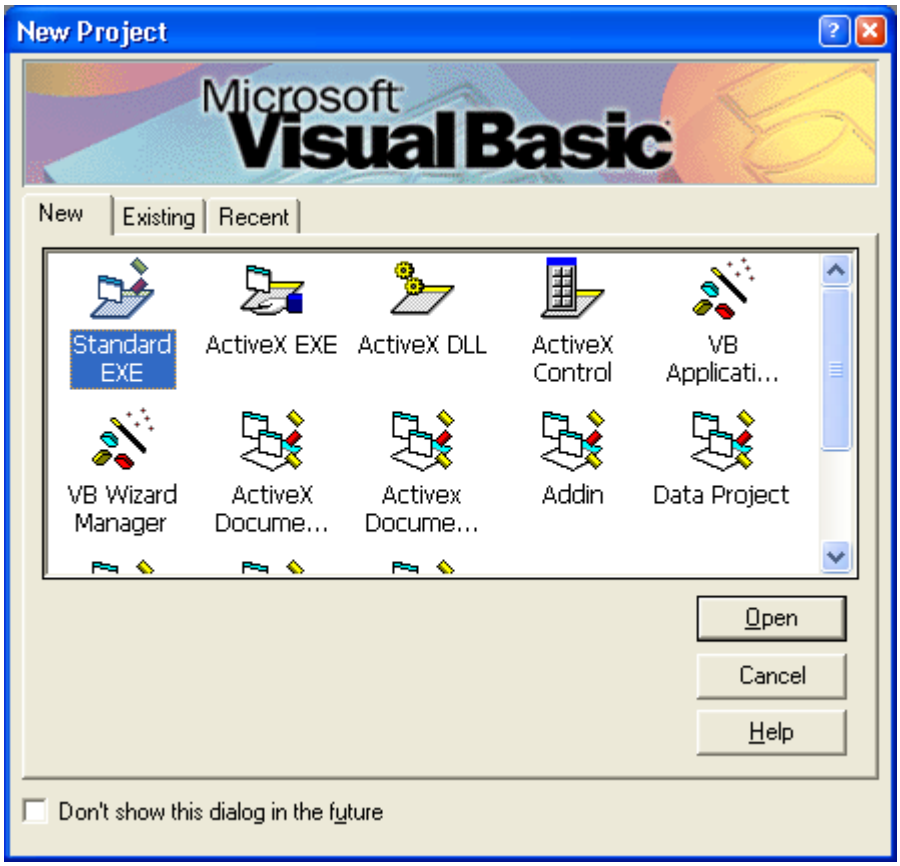
- 装置に対してオンライン移行、レシピ指定、測定開始、データ受信を行うことができます。
- 送信できるメッセージは以下のもののみとします。
  - S1F13W
  - S2F41W
  - S6F12
- 受信できるメッセージは以下のもののみとします。
  - S1F14
  - S2F42
  - S6F11W
- 装置の初期設定はすでに設定されているものとします。
- ストリーム 9 やファンクション 0 の処理は行わない。
- T3 タイムアウトの監視はしない。

8.2 ミニホストを作ってみよう

8.2.1　デフォルトのプログラムグループを作成する

- Visual Basic を起動する

以下のようなダイアログボックスが現れたらキャンセルボタンを押して、いったん Visual Basic を空の状態で立ち上げてみます。

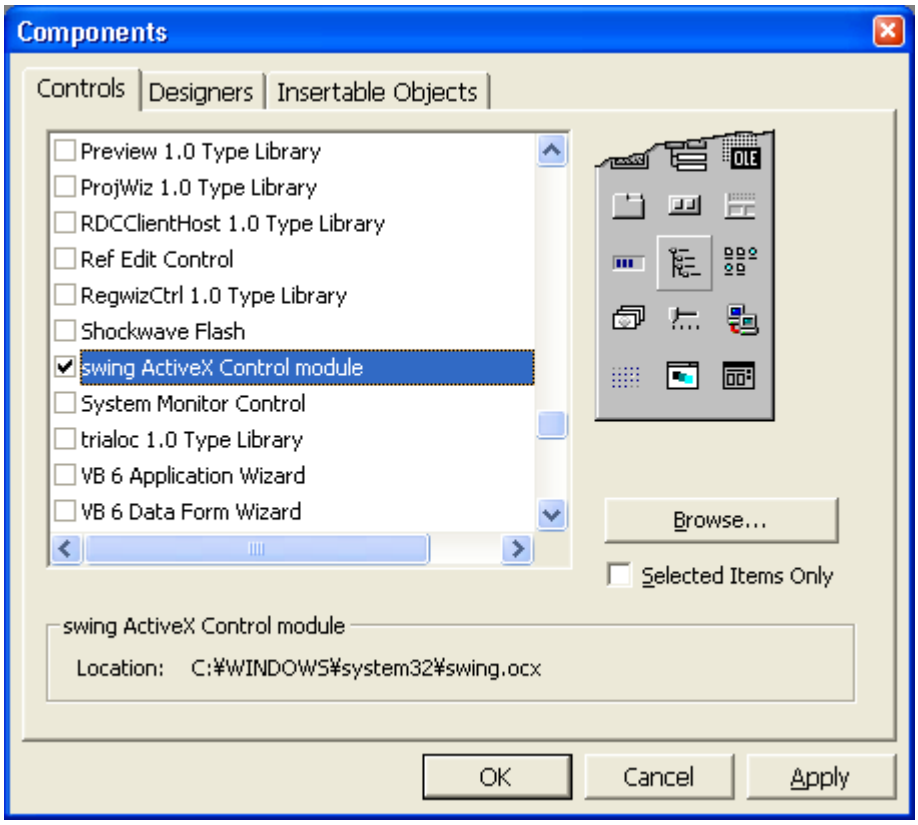


❑ デフォルトのプロジェクトの作成

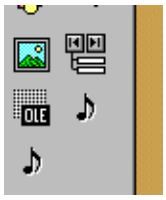
ツールバーから下のボタンを押してデフォルトのプログラムグループを作成します。

8.2.2 Swing ActiveX コントロールの挿入

メニューから『プロジェクト』―『コンポーネント』を選択し Swing ActiveX Control module にチェックマークを付け、ＯＫボタンを押します。

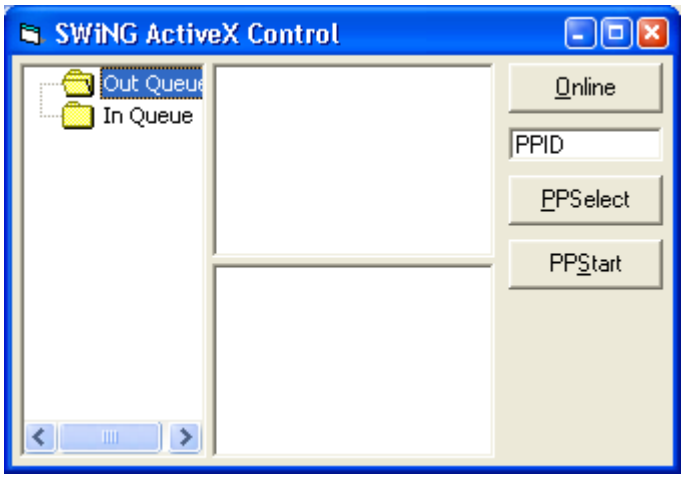


Swing ActiveX コントロールがツールボックスに追加されるのに気が付くはずです。



8.2.3　コントロールの貼り付け

フォームに SwingSecsI、SwingSecsII コントロールを貼り付けてみましょう。SwingSecsII は２つ貼り付けて配列にしておくと便利です。こうすることで受信と送信のメッセージを変数のように別々に扱うことができるからです。



またボタン、ラベル、テキストなどを図のように配置しておきます。ボタンも配列にしておくとすっきりしたプログラムとなります。

プログラミングスタイルまで押し付けようとは考えていませんので、このあたりは各人の好みで自由にアレンジしてもいっこうに構いません。

8.2.4　ポートをオープンする

ポートのオープン処理は *FormLoad* イベント中に記述しましょう。 こうすることでアプリケーション起動時にすぐ通信が可能となります。

```
.Active = True
If Not .Active Then
  MsgBox "エラー：通信ポートをオープンできません！"
End If
```

実務アプリケーションを作成する場合は『オープン』ボタンなどを追加し、これを押すことで初めてオープンするようにした方がいいでしょう。オープンに成功するとは限りませんし、いきなりオープンしたくない場合もあるかもしれないからです。

8.2.5　オンライン移行を送信する

オンライン移行コマンドを送信するにはボタンのハンドラの中から *Send* メソッドを使います。配列にしておいた SwingSecsII コントロールはインデックス 0 を受信用として、インデックス 1 を送信用として使用することにします。 ホストから装置に S1F13 を投げる場合は空のリストを送信してやります。

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

SwingSecsI コントロールの *Send* メソッドには引数として SwingSecsII コントロールの *Msg* プロパティを渡してやります。

8.2.6　レシピ指定を送信する

レシピ指定は S2F41 で PP-SELECT を送信します。

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

8.2.7　測定をかける

測定開始は S2F41 で START を送信します。

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

8.2.8　装置イベントを読む

受信したメッセージは *Read* イベントで報告されます。このイベントハンドラの始めの部分で SwingSecsII コントロールに受信した内容を渡してやります。これにより他のプロパティの内容を参照することができるようになります。

```
.Msg = pszMsg
```

