# SavoySecsII ActiveX コントロール
# ユーザーガイド

*Jazz Soft, Inc.*

# 1 改訂履歴

| バージョン | 日付 | 氏名 | 説明 |
|---|---|---|---|
| 1.00 | 2009 年 7 月 31 日 | Hikaru Okada | 新規に作成。 |
| 1.00a | 2009 年 8 月 22 日 | Hikaru Okada | マニュアルのページ数が大きくなったため分割。 |
| 1.00b | 2009 年 12 月 22 日 | Hikaru Okada | HSMS プロパティをサポート。 |

# 2 目次

# 3 SavoySecsII

SavoySecsII コントロールは SEMI E5 (SECS-II)の機能を作成するための開発支援製品です。装置側ソフト、ホスト側ソフトのいずれの開発にも使えます。SavoyHsms コントロールや SavoySecsI コントロールと組み合わせて使います。

### プロパティ

| 名前 | Description |
|---|---|
| Appearance | 外観を決定する値を取得または設定します。 |
| Async | SML 処理を別スレッドで行うかどうかを取得または設定します。 |
| BlockNumber | ブロック番号を取得または設定します。 |
| BorderStyle | 境界線スタイルを取得または設定します。 |
| DeviceID | デバイス ID を取得または設定します。 |
| Ebit | エンドビットを取得または設定します。 |
| Error | SML 処理でエラーが発生したかどうかを取得します。 |
| ErrorDialog | SML 処理でエラーが発生したらダイアログボックスを表示するかどうかを取得または設定します。 |
| Function | ファンクション番号を取得または設定します。 |
| Host | ホストか装置かの役割を取得または設定します。 |
| HSMS | HSMS か SECS-I かを取得または設定します。 |
| Msg | メッセージデータを取得または設定します。 |
| Node | メッセージ内容を操作するためのノードを取得または設定します。 |
| NodeCount | サブアイテムの個数を取得します。 |
| NodeType | ノードタイプを取得します。 |
| NodeValue | ノードの値を取得します。 |
| NodeValueHex | ノードの値を 16 進法表現で取得します。 |
| PType | プレゼンテーションタイプを取得または設定します。 |
| Rbit | リバースビットを取得または設定します。 |
| SessionID | セッション ID を取得または設定します。 |
| SML | SECS-II メッセージを SML 文字列として取得または設定します。 |
| SourceID | ソース ID を取得または設定します。 |
| Stream | ストリーム番号を取得または設定します。 |
| SType | セッションタイプを取得または設定します。 |
| SuggestedReplyMsg | 適切だと考えられる応答メッセージを取得または設定します。 |
| SystemBytes | システムバイトを取得または設定します。 |
| TransactionID | トランザクション ID を取得または設定します。 |
| Wbit | ウェイトビットを取得または設定します。 |
| XML | SECS-II メッセージを XML 文字列として取得または設定します。 |

### メソッド

| 名前 | Description |
|---|---|
| AboutBox | バージョン情報を表示します。 |
| Reply | 返信メッセージとして SECS-II ヘッダを初期化します。 |
| Reset | 内部のデータ構造とパラメータを初期化します。 |
| Verify | 現在保持しているメッセージの内容を検証します。 |

### イベント

| 名前 | Description |
|---|---|
| Ready | SML 処理が完了したときに通知されます。 |

## 3.1 プロパティ

### 3.1.1 Appearance

SavoySecsII コントロールの外観を決定する値を取得または設定します。

| 値 | 説明 |
|---|---|
| 0 | フラット |
| 1 | 凹んだ枠線 |

**構文**

Visual Basic 6.0

```
Appearance As Integer
```

Visual C++ 6.0

```
short GetAppearance()
void SetAppearance(short)
```

**使用例**

Visual Basic 6.0

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

Visual C++ 6.0

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

**特記事項**

永続化プロパティ。

**参照**

### 3.1.2 Async

SML 文字列の処理を別スレッドで行うかどうかを取得または設定します。

| 値 | 説明 |
|---|---|
| False | 処理は別スレッドで行われません。 |
| True | 処理を別スレッドで行い、処理完了時に Ready イベントで通知します。 |

**構文**

Visual Basic 6.0

```
Async As Boolean
```

Visual C++ 6.0

```
BOOL GetAsync()
void SetAsync (BOOL)
```

**使用例**

Visual Basic 6.0

```
.Async = False
```

Visual C++ 6.0

```
m_ctrl.SetAsync(false);
```

**特記事項**

永続化プロパティ。

このプロパティは現時点では未使用です。

**参照**

### 3.1.3 BlockNumber

ブロック番号を取得または設定します。SECS-I でのみ使用されます。

SECS-I では下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | R デバイス ID |
| 2 | |
| 3 | W ストリーム番号 |
| 4 | ファンクション番号 |
| 5 | E ブロック番号 |
| 6 | |
| 7 | ソース ID |
| 8 | |
| 9 | トランザクション ID |
| 10 | |

