Dixie Gem/Hsms/Secs シミュレータ
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 8 月 17 日	Hikaru Okada	新規に作成。
1.01	2014 年 7 月 13 日	Carl Hikaru Okada	コンパイラを Visual Studio 2008 (C++) から Visual Studio 2010 に変更。これによって Windows 2000 は未サポートとなった。

2. 目次

3. Dixie とは

Dixie（ディキシー）は、GEM、HSMS、SECS の通信シミュレータです。

1.1 使用環境

- Windows XP、Windows Vista、Windows 7、Windows 8 または Windows 8.1

4. セットアップ

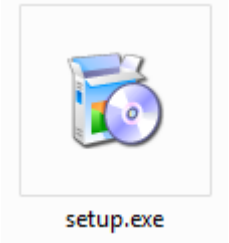
1.2 Dixie のインストール

Windows には Administrator の権限でログインしてください。もし Windows Vista、Windows 7、Windows 8 もしくは Windows 8.1 をお使いでユーザアカウントコントロール（UAC）が有効になっている場合、インストールに失敗することがあります。一時的に UAC を無効にしてください。

Dixie は Jazz Soft Semiconductor Solution の一部です。Setup.exe を実行してインストールします。

1 旧バージョンの Jazz Soft Semiconductor Solution がインストールされている場合は、先にアンインストールしてください。アンインストールはコントロールパネルの「Programs and Features」から行えます。

2 Setup.exe を実行します。Microsoft Visual C++ 2008 SP1 用コンポーネント群がインストールされていない場合は、最初にそれらがインストールされます。



3 Jazz Soft Semiconductor Solution のセットアップ画面が表示されます。Next ボタンをクリックします。

4 インストールするフォルダとユーザアカウントを選択します。通常は何も変更しないで Next ボタンをクリックします。

5 インストール前の確認画面です。Next ボタンをクリックします。

6 インストールが始まります。

7 インストールが正常に完了すると下記の画面になります。Close ボタンをクリックします。

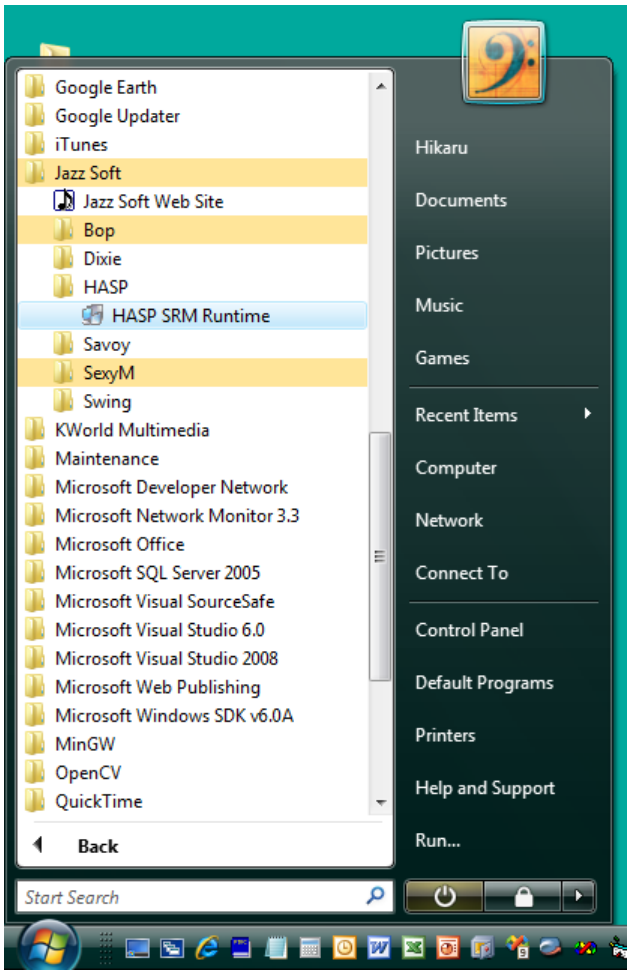
1.3 HASP ドライバのインストール

HASP キー用のドライバは試用版では必要ありませんが、製品版として使用する場合には必要となります。HASP ドライバのセットアップに必要なソフトウェアは、Jazz Soft Semiconductor Solution のセットアップでコピーされます。