装置からは S6F11 でイベントが送られてくるものとします。全てのイベントを拾わず以下の CEID だけとします。

```
10   レシピ指定に成功
11   レシピ指定に失敗
20   測定完了
30   測定データ
```

CEID は”2”のノード位置に格納されてきます。この値が何かによってそれぞれの処理を記述することになります。

```
'CEID
.Pointer = "2"
Select Case CInt(.Value)
Case 10
  MsgBox "レシピ指定に成功した"
Case 11
  MsgBox "レシピ指定に失敗した"
Case 20
  MsgBox "測定が完了した"
Case 30
  '測定データ
End Select
```

S6F11 は返答を期待していますのでバイナリ 0 の２次メッセージを送信します。

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

このほかに S1F14 と S2F42 も受信しますが、同様に記述してやればよいでしょう。

8.2.9　実行

以上でミニホストは完成です。シンプルではあるがホストとしての最低限の機能は持っています。にも関わらずプログラムのリストは僅か 70 行足らずです。同じ機能のソフトウェアをコントロールを使わずに自分でスクラッチから作成することを考えてみれば、いかに工数を短縮できたかが分かるというものです。

```
Option Explicit

Private Sub Command1_Click(Index As Integer)
  With SwingSecsII1(1)
    Select Case Index
    Case 0
      'オンライン移行
      .List = "s1f13w{}"
    Case 1
      'レシピ指定
      .List = "s2f41w{<a'PP-SELECT'>{{<a'PPID'><a'" + Text1.Text +
"'>}}}"
    Case 2
      '測定開始
      .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 "エラー：通信ポートをオープンできません！"
    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 "オンライン移行した"
      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 "レシピ指定に成功した"
      Case 11
        MsgBox "レシピ指定に失敗した"
      Case 20
        MsgBox "測定が完了した"
      Case 30
        '測定データ
      End Select
    End Select
  End Select
 End With
End Sub
```

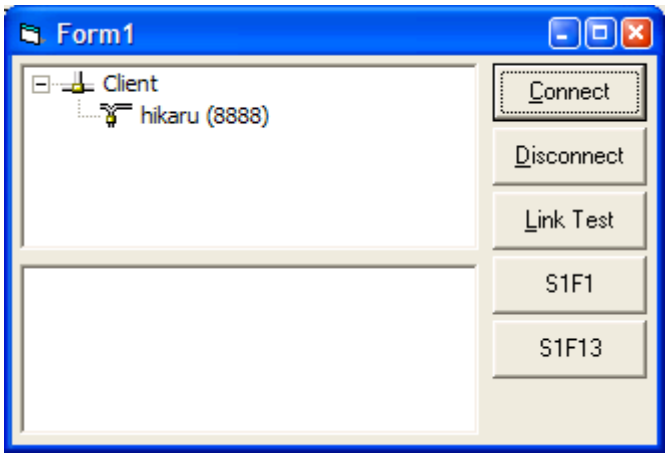
8.3 HSMS クライアントを作ってみよう

8.3.1 準備

最初にミニホストの時と同じように、デフォルトのプログラムグループを作成し、コントロールを挿入しておきます。

8.3.2 Paste ActiveX Control

SwingHsms コントロールと SwingSecsII コントロールをフォームに貼り付けます。今回は説明を簡単にするために SwingSecsII コントロールは一個だけ貼り付けます。



8.3.3 サーバと接続してみる

アプリケーションが起動した直後に接続を有効にするために、*FormLoad* イベントに以下のように追加してやります。

```
'接続した
.Active = True
```

8.3.4 送信の準備

HSMS のメッセージヘッダは SECS-I とは異なります。このため、送信する前に専用の送信関数を作成して、若干の修正を加えてやる必要があります。

```
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　セレクト要求を送信する

HSMS-SS（HSMS シングルセッション）の場合はクライアント（アクティブエンティティ）側が‘Select Request’（セレクト要求）をサーバ（パッシブエンティティ）に送る必要があります。

```
'接続に成功したか？
If .Active Then
  ' Select.Req 送信
  SwingSecsII1.List = "SelectReq"
  Send False
End If
```

8.3.6　HSMS メッセージの受信

HSMS で定義されているメッセージのタイプは以下のものがあります。

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

Read イベントでは毎回受信したメッセージをチェックしてやる必要があります。

```
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　リンクテスト応答を返信する

リンクテスト要求を受信したら、直ちにリンクテスト応答を返さなければなりません。これを送るには以下のように記述します。

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

8.3.8　実行

わずか１００行足らずという簡単さで HSMS クライアントの作成は完了です。お客様向けに出荷するシステムに仕上げるにはこのサンプルプログラムに機能を追加していけばいいでしょう。同じようなプログラムを一から作成することを考えると、信じられないほど簡単に作成できることが分かっていただけたと思います。

```
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 は SECS I のプロトコルを実装した Active X Control です。このコントロールをアプリケーションに組み込むことにより非常に簡単に SECS の送受信が可能となります。基本的には SwingSecsII コントロールと組み合わせて使います。

9.1 リファレンス

いくつかの用語がでてくるので簡単に説明します。

❑ 永続化プロパティ

デザイン時にこのプロパティを設定するとその値をリソースに保存します（永続化）。このため実行時にいちいちプロパティをセットする必要はありません。たとえばコントロールを非表示に決めた場合はデザイン時に Show プロパティを False に設定します。

9.1.1 Active

このプロパティに True をセットすると CommPort プロパティと BaudRate プロパティで指定された条件でポートをオープンします。オープンできたかどうかを調べるには Active プロパティが True になっているかを調べます。False をセットするとポートをクローズします。アプリケーションを終了するときにポートをクローズし忘れても心配ありません。コントロールが自動的にクローズしてくれるからです。

❑ Visual Basic の場合

```
.CommPort = 0                          ‘Com1
.BaudRate = 9600                       ‘9600 bps
.Active = True                         ‘Open
If Not .Active then
  MsgBox “エラー：通信ポートをオープンできません！”
End If
```

❑ Visual C++の場合

```
m_secs.SetCommPort(0);                 // Com1
m_secs.SetBaudRate(9600);              // 9600 bps
m_secs.SetActive(true);                // Open
if(m_secs.GetActive())
  MessageBox(“エラー：通信ポートをオープンできません！”);
```

9.1.2 Appearance

表示状態を設定します。このプロパティが 1 のときはくぼんだ状態で表示されます。このプロパティは永続化プロパティです。

❑Visual Basic の場合

```
  .Appearance = 0                         'フラット
  .Appearance = 1                         'くぼみ
```

❑Visual C++の場合

```
  m_secs.SetAppearance(0);                // フラット
  m_secs.SetAppearance(1);                // くぼみ
```

9.1.3 BaudRate

ボーレートを指定します。ボーレートの値は数字で直接指定してやります。このプロパティは永続化プロパティです。

❑Visual Basic の場合

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

❑Visual C++の場合

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

9.1.4 BorderStyle

枠線の有無を設定します。このプロパティが 1 のときは枠線が表示されます。このプロパティは永続化プロパティです。

❑Visual Basic の場合

```
  .BorderStyle = 0                        '枠なし
  .BorderStyle = 1                        '枠つき
```

❑Visual C++の場合

```
  m_secs.SetBorderStyle(0);               // 枠なし
  m_secs.SetBorderStyle(1);               // 枠つき
```

9.1.5 CommPort

通信ポートの番号を指定します。通信ポートの番号は 0 から始まります。つまり Com1 は 0、Com2 は 1 となります。このプロパティは永続化プロパティです。

❑Visual Basic の場合

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

❑Visual C++の場合

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

9.1.6 DeviceID

デバイスＩＤです。このプロパティは永続化プロパティです。

❑Visual Basic の場合

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

❑Visual C++の場合

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

このプロパティはメッセージの内容であるため本来 SECS II に属するべきものといえます。そのため SwingSecsII コントロールにも DeviceID プロパティが存在します。

☞ワンポイント
ではこの二つが異なっていたらどうなるでしょうか？SwingSecsI では送信メッセージ中のデバイスＩＤが異なっていても自動的に訂正しますので、SwingSecsII コントロールの DeviceID プロパティは無視されます。

9.1.7 IniFile

プロパティの内容を保存する.ini ファイル名です。フルパスで指定した場合はそのディレクトリに作成します。ディレクトリを指定せずファイル名のみの場合は Windows のディレクトリに作成します。このプロパティは永続化プロパティです。

❑Visual Basic の場合

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

❑Visual C++の場合

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

拡張子は必ずしも ini である必要はありませんが習慣上 ini にするのが普通です。

9.1.8 IniSection

プロパティの内容を保存する.ini ファイル中のセクション名です。１つのプロジェクトに複数の SwingSecsI コントロールが埋め込まれていても、別々のセクション名を指定することで設定内容を変えることができます。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

9.1.9 Log

ログを記録するかどうかを指定します。このプロパティが True なら LogFile プロパティで指定したファイルに記録し、False なら記録しません。ただしこのログは SECS の（低レベルな）やり取りが記録されるのみです。メッセージ内容を記録する場合は SwingSecsII コントロールを併用します。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

```
.Log = True                             ‘ログに記録する
.Log = False                            ‘ログに記録しない
```

❑ Visual C++の場合

```
m_secs.SetLog(true);                    // ログに記録する
m_secs.SetLog(false);                   // ログに記録しない
```

9.1.10 LogFile

Log プロパティが True なら LogFile プロパティで指定されたファイルにログを記録します。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