**構文**

Visual Basic 6.0

```
BlockNumber As Long
```

Visual C++ 6.0

```
long GetBlockNumber()
void SetBlockNumber(long)
```

**使用例**

Visual Basic 6.0

```
Dim lBlock As Long
lBlock = .BlockNumber
```

Visual C++ 6.0

```
long lBlock = m_ctrl.GetBlockNumber();
```

**特記事項**

SECS-I の受信メッセージで BlockNumber が 1 以外の場合は、マルチブロックメッセージだったことを示します。

送信時は必ず 1 にセットします。もしメッセージのサイズが大きくて 1 ブロックに収まりきらない場合は、自動的にマルチブロックにして送信されます。

**参照**

### 3.1.4 BorderStyle

SavoySecsII コントロールの境界線スタイルを取得または設定します。

| 値 | 説明 |
|---|---|
| 0 | 境界線なし |
| 1 | 境界線あり |

**構文**

Visual Basic 6.0

```
BorderStyle As Integer
```

Visual C++ 6.0

```
short GetBorderStyle()
void SetBorderStyle(short)
```

**使用例**

Visual Basic 6.0

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

Visual C++ 6.0

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

**特記事項**

永続化プロパティ。

**参照**

### 3.1.5 DeviceID

デバイス ID を取得または設定します。デバイス ID は SECS-II ヘッダの先頭 2 ビット目から 15 ビットです。

SECS-I では下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | R デバイス ID |
| 2 | |
| 3 | W ストリーム番号 |
| 4 | ファンクション番号 |
| 5 | E ブロック番号 |
| 6 | |
| 7 | ソース ID |
| 8 | |
| 9 | トランザクション ID |
| 10 | |

**構文**

Visual Basic 6.0

```
DeviceID As Long
```

Visual C++ 6.0

```
long GetDeviceID()
void SetDeviceID(long)
```

**使用例**

Visual Basic 6.0

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

Visual C++ 6.0

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

**特記事項**

永続化プロパティ。Reset メソッドを呼び出すとこの値で初期化されます。

デバイス ID とセッション ID はほぼ同じですが、デバイス ID は 15 ビット、セッション ID は 16 ビットです。

**参照**

### 3.1.6 Ebit

エンドビットを取得または設定します。SECS-I でのみ使用されます。

| 値 | 説明 |
|---|---|
| False | 途中のブロック |
| True | 最終のブロック |

SECS-I では下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | R デバイス ID |
| 2 | |
| 3 | W ストリーム番号 |
| 4 | ファンクション番号 |
| 5 | E ブロック番号 |
| 6 | |
| 7 | ソース ID |
| 8 | |
| 9 | トランザクション ID |
| 10 | |

**構文**

Visual Basic 6.0

```
Ebit As Boolean
```

Visual C++ 6.0

```
BOOL GetEbit()
void SetEbit(BOOL)
```

**使用例**

Visual Basic 6.0

```
If .Ebit = False Then
   ' Never comes here
End If
```

Visual C++ 6.0

```
if(!m_ctrl.GetEbit())
{
   // Never comes here
}
```

**特記事項**

SECS-I の受信メッセージのエンドビットは常に True となります。これは SavoySecsI コントロールが最終ブロックまで受信してから Received イベントを発生させるからです。

**参照**

### 3.1.7 Error

SML 文字列の処理でエラーが発生したかどうかを取得します。

| 値 | 説明 |
|---|---|
| False | エラーは発生していません。 |
| True | SML の処理でエラーが発生しました。 |

**構文**

Visual Basic 6.0

```
Error As Boolean
```

Visual C++ 6.0

```
BOOL GetError()
void SetError(BOOL)
```

**使用例**

Visual Basic 6.0

```
.SML = Text1.Text
If .Error Then
    ...
```

Visual C++ 6.0

```
m_ctrl.SetSml(m_strText1);
if(m_ctrl.GetError())
    ...
```

**特記事項**

読み出し専用プロパティ。

**参照**

### 3.1.8 ErrorDialog

SML 文字列の処理中にエラーが発生した際に、ダイアログボックスを表示するかどうかを取得または設定します。

| 値 | 説明 |
|---|---|
| False | ダイアログボックスを表示しません。 |
| True | ダイアログボックスを表示します。 |

**構文**

Visual Basic 6.0

```
ErrorDialog As Boolean
```

Visual C++ 6.0

```
BOOL GetErrorDialog()
void SetErrorDialog(BOOL)
```

**使用例**

Visual Basic 6.0

```
.ErrorDialog = False
```

Visual C++ 6.0

```
m_ctrl.SetErrorDialog(false);
```

**特記事項**

永続化プロパティ。

**参照**

### 3.1.9 Function

ファンクション番号を取得または設定します。

SECS-I では下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | R デバイス ID |
| 2 | |
| 3 | W ストリーム番号 |
| 4 | ファンクション番号 |
| 5 | E ブロック番号 |
| 6 | |
| 7 | ソース ID |
| 8 | |
| 9 | トランザクション ID |
| 10 | |

HSMS データメッセージでは下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | セッション ID |
| 2 | |
| 3 | W ストリーム番号 |
| 4 | ファンクション番号 |
| 5 | P タイプ |
| 6 | S タイプ |
| 7 | システムバイト |
| 8 | |
| 9 | |
| 10 | |