HASP ドライバをインストールする際には、HASP キーをコンピュータに挿さないでください。

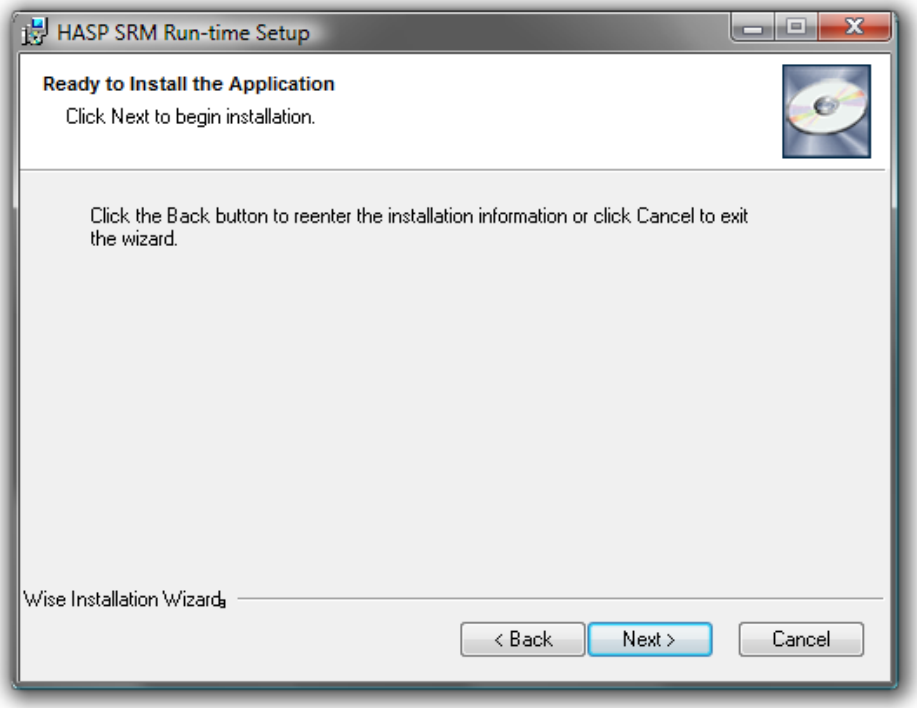
---

1 スタートメニューの「Jazz Soft」 – 「HASP」から「HASP SRM Runtime」をクリックします。

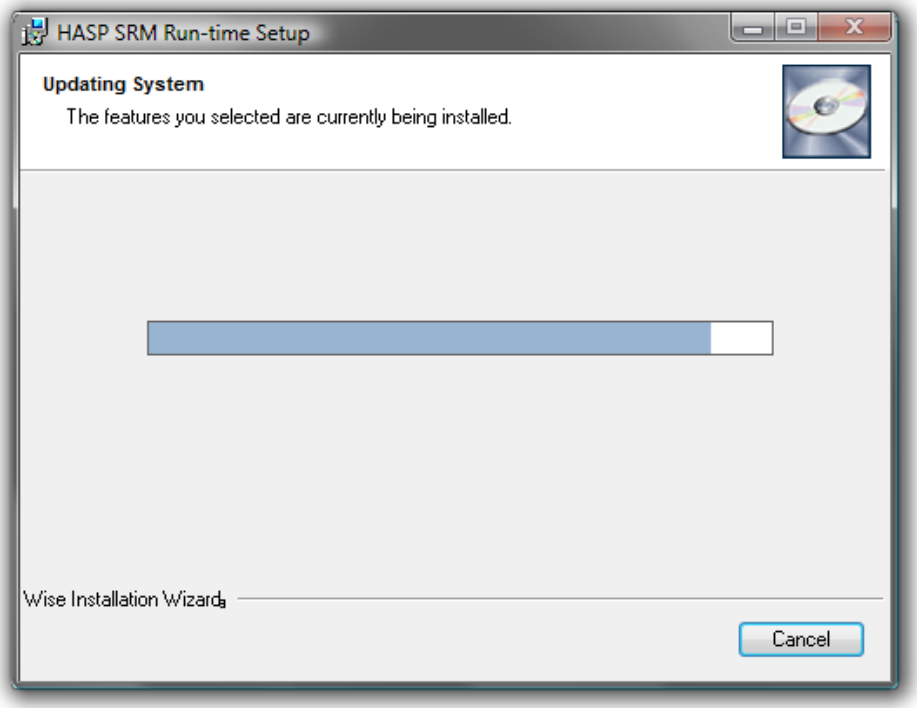


2 ウェルカム画面が表示されます。Next ボタンをクリックします。

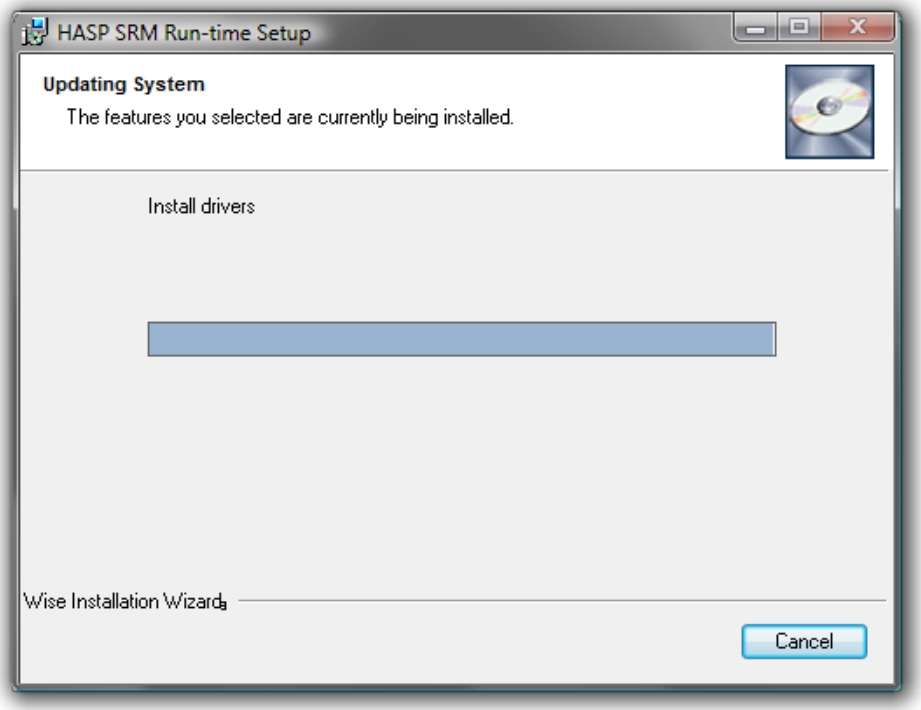
3 インストール前の確認が面が表示されます。Install ボタンをクリックします。