```
.LogFile = “C:\Log\Secs.log”
```

❑ Visual C++の場合

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

フルパスで指定しないとコントロールのあるディレクトリにログファイルを作成します。デフォルトのインストールではこれは Windows の System ディレクトリです。

9.1.11 Master

True ならマスター、False ならスレーブであることをあらわします。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

```
.Master = True                          'マスター
.Master = False                         'スレーブ
```

❑ Visual C++の場合

```
m_secs.SetMaster(true);                 // マスター
m_secs.SetMaster(false);                // スレーブ
```

9.1.12 MSEC

SwingSecsI コントロールでは三菱の MSEC プロトコルもサポートしています。True なら MSEC、False なら SECS であることをあらわします。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

```
.MSEC = True                            'MSEC
.MSEC = False                           'SECS
```

❑ Visual C++の場合

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

9.1.13 Retry

再試行回数を指定します。デフォルトは 3 回です。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

```
.Retry = 3                              'リトライ 3 回
```

❑ Visual C++の場合

```
m_secs.SetRetry(3);                     // リトライ 3 回
```

9.1.14 Show

このプロパティが True ならキューの状態を画面に表示します。 False のときはオープンされているかどうかを LED 風のビットマップで表示します。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

```
.Show = True                          ‘キューを表示
.Show = False                         ‘ＬＥＤを表示
```

❑ Visual C++の場合

```
m_secs.SetShow(true);                 // キューを表示
m_secs.SetShow(false);                // ＬＥＤを表示
```

9.1.15 T1

T1 タイムアウトをミリ秒単位で指定します。デフォルトは 0.5 秒です。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

```
.T1 = 500                             ‘500msec
```

❑ Visual C++の場合

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

9.1.16 T2

T2 タイムアウトをミリ秒単位で指定します。デフォルトは 10 秒です。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

```
.T2 = 10000                           ‘10sec
```

❑ Visual C++の場合

```
m_secs.SetT2(10000);                  ‘10sec
```

T2、T4 タイムアウトの時間は以下の計算式に当てはまるべきです。

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

9.1.17 T3

T3 タイムアウトをミリ秒単位で指定します。デフォルトは 45 秒です。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

```
.T3 = 45000                              ‘45sec
```

❑ Visual C++の場合

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

T3 タイムアウトが発生しても自動的には S9F9 を送信しません。このためアプリケーション側で処理するようにします。

9.1.18 T4

T4 タイムアウトをミリ秒単位で指定します。デフォルトは 45 秒です。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

```
.T4 = 45000                              ‘45sec
```

❑ Visual C++の場合

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

このタイムアウトが発生するとキューの内容を破棄してメモリーを解放します（ガベージコレクション機構）。 T2、T4 タイムアウトの時間は以下の計算式に当てはまるべきです。

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

9.1.19 Config

プロパティを編集するダイアログボックスを表示します。

❑ Visual Basic の場合

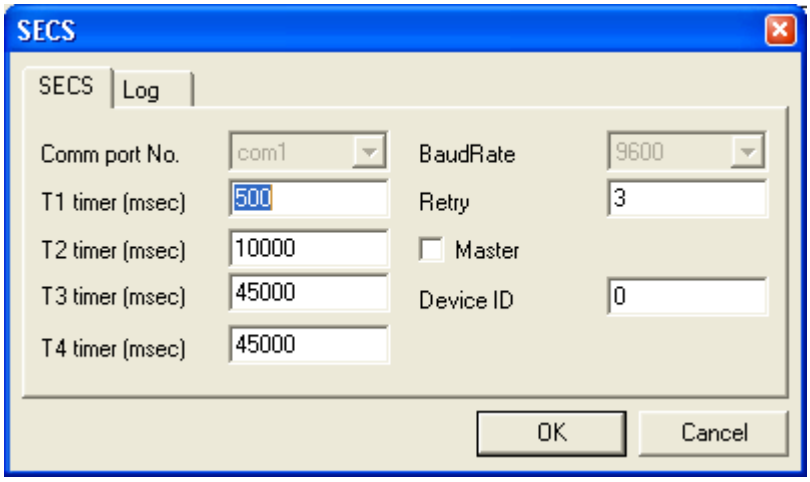
```
Function Config(pszTitle As String) As Boolean
```

❑ Visual C++の場合

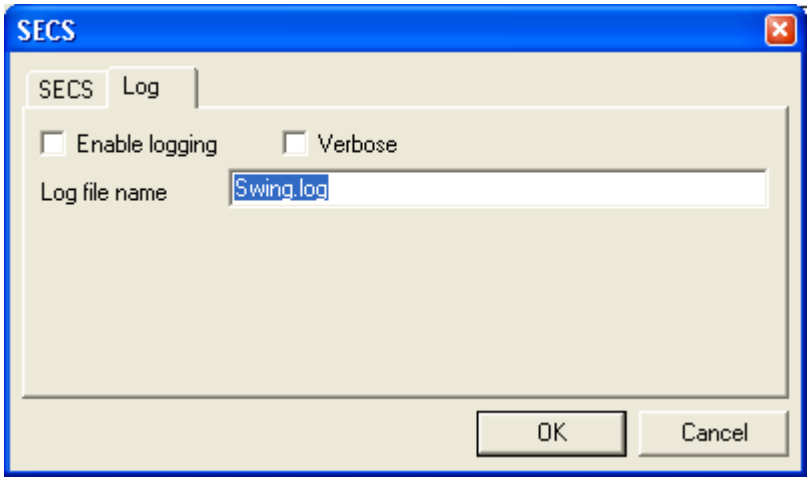
```
BOOL Config(LPCTSTR pszTitle)
```

pszTitle　　ダイアログボックスのキャプションタイトル。

ユーザがＯＫボタンをクリックしてダイアログボックスを閉じると IniFile プロパティ IniSection プロパティで指定されたファイルにプロパティを保存します。.ini ファイルに書き込んだ場合は True が返ります。.ini ファイルに保存された内容を取り出してプロパティにセットするには LoadIni メソッドを呼び出します。



ポートをオープン中（Active プロパティが True のとき）にこのメソッドを呼び出しても CommPort プロパティと BaudRate プロパティの値を変更することはできません。



❑Visual Basic の場合

```
.Config “SECS Configuration”
```

❑Visual C++の場合

```
m_secs.Config(“SECS Configuration”);
```

引数に NULL を指定した場合は、

9.1.20 LoadIni

プロパティを IniFile プロパティ IniSection プロパティで指定されたファイルから読み込みます。
読み込みができない場合はリソースに埋め込まれている値がセットされます。
このメソッドは Config メソッドで保存されたプロパティを取り出すためアプリケーションの起動時に呼び出すとよいでしょう。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

9.1.21 Send

メッセージを送信します。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

pszMsg　　送信するメッセージ。

この時点ではメッセージは送信キューの最後に入ります。送信に成功すると Written イベントが発生します。

❑ Visual Basic の場合

```
  .Send SwingSecsII1.Msg
```

❑ Visual C++の場合

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

9.1.22 Errors

このイベントは通信エラーが起こった場合に発生します。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

sError	エラーコード。エラーコードは以下のいずれかの値です。	
	SecsErrorUnexpectedChar	予期しない文字を受信（=1）
	SecsErrorBadLength	レングスバイト異常（=2）
	SecsErrorSum	チェックサム不良（=3）
	SecsErrorParity	パリティ異常（=4）
	SecsErrorBadMultiBlockNumber	マルチブロック中のブロック番号異常（=5）
	SecsErrorBlockNumber	ブロック番号が 0 でも 1 でもない（=6）
	SecsErrorNak	NAK 受信（=7）
	SecsErrorRetry	リトライオーバー（=8）
	SecsErrorT4	T4 タイムアウト（=9）
	SecsErrorT2	T2 タイムアウト（=10）
	SecsErrorT1	T1 タイムアウト（=11）
	SecsErrorT3	T3 タイムアウト（=12）
	SecsErrorDeviceID	デバイス ID が不一致（=13）
pszInfo	追加情報。現在のところは未使用となっています。	

9.1.23 Read

このイベントはメッセージを正常に受信したら発生します。マルチブロックメッセージの場合は最終ブロックを受信した後に発生します。受信したメッセージは T4 タイムアウトで設定した時間が経過すると自動的に削除されます（ガベージコレクション機構）。

❑ Visual Basic の場合

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

❑ Visual C++の場合

```
void OnRead(LPCTSTR pszMsg);
```

pszMsg	受信したメッセージです。マルチブロックメッセージの場合ヘッダ部は全て同じなので１つにまとめられます。

9.1.24 SelMsg

このイベントはユーザがリストコントロールに表示されているメッセージを選択したときに発生します。

❑ Visual Basic の場合

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

❑ Visual C++の場合