**構文**

Visual Basic 6.0

```
Function As Integer
```

Visual C++ 6.0

```
short GetFunction()
void SetFunction(short)
```

**使用例**

Visual Basic 6.0

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

Visual C++ 6.0

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

**特記事項**

**参照**

### 3.1.10 Host

SavoySecsII コントロールの役割を取得または設定します。このプロパティは Verify()メソッドを使ってメッセージの構造を検証する場合と、検証の結果として SuggestedReplyMsg プロパティにセットされるメッセージにのみ影響します。

| 値 | 説明 |
|---|---|
| False | 装置 |
| True | ホスト |

**構文**

Visual Basic 6.0

```
Host As Boolean
```

Visual C++ 6.0

```
BOOL GetHost()
void SetHost(BOOL)
```

**使用例**

Visual Basic 6.0

```
.Host = False
```

Visual C++ 6.0

```
m_ctrl.SetHost(false);
```

**特記事項**

永続化プロパティ。

**参照**

### 3.1.11 HSMS

HSMS か SECS-I かを取得または設定します。デフォルトは HSMS です。

| 値 | 説明 |
|---|---|
| False | SECS-I |
| True | HSMS |

#### 構文

Visual Basic 6.0

```
HSMS As Boolean
```

Visual C++ 6.0

```
BOOL GetHsms()
void SetHsms(BOOL)
```

#### 使用例

Visual Basic 6.0

```
.HSMS = true
```

Visual C++ 6.0

```
m_ctrl.SetHsms(true);
```

#### 特記事項

永続化プロパティ。

#### 参照

### 3.1.12 Msg

メッセージデータを取得または設定します。メッセージデータ形式は 16 進法の ASCII 文字列です。

#### 構文

Visual Basic 6.0

```
Msg As String
```

Visual C++ 6.0

```
CString GetMsg()
void SetMsg(LPCTSTR)
```

#### 使用例

Visual Basic 6.0

```
Private Sub SavoySecsI1_Read (ByVal pszMsg As String)
With SavoySecsII1
    .Msg = pszMsg
    Select Case .Stream
    Case 1
        Select Case .Fucntion
        Case 1
            's1f1
            ...
```

Visual C++ 6.0

```
void Cxxx::OnxxxRead(LPCTSTR pszMsg)
{
    m_ctrl.SetMsg(pszMsg);
    switch(m_ctrl.GetStream())
    {
    case 1:
        switch(m_ctrl.GetFucntion())
        {
        case 1:
            // s1f1
            ...
```

#### 特記事項

#### 参照

### 3.1.13 Node

メッセージ内容を操作するためのノードを取得または設定します。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。

#### 構文

Visual Basic 6.0

```
Node As String
```

Visual C++ 6.0

```
CString GetNode()
void SetNode(LPCTSTR)
```

#### 使用例

次のような SML 構造のメッセージを作ってみます。

```
s1f13w
{
    <a'Savoy'>
    <a'1'>
}
```

既にSavoySecsIIコントロールにメッセージ構造が入っているかもしれません。まず最初に全体を書き換える必要があるため、ルートノードを指定します。

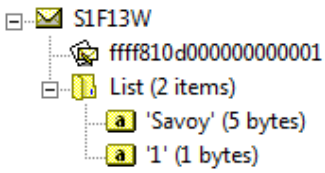
Visual Basic 6.0

```
.Node = ""
.SML = "s1f13w{<a'Savoy'><a'1'>}"
```

Visual C++ 6.0

```
m_ctrl.SetNode("");
m_ctrl.SetSml("s1f13w{<a'Savoy'><a'1'>}");
```

このコードを実行すると以下のような構造のメッセージが作成されます。



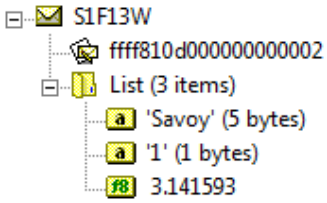

この構造に 3 番目のノードを追加したい場合は、Node に 3 を指定します。

Visual Basic 6.0

```
.Node = "3"
.SML = "<f8 3.1415926535>"
```

Visual C++ 6.0

```
m_ctrl.SetNode("3");
m_ctrl.SetSml("<f8 3.1415926535>");
```



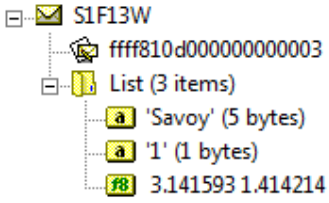

3 番目のノードを配列にしたい場合は、同じ型で数値を指定します。

Visual Basic 6.0

```
.Node = "3"
.SML = "<f8 141421356>"
```

Visual C++ 6.0

```
m_ctrl.SetNode("3");
m_ctrl.SetSml("<f8 141421356>");
```




ここで 3 番目のノードの値を NodeValue プロパティを使って読み出すと、配列の要素はスペース文字で分離され、”3.141593 1.414214”という文字列が返ります。配列の個々の要素を取り出したい場合は、以下のように [ ] を使ってインデックス指定します。インデックスは C/C++/Java/C#言語のように、0 から始まります。