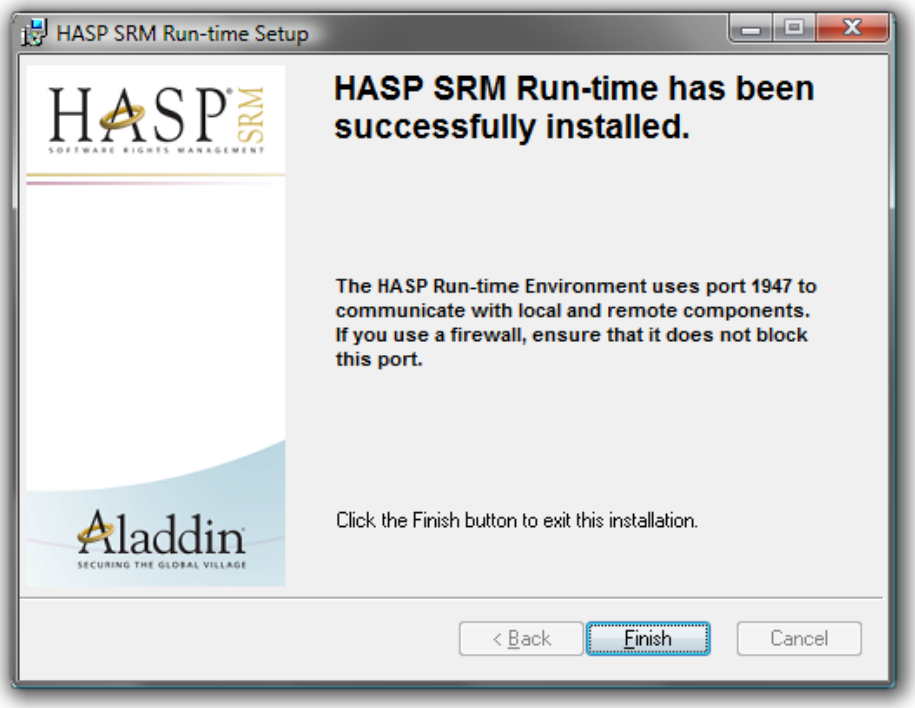
4 インストールが始まります。



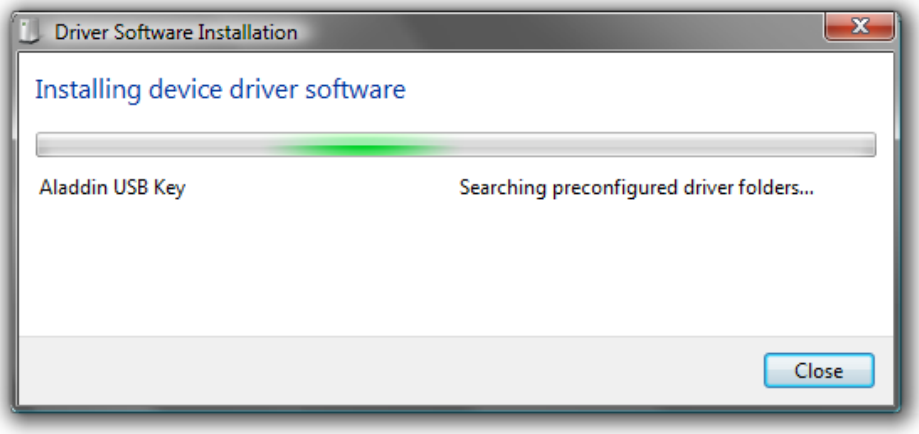
5 「Install drivers」の表示のままで数分かかることがあります。



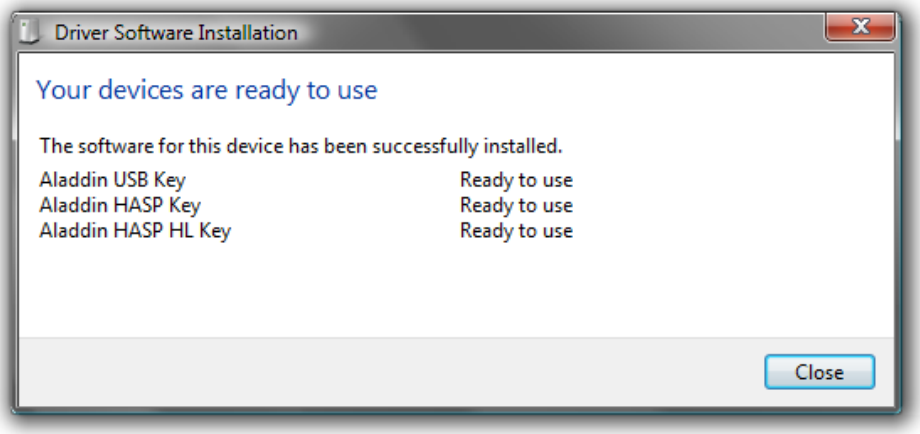
6 ドライバのコピーが正常に完了すると下記の画面になります。Finish ボタンをクリックします。