```
void OnSelMsg(LPCTSTR pszMsg);
```

pszMsg	選択されたメッセージ。

9.1.25 Written

このイベントはメッセージを送信できたら、あるいは送信に失敗したら発生します。マルチブロックメッセージの場合は最終ブロックを送信し終わった後に発生します。このイベントが発生した時には送信の成功／失敗にかかわらず、既に送信キューから削除されています（ガベージコレクション機構）。送信される順序は常に古いメッセージから順に１つずつ送信されていきます（FIFO = 先入れ先出し）。このためイベントの発生する順序も Send メソッドを発行した順と同じとなります。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

pszMsg	送信されたメッセージを１６進アスキー変換した文字列です。
bOK	送信に成功した場合は True(=1)が、失敗した場合は False(=0)がセットされます。False がセットされているときは、リトライ回数分の再試行が行われてもダメだった場合です。

10 SwingSecsII

SwingSecsII は SECSII の構造を実装した Active X Control です。受け取ったメッセージを解析してツリー状に展開します。SwingSecsI コントロールや SwingHsms コントロールと組み合わせて使うことができます。

10.1 リファレンス

いくつかの用語がでてくるので簡単に説明します。

- 永続化プロパティ

デザイン時にこのプロパティを設定するとその値をリソースに保存します（永続化）。このため実行時にいちいちプロパティをセットする必要はありません。たとえばコントロールを非表示に決めた場合はデザイン時に Show プロパティを False に設定します。

- ノード

ノードは“/”（スラッシュ）とノード番号で構成されます。ノードが“”（空）の場合はルートが指定されたとみなされます。一般にルートはリスト型であることが多いのですが他の型も指定できます。子ノードがある場合は必ずルートがリスト型となります。

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

Kelly, Brenda, Donna, Valerie,Andrea をノード位置の記述であらわすと以下のようになります。

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

ノードのネスティングレベルに制限はありません。

10.1.1 Appearance

表示状態を設定します。このプロパティが 1 のときはくぼんだ状態で表示されます。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

```
.Appearance = 0                         ‘フラット
.Appearance = 1                         ‘くぼみ
```

❑ Visual C++の場合

```
m_msg.SetAppearance(0);                 // フラット
m_msg.SetAppearance(1);                 // くぼみ
```

10.1.2 Array

ノードに含まれているアイテムの個数です。配列でなければ 1 になります。リストの場合はノードの数となります。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

このプロパティは読み出し専用です。値をセットすることはできません。

10.1.3 BlockNumber

ブロック番号です。マルチブロックメッセージを受信した場合は最終ブロックのブロック番号がセットされます。これによって受信したメッセージが何ブロックに分割されて送られてきたかが分かります。送信時は必ず１にセットします。

❑ Visual Basic の場合

```
.BlockNumber = 1                        ‘ブロック番号 = 1
```

❑Visual C++の場合

```
m_msg.SetBlockNumber(1);                // ブロック番号 = 1
```

10.1.4 BorderStyle

枠線の有無を設定します。このプロパティが 1 のときは枠線が表示されます。このプロパティは永続化プロパティです。

❑Visual Basic の場合

```
.BorderStyle = 0                        ‘枠なし
.BorderStyle = 1                        ‘枠つき
```

❑Visual C++の場合

```
m_msg.SetBorderStyle(0);                // 枠なし
m_msg.SetBorderStyle(1);                // 枠つき
```

10.1.5 DeviceID

デバイスＩＤです。Init メソッドを呼び出すとこの値で初期化されます。

❑Visual Basic の場合

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

❑Visual C++の場合

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

このプロパティは永続化プロパティです。

10.1.6 EBit

エンドビットです。最終ブロックで True になります。SwingSecsI コントロールは最終ブロックまで受信してからイベントを発生するためこのプロパティは必ず True になります。

❑Visual Basic の場合

```
If .EBit = False Then
  ‘ここにくることはない
End If
```

❑ Visual C++の場合

```
if(!m_msg.GetEBit())
{
  // ここにくることはない
}
```

一方、HSMS は Ebit プロパティを使用せずＰタイプとなっています。このため SwingHsms コントロールでは Ebit プロパティが必ず True になるという保証はありません。

10.1.7 Fucntion

ファンクション番号です。

❑ Visual Basic の場合

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

❑ Visual C++の場合

```
if(m_msg.GetStream()==2 && m_msg.GetFucntion()==42)

  // s2f42
  ...
}
```

10.1.8 List

このプロパティを読み出すとコントロールにセットされているメッセージの構造をツリー形式で取得します。またこのプロパティにツリー形式をセットすることもできます。セットする場合、文法中に改行コードやスペース、タブなどを自由に入れても構いません。ただし全角文字は使用できません。

List プロパティを使用するとかなり簡潔に記述することができます。Add プロパティを使って記述したものと比べるとその違いは顕著です。以下の２つは同じメッセージを作成します。

❑ Add を使用した場合

❑ Visual Basic の場合

```
.Init
.Stream = 1
.Fucntion = 13
.WBit = True
.Pointer = “”
.Add SecsTypeList, ””
.Pointer = “2”
.Add SecsTypeAscii, “Swing”
.Add SecsTypeAscii, “Ver 2.71”
```

❑ Visual C++の場合

```
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 2.71");
```

❑ List を使用した場合

❑ Visual Basic の場合

```
.List = "s1f13w{<A'Swing'><A'Ver 2.71'>}"
```

❑ Visual C++の場合

```
m_msg.SetList("s1f13w{<A'Swing'><A'Ver 2.71'>}");
```

List プロパティにセットする文字列の構文は以下のようになっています。

❑ 一般的な注意

ホワイトスペース（スペース、タブ、改行、復改コード）は区切り文字としての意味しかありません。このため適度にタブや改行コードを挿入することで見易くすることができます。ただしコメント中および文字列中は文字として扱われます。

アスタリスク（*）から行末まではコメントとなります。ただし文字列中のアスタリスクは除きます。

整数は 0～9 までの文字とマイナス（-）から構成されます。16 進数で記述したい場合は'0x'を先頭に付加します。この場合は a～f と A～F までの文字も使用できます。小数は'0.9'を'.9'というように'0'を省略して記述することもできます。指数表現も可能です。また予約語として true（=1）と false（=0）を使うこともできます。

文字列はシングルクォーテーション（'）で囲まれた範囲です。文字列中には改行コードとシングルクォーテーション自身を含めることはできません。このためどうしてもこれらの文字を入れたい場合は、0x0a などのように１６進数表現を併用します。

説明文中の太字部分はその文字を記述することを表します。基本的にこれらの文字は大文字でも小文字でも構いません。斜体字はそれぞれの説明を参照してください。また[]で囲まれた部分は省略することができます。

❑ 構文

```
[sxxfyy[w]] Body
```

xx	ストリーム番号。文字's'、'f'との間にはスペースを入れません。
yy	ファンクション番号。文字'f'、'w'との間にはスペースを入れません。
w	ウェイトビット。指定する場合は'w'と記述します。省略可能です。
Body	メッセージのボディ。

ストリーム、ファンクション、ウェイトビットはひとかたまりで認識するため、これらの間にスペースや改行コードを入れないようにします。またストリーム、ファンクションを全て省略してメッセージボディのみを記述することもできます。

❑ メッセージボディ
メッセージのボディは階層構造になっています。

10.1.8.1 リスト

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

Number          リストの数です。SECSIM との互換性のためだけに用意されています。この数字は無視されます。
Body            メッセージのボディ。他のアイテムを並べることができます。
```

10.1.8.2 アスキー文字列

```
<a [Strings]>

Strings         文字列です。長い文字列は分割して記述することもできます。また直接文字コードを記述することもできます。例えば

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

                これは

                <a ‘ABCDEF0123456789’>

                と同じです。
```

10.1.8.3 2 バイト文字列

```
<a2 [Strings]>

Strings         2 バイト文字列です。現在のバージョンでは MBCS にのみ対応しています。
```

10.1.8.4 JIS8 文字列

```
<j [Strings]>

アスキー型と同じように扱われます。

Strings         文字列です。長い文字列は分割して記述することもできます。また直接文字コードを記述することもできます。例えば

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

                これは

                <j ‘ABCDEF0123456789’>

                と同じです。
```

10.1.8.5 整数

```
<i1[Numbers]>
<i2[Numbers]>
<i4[Numbers]>
<i8[Numbers]>
<u1[Numbers]>
<u2[Numbers]>
<u4[Numbers]>
<u8[Numbers]>

Numbers          数値です。それぞれ以下の意味となります。

                 i1                                  符号付き 8 ビット整数
                 i2                                  符号付き 16 ビット整数
                 i4                                  符号付き 32 ビット整数
                 i8                                  符号付き 64 ビット整数
                 u1                                  符号なし 8 ビット整数
                 u2                                  符号なし 16 ビット整数
                 u4                                  符号なし 32 ビット整数
                 u8                                  符号なし 64 ビット整数

                 いくつかの数字を並べて記述することもできます。この場合は配列となり
                 ます。例えば

                 <i1 1 0x02 3>

                 のように記述することができます。

現在のバージョンでは i8 と u8 に巨大な値を入れることはできません。
```

10.1.8.6 浮動小数点数

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

Fnumbers         浮動小数点数です。それぞれ以下の意味となります。

                 f4                                  32 ビット浮動小数点数
                 f8                                  64 ビット浮動小数点数

                 例えば

                 <f4 0 1.0 3.14>

                 のように記述します。
```

10.1.8.7 バイナリ

```
<b [Numbers]>
```

```
Numbers          数値です。例えば

                 <b 0xff 0x3e 255 0>

                 のように記述します。
```

10.1.8.8 ブーリアン

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

Numbers          数値です。例えば

                 <bool true false 1 0>

                 のように記述します。
```

10.1.9 Msg

このプロパティに SwingSecsI コントロールの Read イベントで受信文字列をセットすると各プロパティを更新してツリー表示します。またこのプロパティの内容を取り出す（ゲットする）と各プロパティの内容から文字列を作成して返します。