Visual Basic 6.0

```
.Node = "3[0]"
.Node = "3[1]"
```

Visual C++ 6.0

```
m_ctrl.SetNode("3[0]");
m_ctrl.SetNode("3[1]");
```

上記”3[0]”では、”3.141593”という文字列が返り、”3[1]”では”1.414214”という文字列が返ります。

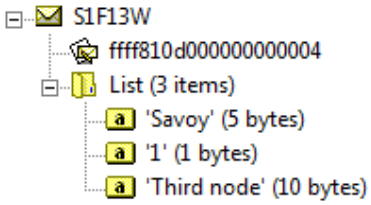
もし型の異なるノードに変更したい場合は、違う型で指定します。

Visual Basic 6.0

```
.Node = "3"
.SML = "<a’Third node’>"
```

Visual C++ 6.0

```
m_ctrl.SetNode("3");
m_ctrl.SetSml("<a'Third node'>");
```



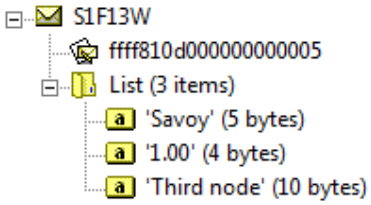

文字列を連結したい場合は、同じ型で文字列を指定します。

Visual Basic 6.0

```
.Node = "2"
.SML = "<a'.00'>"
```

Visual C++ 6.0

```
m_ctrl.SetNode("2");
m_ctrl.SetSml("<a'.00'>");
```



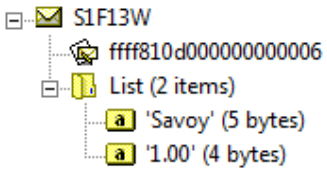

SML に空文字を指定すると、そのノードを消去します。

Visual Basic 6.0

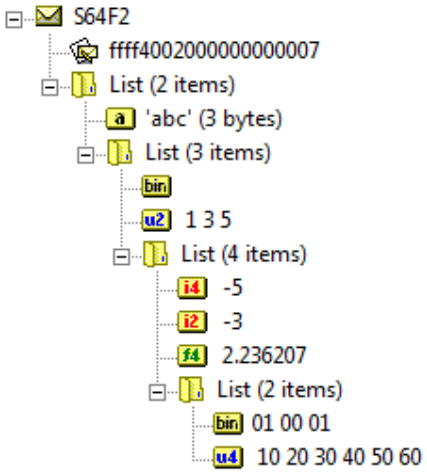
```
.Node = "3"
.SML = ""
```

Visual C++ 6.0

```
m_ctrl.SetNode("3");
m_ctrl.SetSml("");
```




複雑な構造のメッセージでも Node プロパティを使えば、ピンポイントで値を読み出すことができます。




このメッセージの最後に u4 型の 6 個の配列があります。この 4 番目の値を読み出してみましょう。それにはノードの位置を特定する必要があります。最初から見ていくと、リストの 2 番目、その中の 3 番目、その中の 4 番目、その中の 2 番目、配列の 4 番目だということが分かります。

Visual Basic 6.0

```
.Node = "2/3/4/2[3]"
```

Visual C++ 6.0

```
m_ctrl.SetNode("2/3/4/2[3]");
```

この例ではノードに”2/3/4/2[3]”を指定しています。NodeValue プロパティを読み出すと”40”が戻ります。

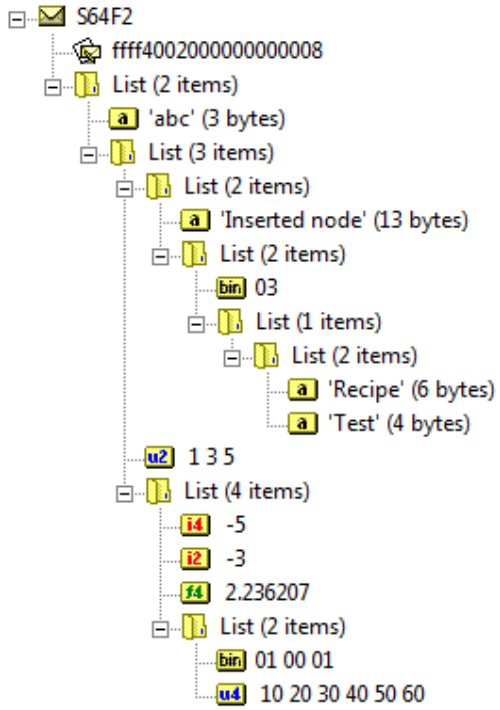
複雑な構造の SML をノードに指定することもできます。

Visual Basic 6.0

```
.Node = "2/1"
.SML = " {<a'Inserted node'>{<b 3>{{<a'Recipe'><a'Test'>}}}}"
```

Visual C++ 6.0

```
m_ctrl.SetNode("2/1");
m_ctrl.SetSml("{<a'Inserted node'>{<b 3>{{<a'Recipe'><a'Test'>}}}}");
```




**特記事項**

ノードは Windows のフォルダ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『フォルダ』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。メッセージボディ全体を書き換える場合は、ルートノードを指定します。

**参照**

### 3.1.14 NodeCount

サブアイテムの個数を取得します。もしノードプロパティがリスト型の場合、このプロパティはサブノードの数を意味します。それ以外は配列の個数です。

#### 構文

Visual Basic 6.0

```
NodeCount As Long
```

Visual C++ 6.0

```
long GetNodeCount()
```

#### 使用例

Visual Basic 6.0

```
.Node = ""
.SML = “{{<b 1>}}”
.Node = "99"
Text1.Text = "NodeCount = " + Format$(.NodeCount)
```

Visual C++ 6.0

```
m_ctrl.SetNode("");
m_ctrl.SetSml(“{{<b 1>}}”);
m_ctrl.SetNode("99");
m_text1.Format("NodeCount = %d",m_ctrl.GetNodeCount());
```

#### 特記事項

読み出し専用プロパティ。

#### 参照

### 3.1.15 NodeType

ノードタイプを取得します。

| 値 | 列挙型 | 説明 |
|---|---|---|
| 1 | SecsTypeList | リスト |
| 2 | SecsTypeBinary | バイナリ |
| 3 | SecsTypeBoolean | ブーリアン |
| 4 | SecsTypeAscii | ASCII 文字列 |
| 5 | SecsTypeJis | JIS 8 文字列 |
| 6 | SecsTypeLong8 | 8 バイト符号付き整数 |
| 7 | SecsTypeChar | 1 バイト符号付き整数 |
| 8 | SecsTypeShort | 2 バイト符号付き整数 |
| 9 | SecsTypeLong | 4 バイト符号付き整数 |
| 10 | SecsTypeDouble | 8 バイト浮動小数点数 |
| 11 | SecsTypeFloat | 4 バイト浮動小数点数 |
| 12 | SecsTypeDWord8 | 8 バイト符号なし整数 |
| 13 | SecsTypeByte | 1 バイト符号なし整数 |
| 14 | SecsTypeWord | 2 バイト符号なし整数 |
| 15 | SecsTypeDWord | 4 バイト符号なし整数 |
| 16 | SecsTypeAscii2 | 2 バイト ASCII 文字列 |

**構文**

Visual Basic 6.0

```
NodeType As Integer
```

Visual C++ 6.0

```
short GetNodeType()
```

**使用例**

Visual Basic 6.0

```
.Node = "1/2"
Text1.Text = "NodeType = " + Format$(.NodeType)
```

Visual C++ 6.0

```
m_ctrl.SetNode("1/2");
m_text1.Format("NodeType = %d",m_ctrl.GetNodeType());
```

**特記事項**

読み出し専用プロパティ。

**参照**

### 3.1.16 NodeValue

ノードの値を取得します。もしノードが数値型の場合、値は 10 進法文字列に変換されます。

#### 構文

Visual Basic 6.0

```
NodeValue As String
```

Visual C++ 6.0

```
CString GetNodeValue()
```

#### 使用例

Visual Basic 6.0

```
If CInt(.NodeValue) = 201 Then
   Text1.Text = "CEID is 201"
End If
```

Visual C++ 6.0

```
if(::atoi(m_ctrl.GetNodeValue())==201)
   m_text1 = "CEID is 201";
```

#### 特記事項

読み出し専用プロパティ。

#### 参照

### 3.1.17 NodeValueHex

ノードの値を 16 進法表現で取得します。

#### 構文

Visual Basic 6.0

```
NodeValueHex As String
```

Visual C++ 6.0

```
CString GetNodeValueHex()
```

#### 使用例

Visual Basic 6.0

```
If .NodeValueHex = "ff" Then
   Text1.Text = "Value is 0xff"
End If
```

Visual C++ 6.0

```
if(m_ctrl.GetNodeValueHex()=="ff")
   m_text1="Value is 0xff";
```

#### 特記事項

読み出し専用プロパティ。

#### 参照

### 3.1.18 PType

プレゼンテーションタイプを取得または設定します。SECS-II メッセージを使いますので、0 を指定します。

HSMS データメッセージでは下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | セッション ID |
| 2 | |
| 3 | W ストリーム番号 |
| 4 | ファンクション番号 |
| 5 | P タイプ |
| 6 | S タイプ |
| 7 | システムバイト |
| 8 | |
| 9 | |
| 10 | |

HSMS コントロールメッセージでは下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | セッション ID |
| 2 | |
| 3 | |
| 4 | |
| 5 | P タイプ |
| 6 | S タイプ |
| 7 | システムバイト |
| 8 | |
| 9 | |
| 10 | |