❑ Visual Basic の場合

```
Private Sub SwingSecsI1_Read(ByVal pszMsg As String)
  With SwingSecsII1I1
    .Msg = pszMsg
    Select Case .Stream
    Case 1
      Select Case .Fucntion
      Case 1
        ‘s1f1
        ...
```

❑ Visual C++の場合

```
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

操作対象となるノードを指定します。ノードは“/”（スラッシュ）とノード番号と“[”“]”（カギカッコ）で構成されます。ノード番号は 1 から始まる数字です。ノードが空文字列の場合はルート

が指定されたとみなされます。ノードは MS-DOS のディレクトリ構造に例えることができます（説明の中で『ノード』と記述されている部分を『ディレクトリ』と置き換えると理解しやすいと思います）。

ノードを作成する場合にはまずノード指定してそれに対して追加命令を発行します。

❑ Visual Basic の場合

```
.Pointer = “1”
.Add SecsTypeAscii, “abc”
```

❑ Visual C++の場合

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

ノード指定の文字列の先頭にスラッシュを付けても構いません（無視されます）。

❑ Visual Basic の場合

```
.Pointer = “”
.Pointer = “/”                          ‘これらは同じ

.Pointer = “1/2”
.Pointer = “/1/2”                       ‘これらも同じ
```

❑ Visual C++の場合

```
m_msg.SetPointer(“”);
m_msg.SetPointer(“/”);                  // これらは同じ

m_msg.SetPointer(“1/2”);
m_msg.SetPointer(“/1/2”);               // これらも同じ
```

ノードの下にノードを作ることができます。この場合は子ノードより先に親ノードを作らなければなりません。ノードを作る順番は実際の SECS のバッファの順序と異なっていても構いません。

```
1 :  <L
2 :    <L
3 :      <A‘abc’>
4 :    >
5 :    <L
6 :      <A‘def’>
7 :    >
8 :  >
```

作成する順序は 1→2→3→5→6 でも 1→2→5→3→6 でも 1→2→5→6→3 でも構いません。

❑　1→2→3→5→6 の場合

❑ Visual Basic の場合

```
  .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++の場合

```
  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 の場合

❑ Visual Basic の場合

```
  .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++の場合

```
  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 の場合

❑ Visual Basic の場合

```
  .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++の場合

```
  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
```

ノードの値を読み出す場合も先にノードを指定してから読み出します。

❑Visual Basic の場合

```
  .Pointer = "1"
  If .Value = "abc" Then
    ...
  End If
```

❑Visual C++の場合

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

ノードの内容が配列だった場合はスペースコードで区切られて返されます。

❑Visual Basic の場合

```
  .Pointer = "1"
  .Add SecsTypeBool, "1"
  .Add SecsTypeBool, "0"
  ...
  .Pointer = "1"
  Dim strBuf As String
  strBuf = .Value                        '"1 0"が返る
```

❑Visual C++の場合

```
  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"が返る
```

配列の１つの要素だけを取り出したい場合はカギカッコを用います。配列のインデックスは 0 から始まります。

❑Visual Basic の場合

```
.Pointer = "1[1]"
Dim strBuf As String
strBuf = .Value                        '"0"が返る
```

❑Visual C++の場合

```
m_msg.SetPointer("1[1]");
CString strBuf=m_msg.GetValue();       // "0"が返る
```

このためルートノードが配列の場合はカギカッコだけというちょっと奇妙な形になります。

❑Visual Basic の場合

```
.List = "<f4 1.414 3.14 2.236>"
.Pointer = "[1]"
Dim strBuf As String
strBuf = .Value                        '"3.14"が返る
```

❑Visual C++の場合

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

ただしカギカッコはアスキーやリストでは無視されます。

10.1.11 PType

Ｐタイプ（プレゼンテーションタイプ）です。HSMS ではほとんど例外なく SECS-II メッセージを使用しますのでこの値は 0 を指定します。

❑Visual Basic の場合

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

❑Visual C++の場合

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

10.1.12 RBit

リバースビットです。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

10.1.13 SessionID

セッションＩＤです。デバイスＩＤと呼ぶこともあります。

❑ Visual Basic の場合

```
If .SessionID <> &HFFFF Then
  MsgBox “Invalid Session ID!”
End If
```

❑ Visual C++の場合

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

10.1.14 Show

このプロパティが True ならポートがオープンされているかどうかを画面に表示します。False のときは背景色で塗るだけでコントロールを描画しません。このプロパティは永続化プロパティです。

Show プロパティが True のときに大量にノード追加を行うと、再描画のために時間がかかる場合があります。このときは行ったん Show プロパティを False にしてからノード追加を行うとよいでしょう。追加が完了したら Show プロパティを True に戻してやります。

❑ Visual Basic の場合

```
.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++の場合

```
  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

ソースＩＤです。

❑ Visual Basic の場合

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

❑ Visual C++の場合

```
  m_msg.SetSourceID(m_msg.GetSourceID());
```

10.1.16 Stream

ストリームです。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

```
    ...
```

10.1.17 SType

Ｓタイプ（セッションタイプ）です。現時点では以下のように定義されています。

値	説明
0	データメッセージ
1	Select.Req
2	Select.Rsp
3	Deselect.Req
4	Deselect.Rsp
5	LinkTest.Req
6	LinkTest.Rsp
7	Reject.Req
8	（未使用）
9	Separate.Req
10	（未使用）
11～127	
128～255	

❑Visual Basic の場合

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

❑Visual C++の場合

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

10.1.18 SystemBytes

システムバイトです。ソースＩＤとトランザクションＩＤを合わせた４バイトの事を指します。
二次メッセージは一次メッセージのシステムバイトと同じでなければなりません。

❑Visual Basic の場合

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

❑Visual C++の場合

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

10.1.19 TransactionID

トランザクションＩＤです。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

10.1.20 Type

ノードの型です。これは以下のいずれかの値となります。

SecsTypeInvalid	0	無効
SecsTypeList	1	リスト
SecsTypeBinary	2	バイナリ
SecsTypeBoolean	3	ブーリアン
SecsTypeAscii	4	アスキー
SecsTypeJis	5	ＪＩＳ８
SecsTypeLong8	6	64 ビット符号つき整数
SecsTypeChar	7	8 ビット符号つき整数
SecsTypeShort	8	16 ビット符号つき整数
SecsTypeLong	9	32 ビット符号つき整数
SecsTypeDouble	10	64 ビット浮動小数点数
SecsTypeFloat	11	32 ビット浮動小数点数
SecsTypeDWord8	12	64 ビット符号なし整数
SecsTypeByte	13	8 ビット符号なし整数
SecsTypeWord	14	16 ビット符号なし整数
SecsTypeDWord	15	32 ビット符号なし整数

このプロパティは読み出し専用です。値をセットすることはできません。

10.1.21 Value

ノードの値を１０進数で返します。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

ノードが配列になっている場合はスペースで区切られて格納されます。

❑Visual Basic の場合

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

❑Visual C++の場合

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

配列の１つの要素だけを取り出したい場合はカギカッコを用います。配列のインデックスは０から始まります。

❑Visual Basic の場合
❑Visual C++の場合

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

❑Visual C++の場合

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

このプロパティは読み出し専用です。値をセットすることはできません。

10.1.22 ValueHex

ノードの値を１６進数で返します。

❑Visual Basic の場合

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

❑Visual C++の場合

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

ノードが配列になっている場合はスペースで区切られて格納されます。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

配列の１つの要素だけを取り出したい場合はカギカッコを用います。配列のインデックスは 0 から始まります。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

このプロパティは読み出し専用です。値をセットすることはできません。

10.1.23 WBit

ウェイトビットです。返信要求がある場合は True となります。

❑ Visual Basic の場合

```
If (.Fucntion Mod 2) And .WBit Then
  ‘Send default reply message
  ...
End If
```

❑ Visual C++の場合

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

10.1.24 Add

ノードを追加します。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

sType	ノードの型です。（☞Type プロパティを参照）
pszValue	ノードの値です。文字列で指定します。リストの場合は無視されます。

10.1.25 Init

すべてのリスト構造を消去します。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

10.1.26 Reply

２次メッセージのヘッダを作成します。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

pszMsg	１次メッセージです。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

11 SwingHsms

SwingHsms は HSMS のプロトコルを実装した Active X Control です。このコントロールをアプリケーションに組み込むことにより非常に簡単に HSMS の送受信が可能となります。このコントロールも基本的には SwingSecsII コントロールと組み合わせて使います。

11.1 リファレンス

いくつかの用語がでてくるので簡単に説明します。

❑ 永続化プロパティ

デザイン時にこのプロパティを設定するとその値をリソースに保存します（永続化）。このため実行時にいちいちプロパティをセットする必要はありません。たとえばコントロールを非表示に決めた場合はデザイン時に Show プロパティを False に設定します。

11.1.1 Active

このプロパティの動作は Serve プロパティの値によって異なります。

まず Server プロパティが True のときは LocalPortNumber プロパティで指定されたポートをオープンして待機状態になります。この時点ではまだコネクションは成立しておらず、クライアントが接続してきたときに初めて成立します。このようにコネクションが成立していなくてもポートがオープンできてクライアントの接続待ち状態になれば Active プロパティの値は True になります。