**構文**

Visual Basic 6.0

```
PType As Integer
```

Visual C++ 6.0

```
short GetPType()
void SetPType(short)
```

**使用例**

Visual Basic 6.0

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

Visual C++ 6.0

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

**特記事項**

現在は SECS-II のみが規定されているため、PType プロパティは常に 0 です。

**参照**

### 3.1.19 Rbit

リバースビットを取得または設定します。

| 値 | 説明 |
|---|---|
| False | ホストから装置 |
| True | 装置からホスト |

SECS-I では下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | R デバイス ID |
| 2 | |
| 3 | W ストリーム番号 |
| 4 | ファンクション番号 |
| 5 | E ブロック番号 |
| 6 | |
| 7 | ソース ID |
| 8 | |
| 9 | トランザクション ID |
| 10 | |

**構文**

Visual Basic 6.0

```
Rbit As Boolean
```

Visual C++ 6.0

```
BOOL GetRbit()
void SetRbit(BOOL)
```

**使用例**

Visual Basic 6.0

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

Visual C++ 6.0

```
if(m_ctrl.GetRbit())
   MessageBox("Invalid reverse-bit!");
```

**特記事項**

**参照**

### 3.1.20 SessionID

HSMS のセッション ID を取得または設定します。セッション ID は SECS-II ヘッダの先頭 16 ビットです。

HSMS データメッセージでは下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | セッション ID |
| 2 | |
| 3 | W ストリーム番号 |
| 4 | ファンクション番号 |
| 5 | P タイプ |
| 6 | S タイプ |
| 7 | システムバイト |
| 8 | |
| 9 | |
| 10 | |

HSMS コントロールメッセージでは下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | セッション ID |
| 2 | |
| 3 | |
| 4 | |
| 5 | P タイプ |
| 6 | S タイプ |
| 7 | システムバイト |
| 8 | |
| 9 | |
| 10 | |

**構文**

Visual Basic 6.0

```
SessionID As Long
```

Visual C++ 6.0

```
long GetSessionID()
void SetSessionID(long)
```

**使用例**

Visual Basic 6.0

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

Visual C++ 6.0

```
if(m_ctrl.GetSessionID()!=0xffff)
   MessageBox("Invalid Session ID!");
```

**特記事項**

**参照**

### 3.1.21 SML

SECS-II メッセージを SML 文字列として取得または設定します。SML 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の SML 文字列で取得します。また SML プロパティに SML 文字列をセットする場合、文字列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（無視されます）。

#### 構文

Visual Basic 6.0

```
SML As String
```

Visual C++ 6.0

```
CString GetSml()
void SetSml(LPCTSTR)
```

#### 使用例

Visual Basic 6.0

```
.SML = "s1f13w{<a'Savoy'><a'1.00'>}"
```

Visual C++ 6.0

```
m_ctrl.SetSml("s1f13w{<a'Savoy'><a'1.00'>}");
```

#### 特記事項

SML 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。

説明文中の青色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


| 項目 | 説明 |
|---|---|

| | |
|---|---|
| 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。

**リスト**

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

| 項目 | 説明 |
|---|---|
| NumOfItem | 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。 |

**アスキー文字列**

```
<a [Strings]>
```

| 項目 | 説明 |
|---|---|
| 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。

**2 バイト文字列**

```
<a2 [Strings]>
```

| 項目 | 説明 |
|---|---|

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

**JIS8 文字列**

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

アスキー型と同じように扱われます。

| 項目 | 説明 |
|---|---|
| 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。

**整数**

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

| 項目 | 説明 |
|---|---|
| Numbers | 数値です。それぞれ以下の意味となります。 |

| 型 | 説明 |
|---|---|
| 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。

### 浮動小数点数

**<f4** [*FNumbers*]**>**
**<f8** [*FNumbers*]**>**

| 項目 | 説明 |
|---|---|
| FNumbers | 浮動小数点数です。それぞれ以下の意味となります。 |

| 型 | 説明 |
|---|---|
| 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。

### バイナリ

**<b** [*Numbers*]**>**

| 項目 | 説明 |
|---|---|
| Numbers | 数値です。 |

例えば

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

のように記述します。

### ブーリアン

**<bool [*Numbers*]>**
**<boolean [*Numbers*]>**

| 項目 | 説明 |
|---|---|
| Numbers | 数値です。 |

例えば

```
<bool true false 1 0>
```

のように記述します。

**参照**

### 3.1.22 SourceID

ソース ID を取得または設定します。

SECS-I では下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | R デバイス ID |
| 2 | |
| 3 | W ストリーム番号 |
| 4 | ファンクション番号 |
| 5 | E ブロック番号 |
| 6 | |
| 7 | ソース ID |
| 8 | |
| 9 | トランザクション ID |
| 10 | |

**構文**

Visual Basic 6.0

```
SourceID As Long
```

Visual C++ 6.0

```
long GetSourceID()
void SetSourceID(long)
```

**使用例**

Visual Basic 6.0

```
ctrl1.SourceID = ctrl2.SourceID
```

Visual C++ 6.0

```
m_ctrl1.SetSourceID(m_ctrl2.GetSourceID());
```

**特記事項**

**参照**

### 3.1.23 Stream

ストリーム番号を取得または設定します。

SECS-I では下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | R デバイス ID |
| 2 | |
| 3 | W ストリーム番号 |
| 4 | ファンクション番号 |
| 5 | E ブロック番号 |
| 6 | |
| 7 | ソース ID |
| 8 | |
| 9 | トランザクション ID |
| 10 | |