一方、Server プロパティが False のときは IPAddress プロパティと PortNumber プロパティで指定されたサーバに接続を試みます。IPAddress プロパティには IP アドレスまたはコンピュータ名を指定することができます。サーバが同じコンピュータにある場合は IPAddress プロパティを""（空文字）で指定することもできます。自分のポート番号は LocalPortNumber プロパティで指定できますが、サーバの空いているポート番号を自動的に割り当てさせることも可能です。ポート番号が固定だと再接続が成功するまでに時間がかかってしまうことがありますので、自動割り当てを使うと便利です。自動割り当てを行うには LocalPortNumber プロパティに 0 を指定してやります。サーバとの間にコネクションが成立したら Active プロパティの値は True になります。

False をセットするとコネクションを解除します。アプリケーションを終了するときにコネクションを解除し忘れても心配ありません。コントロールが自動的に解除してくれるからです。サーバ側の Active プロパティに False をセットすると接続していた全てのクライアントとのコネクションが解消されます。

❑ Visual Basic の場合

```
  .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++の場合

```
  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!");
```

接続相手の Active プロパティが False になると接続は切断されます。このためクライアント側なら Active プロパティが自動的に False になるので注意が必要です。

11.1.2 Appearance

表示状態を設定します。このプロパティが 1 のときはくぼんだ状態で表示されます。このプロパティは永続化プロパティです。

❑Visual Basic の場合

```
  .Appearance = 0                      'フラット
  .Appearance = 1                      'くぼみ
```

❑Visual C++の場合

```
  m_hsms.SetAppearance(0);             // フラット
  m_hsms.SetAppearance(1);             // くぼみ
```

11.1.3 BorderStyle

枠線の有無を設定します。このプロパティが 1 のときは枠線が表示されます。このプロパティは永続化プロパティです。

❑Visual Basic の場合

```
  .BorderStyle = 0                     '枠なし
  .BorderStyle = 1                     '枠つき
```

❑Visual C++の場合

```
  m_hsms.SetBorderStyle(0);            // 枠なし
  m_hsms.SetBorderStyle(1);            // 枠つき
```

11.1.4 IniFile

プロパティの内容を保存する.ini ファイル名です。フルパスで指定した場合はそのディレクトリに作成します。ディレクトリを指定せずファイル名のみの場合は Windows のディレクトリに作成します。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

拡張子は必ずしも ini である必要はありませんが、習慣上 ini にするのが普通です。

11.1.5 IniSection

プロパティの内容を保存する.ini ファイル中のセクション名です。１つのプロジェクトに複数の SwingHsms コントロールが埋め込まれていても、別々のセクション名を指定することで設定内容を変えることができます。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

11.1.6 IPAddress

接続するホストの IP アドレスを指定します。IPAddress プロパティには IP アドレスかコンピュータ名を指定することができます。サーバが同じコンピュータにある場合は IPAddress プロパティを""（空文字）で指定することもできます。

❑ Visual Basic の場合

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

❑ Visual C++の場合

```
  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

ローカルポート番号を指定します。サーバ側はポート番号を指定する必要はありませんが、ローカルポート番号は必ず指定する必要があります。逆にクライアント側は PortNumber プロパティにポート番号を必ず指定する必要があります。クライアント側でローカルポート番号を 0 に指定して接続しようとすると、サーバ側の空いているポート番号を自動的に割り当ててくれます。

固定のポート番号を設定する場合はシステムで使用しているポート番号（http サーバなどは 80）があるため、一般的には 1024 以上の値を指定すべきです。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

```
.LocalPortNumber = 0                   ‘自動選択
```

❑ Visual C++の場合

```
m_hsms.SetLocalPortNumber(0);          // 自動選択
```

11.1.8 Log

このプロパティの機能はまだ実装されておりません。

11.1.9 LogFile

このプロパティの機能はまだ実装されておりません。

11.1.10 MaxLength

扱えるメッセージの最大長を指定します。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

```
.MaxLength = 10240                     ‘10KB
```

❑ Visual C++の場合

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

11.1.11 PortNumber

ポート番号を指定します。サーバ側はポート番号を指定する必要はありませんが、ローカルポート番号は必ず指定する必要があります。逆にクライアント側はポート番号は必ず指定する必要があります。クライアント側でローカルポート番号を 0 に指定して接続しようとすると、サーバ側の空いているポート番号を自動的に割り当ててくれます。システムで使用しているポート番号があるため、一般的には 1024 以上の値を指定すべきです。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

```
.PortNumber = 5000
```

❑ Visual C++の場合

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

11.1.12 Selected

セレクト状態かどうかを指定します。セレクト要求を受け取ると SwingHsms は自動的にセレクト状態に移行します。もし何らかの理由でセレクト状態にしたくない場合は Selected プロパティを False にセットし直してやります。セレクト応答を受け取った場合は理由コードによってセレクト状態に移行するかどうかを判断します。このプロパティは T7 タイムアウトにも関連します。

❑ Visual Basic の場合

```
SwingHsms1.Selected = False
```

❑ Visual C++の場合

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

11.1.13 Server

True ならサーバ側、False ならクライアント側であることをあらわします。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

```
.Server = True                          'サーバ
.Server = False                         'クライアント
```

❑ Visual C++の場合

```
m_hsms.SetServer(true);                 // サーバ
m_hsms.SetServer(false);                // クライアント
```

11.1.14 Show

このプロパティが True なら接続の状態を画面に表示します。サーバ側で複数のクライアントが接続されているときは、そのクライアント数だけ表示されます。当然のことですがクライアント側は常に１つのサーバとしか接続することができません。False のときは接続されているかどうかを LED 風のビットマップで表示します。このプロパティは永続化プロパティです。

❑Visual Basic の場合

```
.Show = True                          ‘接続状態を表示
.Show = False                         ‘ＬＥＤを表示
```

❑Visual C++の場合

```
m_hsms.SetShow(true);                 // 接続状態を表示
m_hsms.SetShow(false);                // ＬＥＤを表示
```

11.1.15 T3

T3 タイムアウトをミリ秒単位で指定します。デフォルトは 45 秒です。このプロパティは永続化プロパティです。

❑Visual Basic の場合

```
.T3 = 500                             ‘500msec
```

❑Visual C++の場合

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

11.1.16 T5

このプロパティの機能はまだ実装されておりません。

11.1.17 T6

T6 タイムアウトをミリ秒単位で指定します。デフォルトは 5 秒です。このプロパティは永続化プロパティです。

❑Visual Basic の場合

```
.T6 = 5000                            ‘5sec
```

❑Visual C++の場合

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

11.1.18 T7

T7 タイムアウトをミリ秒単位で指定します。デフォルトは 10 秒です。このプロパティは永続化プロパティです。

```
❑ Visual Basic の場合
  .T7 = 10000                              ‘10sec

❑ Visual C++の場合
  m_hsms.SetT7(10000);                     // 10sec
```

11.1.19 T8

T8 タイムアウトをミリ秒単位で指定します。デフォルトは 0.5 秒です。このプロパティは永続化プロパティです。

```
❑ Visual Basic の場合
  .T8 = 500                                ‘500msec

❑ Visual C++の場合
  m_hsms.SetT8(500);                       // 500msec
```

11.1.20 Config

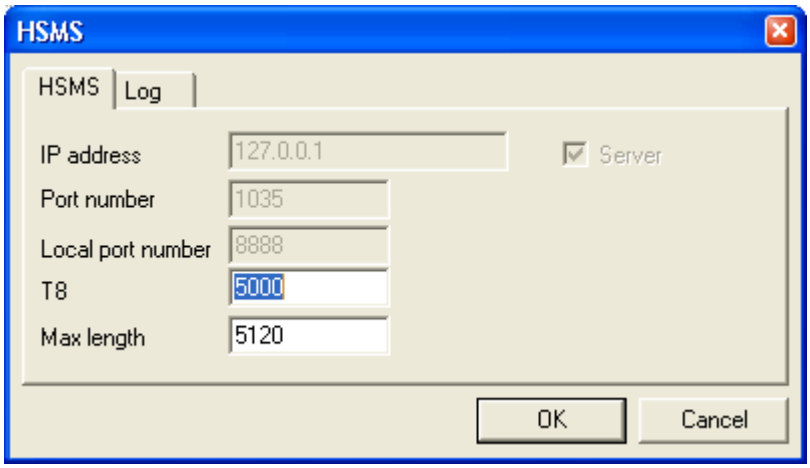
プロパティを編集するダイアログボックスを表示します。

```
❑ Visual Basic の場合
Function Config(pszTitle As String) As Boolean

❑ Visual C++の場合
BOOL Config(LPCTSTR pszTitle)

pszTitle          ダイアログボックスのキャプションタイトル。
```

ユーザがＯＫボタンをクリックしてダイアログボックスを閉じると IniFile プロパティ IniSection プロパティで指定されたファイルにプロパティを保存します。.ini ファイルに書き込んだ場合は True が返ります。.ini ファイルに保存された内容を取り出してプロパティにセットするには LoadIni メソッドを呼び出します。



ポートをオープン中（Active プロパティが True のとき）にこのメソッドを呼び出しても IPAddress プロパティと PortNumber プロパティ、LocalPortNumber プロパティの値を変更することはできません。