HSMS データメッセージでは下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | セッション ID |
| 2 | |
| 3 | W ストリーム番号 |
| 4 | ファンクション番号 |
| 5 | P タイプ |
| 6 | S タイプ |
| 7 | システムバイト |
| 8 | |
| 9 | |
| 10 | |

**構文**

Visual Basic 6.0

```
Stream As Integer
```

Visual C++ 6.0

```
short GetStream()
void SetStream(short)
```

**使用例**

Visual Basic 6.0

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

Visual C++ 6.0

```
switch(m_ctrl.GetStream())
{
```

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

**特記事項**

**參照**

### 3.1.24 SType

セッションタイプを取得または設定します。

| 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 | |

HSMS データメッセージでは下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | セッション ID |
| 2 | |
| 3 | W ストリーム番号 |
| 4 | ファンクション番号 |
| 5 | P タイプ |
| 6 | S タイプ |
| 7 | システムバイト |
| 8 | |
| 9 | |
| 10 | |

HSMS コントロールメッセージでは下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | セッション ID |
| 2 | |
| 3 | |
| 4 | |
| 5 | P タイプ |
| 6 | S タイプ |
| 7 | システムバイト |
| 8 | |
| 9 | |
| 10 | |

**構文**

Visual Basic 6.0

```
SType As Integer
```

Visual C++ 6.0

```
short GetSType()
void SetSType(short)
```

**使用例**

Visual Basic 6.0

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

Visual C++ 6.0

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

**特記事項**

**参照**

### 3.1.25 SuggestedReplyMsg

メッセージ構造を検証した結果、もっとも適切だと考えられる応答メッセージを取得または設定します。

#### 構文

Visual Basic 6.0

```
SuggestedReplyMsg As String
```

Visual C++ 6.0

```
CString GetSuggestedReplyMsg()
```

#### 使用例

Visual Basic 6.0

```
SavoySecsII1.Msg = SavoySecsII2.SuggestedReplyMsg
```

Visual C++ 6.0

```
m_send.SetMsg(m_receive.GetSuggestedReplyMsg());
```

#### 特記事項

#### 参照

### 3.1.26 SystemBytes

システムバイトを取得または設定します。

SECS-I では下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | R デバイス ID |
| 2 | |
| 3 | W ストリーム番号 |
| 4 | ファンクション番号 |
| 5 | E ブロック番号 |
| 6 | |
| 7 | ソース ID |
| 8 | |
| 9 | トランザクション ID |
| 10 | |

HSMS データメッセージでは下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | セッション ID |
| 2 | |
| 3 | W ストリーム番号 |
| 4 | ファンクション番号 |
| 5 | P タイプ |
| 6 | S タイプ |
| 7 | システムバイト |
| 8 | |
| 9 | |
| 10 | |

HSMS コントロールメッセージでは下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | セッション ID |
| 2 | |
| 3 | |
| 4 | |
| 5 | P タイプ |
| 6 | S タイプ |
| 7 | システムバイト |
| 8 | |
| 9 | |
| 10 | |

**構文**

Visual Basic 6.0

```
SystemBytes As Long
```

Visual C++ 6.0

```
long GetSystemBytes()
void SetSystemBytes(long)
```

**使用例**

Visual Basic 6.0

```
ctrl1.SystemBytes = ctrl2.SystemBytes
```

Visual C++ 6.0

```
m_ctrl1.SetSystemBytes(m_ctrl2.GetSystemBytes());
```

**特記事項**

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。

**参照**

### 3.1.27 TransactionID

トランザクション ID を取得または設定します。

SECS-I では下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | R デバイス ID |
| 2 | |
| 3 | W ストリーム番号 |
| 4 | ファンクション番号 |
| 5 | E ブロック番号 |
| 6 | |
| 7 | ソース ID |
| 8 | |
| 9 | トランザクション ID |
| 10 | |

**構文**

Visual Basic 6.0

```
TransactionID As Long
```

Visual C++ 6.0

```
long GetTransactionID()
void SetTransactionID(long)
```

**使用例**

Visual Basic 6.0

```
ctrl1.TransactionID = ctrl2.TransactionID
```

Visual C++ 6.0

```
m_ctrl1.SetTransactionID(m_ctrl2.GetTransactionID());
```

**特記事項**

**参照**

### 3.1.28 Wbit

ウェイトビットを取得または設定します。

| 値 | 説明 |
|---|---|
| False | 返信なし |
| True | 返信あり |

SECS-I では下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | R デバイス ID |
| 2 | |
| 3 | W ストリーム番号 |
| 4 | ファンクション番号 |
| 5 | E ブロック番号 |
| 6 | |
| 7 | ソース ID |
| 8 | |
| 9 | トランザクション ID |
| 10 | |

HSMS データメッセージでは下記のヘッダ構造が使用されます。

| Byte | 説明 |
|---|---|
| 1 | セッション ID |
| 2 | |
| 3 | W ストリーム番号 |
| 4 | ファンクション番号 |
| 5 | P タイプ |
| 6 | S タイプ |
| 7 | システムバイト |
| 8 | |
| 9 | |
| 10 | |