❑ Visual Basic の場合

```
.Config “HSMS Configuration”
```

❑ Visual C++の場合

```
m_hsms.Config(“HSMS Configuration”);
```

11.1.21 ConvertIPAddress

ホスト名をＩＰアドレスに変換します。

❑ Visual Basic の場合

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

❑ Visual C++の場合

```
CString ConvertIPAddress(LPCTSTR pszName)
```

PszName　　変換したいホスト名。

❑ Visual Basic の場合

```
.IPAddress = .ConvertIPAddress(“”)
```

❑ Visual C++の場合

```
m_hsms.SetIPAddress(m_hsms.ConvertIPAddress(“”));
```

11.1.22 Disconnect

指定された相手との接続を切断します。このメソッドはサーバ側で使用します。

❑Visual Basic の場合

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

❑Visual C++の場合

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

pszIPAddress	切断相手のＩＰアドレス。
lPortNumber	切断相手のポート番号。

❑Visual Basic の場合

```
  .Disconnect “”, 5000
```

❑Visual C++の場合

```
  m_hsms.Disconnect(“”,5000);
```

11.1.23 GetHostName

自分のコンピュータのホスト名を取得します。

❑Visual Basic の場合

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

❑Visual C++の場合

```
CString GetHostName()
```

❑Visual Basic の場合

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

❑Visual C++の場合

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

11.1.24 LoadIni

プロパティを IniFile プロパティ IniSection プロパティで指定されたファイルから読み込みます。読み込みができない場合はリソースに埋め込まれている値がセットされます。

このメソッドは Config メソッドで保存されたプロパティを取り出すためアプリケーションの起動時に呼び出すといいでしょう。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

11.1.25 Send

メッセージを送信します。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

pszMsg　　送信するメッセージ。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

11.1.26 Connected

このイベントは相手とのコネクションが成立したら発生します。サーバ側の場合、Active プロパティを True にセットしただけではコネクションは成立していません。クライアント側が接続してきてはじめてコネクションが成立したことになります。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

pszIPAddress	メッセージの発信元のＩＰアドレス。
lPortNumber	メッセージの発信元のポート番号。
bConnect	接続した場合は True、切断した場合は False がセットされます。

11.1.27 Errors

このイベントは通信エラーが起こった場合に発生します。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

pszIPAddress	メッセージの発信元のＩＰアドレス。
lPortNumber	メッセージの発信元のポート番号。
sError	エラーコード。エラーコードは以下のいずれかの値です。

❑ Swing の出すエラー

-2	設定した最大バッファサイズより長いメッセージを受信
-4	T8 タイムアウト
-7	T6 タイムアウト
-8	T7 タイムアウト

❑ WinSock の出すエラー

10093	WSANOTINITIALISED	ソケットの初期化がされていない
10050	WSAENETDOWN	ネットワークサブシステムのエラー
10048	WSAEADDRINUSE	ソケットのローカルアドレスが既に使用中
10014	WSAEFAULT	ユーザアドレスが正しくない（禁則文字など）
10036	WSAEINPROGRESS	現在サービスプロバイダが処理中
10049	WSAEADDRNOTAVAIL	リモートアドレスが正しくない
10047	WSAEAFNOSUPPORT	指定されたアドレスファミリーはこのソケットで使用できない
10061	WSAECONNREFUSED	接続拒否された

```
   10039  WSAEDESTADDRREQ      ？
   10022  WSAEINVAL            リスニングソケット
   10056  WSAEISCONN           既に接続されている
   10024  WSAEMFILE            ？
   10051  WSAENETUNREACH       ネットワークに到達できなかった
   10055  WSAENOBUFS           バッファが足りない
   10038  WSAENOTSOCK          ソケットでない
   10060  WSAETIMEDOUT         コネクションが成立する前にタイムアウトした
   10035  WSAEWOULDBLOCK       すぐに実行できない

pszInfo          追加情報。現在のところは未使用となっています。
```

11.1.28 Read

このイベントはメッセージを正常に受信したら発生します。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

```
pszIPAddress     メッセージの発信元のＩＰアドレス。
lPortNumber      メッセージの発信元のポート番号。
pszMsg           受信したメッセージ。
```

11.1.29 SelConnection

このイベントはマウスで画面上の接続の選択を変更したら発生します。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

```
pszIPAddress     メッセージの発信元のＩＰアドレス。
lPortNumber      メッセージの発信元のポート番号。
```

12 SwingComm

SwingComm は RS-232C の汎用通信 Active X Control です。このコントロールをアプリケーションに組み込むことにより非常に簡単にシリアルポートでの送受信が可能となります。同様な製品にマイクロソフトの COMM コントロールや PDQ Comm があります。

12.1 リファレンス

いくつかの用語がでてくるので簡単に説明します。

❑ 永続化プロパティ

デザイン時にこのプロパティを設定するとその値をリソースに保存します（永続化）。このため実行時にいちいちプロパティをセットする必要はありません。

12.1.1 Active

このプロパティに True をセットすると CommPort プロパティ、BaudRate プロパティ、ByteLength プロパティ、StopBits プロパティ、Parity プロパティで指定された条件でポートをオープンします。オープンできたかどうかを調べるには Active プロパティが True になっているかを調べます。False をセットするとポートをクローズします。アプリケーションを終了するときにポートをクローズし忘れても心配ありません。コントロールが自動的にクローズしてくれるからです。

❑ Visual Basic の場合

```
.CommPort = 0                          ‘Com1
.BaudRate = 9600                       ‘9600 bps
.ByteLength = 8                        ‘8 bit
.StopBits = 0                          ‘1 bit
.Parity = 0                            ‘なし
.Active = True                         ‘Open
If Not .Active then
  MsgBox “エラー：通信ポートをオープンできません！”
End If
```

❑ Visual C++の場合

```
m_comm.SetCommPort(0);                 // Com1
m_comm.SetBaudRate(9600);              // 9600 bps
m_comm.SetByteLength(8);               // 8 bit
m_comm.SetStopBits(0);                 // 1 bit
m_comm.SetParity(0);                   // なし
m_comm.SetActive(true);                // Open
if(m_comm.GetActive())
  MessageBox(“エラー：通信ポートをオープンできません！”);
```

12.1.2 BaudRate

ボーレートを指定します。ボーレートの値は数字で直接指定してやります。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

12.1.3 ByteLength

１文字のビット数を指定します。このプロパティは永続化プロパティです。

```
      7                           7 ビット
      8                           8 ビット
```

12.1.4 ByteStream

このプロパティに値をセットすると送信し、取り出す（ゲットする）と受信キューの内容を１バイトだけ読み出します。通常、読み出しの場合は Read イベントが発生した時点で Count プロパティの回数だけゲットします。

12.1.5 CommPort

通信ポートの番号を指定します。通信ポートの番号は 0 から始まります。つまり Com1 は 0、Com2 は 1 となります。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

12.1.6 Count

受信キューに入っている文字数です。このプロパティが 0 でないときは Read イベントが発生します。もし Read イベントでキューの内容を全て読み取らなかった場合は何度でも（0 になるまで）イベントが発生します。

12.1.7 IniFile

プロパティの内容を保存する.ini ファイル名です。フルパスで指定した場合はそのディレクトリに作成します。ディレクトリを指定せずファイル名のみの場合は Windows のディレクトリに作成します。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

拡張子は必ずしも ini である必要はありませんが、習慣上 ini にするのが普通です。

12.1.8 IniSection

プロパティの内容を保存する.ini ファイル中のセクション名です。１つのプロジェクトに複数の SwingComm コントロールが埋め込まれていても、別々のセクション名を指定することで設定内容を変えることができます。このプロパティは永続化プロパティです。

❑ Visual Basic の場合

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

❑ Visual C++の場合

```
m_comm.SetIniSection(“Host”);           // [Host] section
```

12.1.9 Parity

パリティを指定します。このプロパティは永続化プロパティです。

0	なし
1	奇数
2	偶数
3	マーク
4	スペース

12.1.10 StopBits

ストップビット長です。このプロパティは永続化プロパティです。

```
0                       1 ビット
1                       1.5 ビット
2                       2 ビット
```

12.1.11 Stream

文字列を送信します。ByteStream プロパティは 16 ビット符号つき整数ですがアプリケーションによっては文字列で扱った方が簡単な場合もあります。

12.1.12 Config

プロパティを編集するダイアログボックスを表示します。

❑ Visual Basic の場合

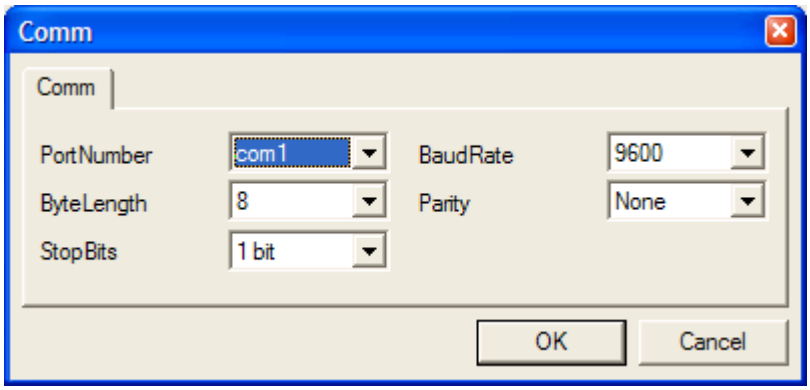
```
Function Config(pszTitle As String) As Boolean
```

❑ Visual C++の場合

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

pszTitle　　ダイアログボックスのキャプションタイトル。

ユーザがＯＫボタンをクリックしてダイアログボックスを閉じると IniFile プロパティと IniSection プロパティで指定されたファイルにプロパティを保存します。.ini ファイルに書き込んだ場合は True が返ります。.ini ファイルに保存された内容を取り出してプロパティにセットするには LoadIni メソッドを呼び出します。



ポートをオープン中（Active プロパティが True のとき）にこのメソッドを呼び出してもプロパティの値を変更することはできません。

❑Visual Basic の場合

```
  .Config “COMM Configuration”
```

❑Visual C++の場合

```
  m_comm.Config(“COMM Configuration”);
```

12.1.13 LoadIni

プロパティを IniFile プロパティと IniSection プロパティで指定されたファイルから読み込みます。読み込みができない場合はリソースに埋め込まれている値がセットされます。
このメソッドは Config メソッドで保存されたプロパティを取り出すためアプリケーションの起動時に呼び出すとよいでしょう。

❑Visual Basic の場合

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

❑Visual C++の場合

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

12.1.14 Read

このイベントは１文字でも受信したら発生します。アプリケーションはこのイベントが発生したら受信キューの内容を読み出さなければなりません。受信キューが空にならない限り何度でもこのイベントが発生します。Count プロパティの回数だけ ByteStream プロパティを読み出します。

❑ Visual Basic の場合

```
Sub Read()
```

❑ Visual C++の場合

```
void OnRead()
{
  char szBuf[128];
  for(int nCnt=0;nCnt<sizeof(szBuf) && xxx.GetCount();nCnt++)
    szBuf[nCnt]=(char)xxx.GetByteStream();
    ...
}
```

13 プログラミングのヒント

13.1 即値を使わない

即値を使うのではなく定数名で記述するようにすると可読性が向上します。コントロールには既に以下の定数が埋め込まれています。

❑ Visual Basic の場合

SwingSecsII の Type プロパティ用

```
  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
```

SwingSecsI の Errors イベント引数中の *sError* 用

```
  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
```

Visual C++の場合は埋め込まれた定数を直接利用できないため、プログラムのソースに定数名を記述しておくと便利です。

❑ Visual C++の場合

```
enum[2]
{
  SecsTypeInvalid=0,                    // “=0”は不要
```

[2] Visual C++ .NET では enum に名前を付けなくてもコンパイルエラーにはなりませんが、名無しになってしまいます。

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

13.2S1F1 を送信するとき

メッセージを作成する場合は基本的に List プロパティを使用して記述すると簡単です。またあとで読むことを考えても List プロパティを使うことをお薦めします。

❑ Visual Basic の場合

```
  .List = “s1f1w”
```

❑ Visual C++の場合

```
  m_msg.SetList(“s1f1w”);
```

個別に記述することも可能ですが以下のように可読性の悪いソースコードになります。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

13.3S1F13 を作るとき

List プロパティを使用すると以下のようになります。

❑ Visual Basic の場合

```
.List = _
 “s1f13w” + _
 “{” + _
 “   <a‘Swing’>” + _
 “   <a‘ver 2.71 Copyright© 1996-2001 JazzSoft’>” + _
 “}”
```

❑ Visual C++の場合

```
m_msg.SetList(
 “s1f13w”
 “{”
 “   <a‘Swing’>”
 “   <a‘ver 2.71 Copyright© 1996-2001 JazzSoft’>”
 “}”
 );
```

もちろん一行で書くこともできますが、可読性を考慮すると適度に区切ってインデントを入れた方が無難でしょう。なお、あまりお薦めはできませんが、以下のように記述することも可能です。

❑ Visual Basic の場合（Init, Stream, Fucntion, WBit, Msg については省略）

```
.Pointer = “”
.Add SecsTypeList, “”
.Pointer = “1”
.Add SecsTypeAscii, “Swing”
.Pointer = “2”
.Add SecsTypeAscii, “Ver 2.71”
```

❑ Visual C++の場合

```
m_msg.SetPointer(“”);
m_msg.Add(SecsTypeList,“”);
m_msg.SetPointer(“1”);
m_msg.Add(SecsTypeAscii,“Swing”);
m_msg.SetPointer(“2”);
m_msg.Add(SecsTypeAscii,“Ver 2.71”);
```

ノードを作成する場合にはまずノード指定してそれに対して Add を実行することになります。このため毎回２ステップの命令が必要となりますのでやや煩雑です。そこで子ノードの数字が実際の個数より多い場合は新規ノードとみなす機能が組み込まれています。よってこのソースは以下のように書くことができます。

❑ Visual Basic の場合

```
.Pointer = “”
.Add SecsTypeList, “”
.Pointer = “2”
.Add SecsTypeAscii, “Swing”
.Add SecsTypeAscii, “ver 2.71”
```

❑Visual C++の場合

```
m_msg.SetPointer("");
m_msg.Add(SecsTypeList,"");
m_msg.SetPointer("2");
m_msg.Add(SecsTypeAscii,"Swing");
m_msg.Add(SecsTypeAscii,"ver 2.71");
```

もしこれにもう一行追加したらどうなるでしょうか？この場合３回目の Add で指定されているノードは"2"のままです。

❑Visual Basic の場合

```
.Pointer = "2"
.Add SecsTypeAscii, "Swing"
.Add SecsTypeAscii, "ver 2.71"
.Add SecsTypeAscii, "Copyright© 1996-2001 JazzSoft"
```

❑Visual C++の場合

```
m_msg.SetPointer("2");
m_msg.Add(SecsTypeAscii,"Swing");
m_msg.Add(SecsTypeAscii,"ver 2.71");
m_msg.Add(SecsTypeAscii,"Copyright© 1996-2001 JazzSoft");
```

追加対象のノードが既に存在する場合は配列を作ります。アスキー文字列の場合はもともと配列ですから末尾に追加されます（Visual Basic の String 型、Visual C++の CString クラスと類似）。よってこのコードは以下のコードと同じです。

❑Visual Basic の場合

```
.Pointer = "2"
.Add SecsTypeAscii, "Swing"
.Add SecsTypeAscii, "ver 2.71 Copyright© 1996-2001 JazzSoft"
```

❑Visual C++の場合

```
m_msg.SetPointer("2");
m_msg.Add(SecsTypeAscii,"Swing");
m_msg.Add(SecsTypeAscii,"ver 2.71 Copyright© 1996-2001 JazzSoft");
```

13.4 ノードの要素が複数個あるとき

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

これもやはり List プロパティを使います。

❑ Visual Basic の場合

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

❑ Visual C++の場合

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

Add メソッドを使った方法も一応記述しておきますがお薦めはできません。

❑ Visual Basic の場合

```
.Pointer = “”
.Add SecsTypeList, “”
.Pointer = “99”
.Add SecsTypeBoolean, “1”
.Add SecsTypeList, “”
.Pointer = “2/99”
.Add SecsTypeWord, “201”
.Add SecsTypeWord, “92”
```

❑ Visual C++の場合

```
m_msg.SetPointer(“”);
m_msg.Add(SecsTypeList,“”);
m_msg.SetPointer(“99”);
m_msg.Add(SecsTypeBoolean,“1”);
m_msg.Add(SecsTypeList,“”);
m_msg.SetPointer(“2/99”);
m_msg.Add(SecsTypeWord,“201”);
m_msg.Add(SecsTypeWord,“92”);
```

となります。

13.5受信したメッセージを解析するとき

受信したメッセージを解析するにはノードを指定して直接、値を取り出します。

❑ Visual Basic の場合

```
.Pointer = “3”
If .Type <> SecsTypeAscii Then
  MsgBox “タイプが違います”
  Exit Sub
End If
If CInt(.Value) = 3 Then
  MsgBox “受け取った値は３だ”
End If
```

❑ Visual C++の場合

```
m_msg.SetPointer(“3”);
if(m_msg.GetType()!=SecsTypeAscii)
{
  MessageBox(“タイプが違います”);
  return;
}
if(::atoi(m_msg.GetValue())==3)
  MessageBox(“受け取った値は３だ”);
```

値は文字列なので型が不明な場合でも値を得ることが可能です。例えば S2F37 の内容の型が u4（符号なし 4 バイト整数）か u2（符号なし 2 バイト整数）か分からないといったことが考えられます。このようなケースでも型を無視して値を取得することができます。

❑ Visual Basic の場合

```
.Pointer = “3”
Dim nType As Integer
nType = .Type
If (nType <> SecsTypeDWord) And (nType <> SecsTypeWord) Then
  MsgBox “タイプが違います”
  Exit Sub
End If
If .Value = “3” Then
  MsgBox “受け取った値は 3 だ”
End If
```

❑ Visual C++の場合

```
m_msg.SetPointer(“3”);
int nType=m_msg.GetType();
if(nType!=SecsTypeDWord && nType!=SecsTypeWord)
{
  MessageBox(“タイプが違います”);
  return;
}
if(m_msg.GetValue()==“3”)
  MessageBox(“受け取った値は 3 だ”);
```

13.6イベントでのメッセージボックス

イベントハンドラ中でモーダルダイアログボックス（メッセージボックスも同様）を表示するのは非常に危険です。次のようなコードを考えてみましょう。

❑ Visual Basic の場合

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

もしメッセージボックスを閉じるのが遅れたら相手に返信するのも遅れてしまいます。このため T3 タイムアウトしてしまうこともあります。T3 タイムアウトすると S9F9 が送られてくるため再びイベントが発生します。そうなると再びイベントハンドラに突入してしまいます。

これを避けるにはモードレスダイアログボックスにするとよいでしょう。あるいはメッセージを表示する為のコントロールを画面に貼り付け、そこに表示するようにします。

❑ Visual Basic の場合

```
  ErrorMsg.Caption = “NG でした”
  ErrorMsg.Blink = True
```