**構文**

Visual Basic 6.0

```
Wbit As Boolean
```

Visual C++ 6.0

```
BOOL GetWbit()
void SetWbit(BOOL)
```

**使用例**

Visual Basic 6.0

```
If (.Fucntion Mod 2) And .Wbit Then
   ' Send default reply message
   ...
End If
```

Visual C++ 6.0

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

**特記事項**

返信要求がある場合は True となります。

**参照**

### 3.1.29 XML

SECS-II メッセージを XML 文字列として取得または設定します。

#### 構文

Visual Basic 6.0

```
XML As String
```

Visual C++ 6.0

```
CString GetXml()
void SetXml(LPCTSTR)
```

#### 使用例

Visual Basic 6.0

```
Text1.Text = .XML
```

Visual C++ 6.0

```
m_text1.Format("%s",(LPCTSTR)m_ctrl.GetSml());
```

#### 特記事項

このプロパティは現時点では未使用です。

#### 参照

## 3.2 メソッド

### 3.2.1 AboutBox

バージョン情報を表示します。

**構文**

Visual Basic 6.0

```
Sub AboutBox()
```

Visual C++ 6.0

```
void AboutBox()
```

**戻り値**

ありません。

**使用例**

Visual Basic 6.0

```
.AboutBox
```

Visual C++ 6.0

```
m_hsms.AboutBox();
```

**特記事項**

**参照**

### 3.2.2 Reply

指定されたメッセージの返信メッセージとして SECS-II ヘッダを初期化します。もし指定されたメッセージが HSMS コントロールメッセージの場合、SavoySecsII コントロールはメッセージボディを消去します。その他の場合はメッセージボディは影響を受けません。

**構文**

Visual Basic 6.0

```
Sub Reply(lpszMsgHeader As String)
```

Visual C++ 6.0

```
void Reply(LPCTSTR lpszMsgHeader)
```

| 引数 | 説明 |
|---|---|
| lpszMsgHeader | 一次メッセージの Msg プロパティの値 |

**戻り値**

ありません。

**使用例**

Visual Basic 6.0

```
.SML = "<b 0>"
.Reply pszMsg
SavoySecsI1.Send .Msg
```

Visual C++ 6.0

```
m_ctrl.SetSml("<b 0>");
m_ctrl.Reply(pszMsg);
m_secs.Send(m_ctrl.GetMsg());
```

**特記事項**

**参照**

### 3.2.3 Reset

内部のデータ構造とパラメータを初期化します。

#### 構文

Visual Basic 6.0

```
Sub Reset()
```

Visual C++ 6.0

```
void Reset()
```

#### 戻り値

ありません。

#### 使用例

Visual Basic 6.0

```
.Reset
.Stream = 1
.Fucntion = 13
.Wbit = True
```

Visual C++ 6.0

```
m_ctrl.Reset();
m_ctrl.SetStream(1);
m_ctrl.SetFucntion(13);
m_ctrl.SetWbit(true);
```

#### 特記事項

#### 参照

### 3.2.4 Verify

現在保持しているメッセージの内容を検証します。

#### 構文

Visual Basic 6.0

```
Verify As Integer
```

Visual C++ 6.0

```
short Verify()
```

#### 戻り値

検証結果が返ります。

| 値 | 列挙型 | 説明 |
|---|---|---|
| 0 | VerificationCorrect | 問題なし。 |
| 1 | VerificationUserDefined | ユーザ定義メッセージ。 |
| 2 | VerificationIncorrect | メッセージ内容に問題あり。 |
| 3 | VerificationIncorrectAndReply | メッセージ内容に問題があり、返信する必要がある。 |
| 4 | VerificationNoWBit | ウェイトビットが必要なメッセージなのに、実際にはない。 |
| 5 | VerificationWBit | ウェイトビットが不要なメッセージなのに、実際にはある。 |
| 6 | VerificationWrongDirection | メッセージの方向が間違っている。 |
| 7 | VerificationUnrecognizedStream | 認識できないストリーム。 |
| 8 | VerificationUnrecognizedFunction | 認識できないファンクション。 |

#### 使用例

Visual Basic 6.0

```
Dim nResult As Integer
nResult = .Verify()
```

Visual C++ 6.0

```
Int nResult = m_ctrl.Verify();
```

#### 特記事項

検証したメッセージが一次メッセージの場合、SuggestedReplyMsg に推奨される二次メッセージがセットされます。

#### 参照

## 3.3 イベント

### 3.3.1 Ready

SML 文字列の処理を別スレッドで行う際に、処理が完了したときに通知されます。

**構文**

Visual Basic 6.0

```
Event Ready()
```

Visual C++ 6.0

```
void OnReady()
```

**使用例**

Visual Basic 6.0

```
Text1.Text = “SML string has been processed”
```

Visual C++ 6.0

```
TRACE(“SML string has been processed”);
```

**特記事項**

このイベントは現時点では未使用です。

**参照**