7 ここで HASP キーをコンピュータに挿します。Windows Vista の場合は画面右下にドライバのインストール状況が表示されます。



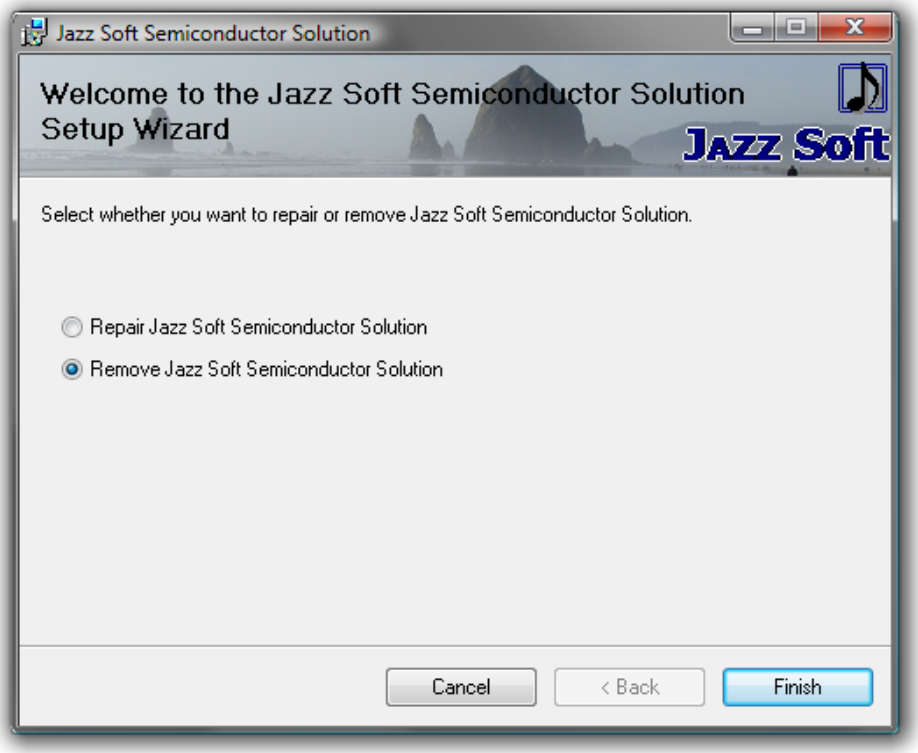
8 インストールが正常に完了すると下記の画面になります。Close ボタンをクリックします。



1.4 Dixie のアンインストール

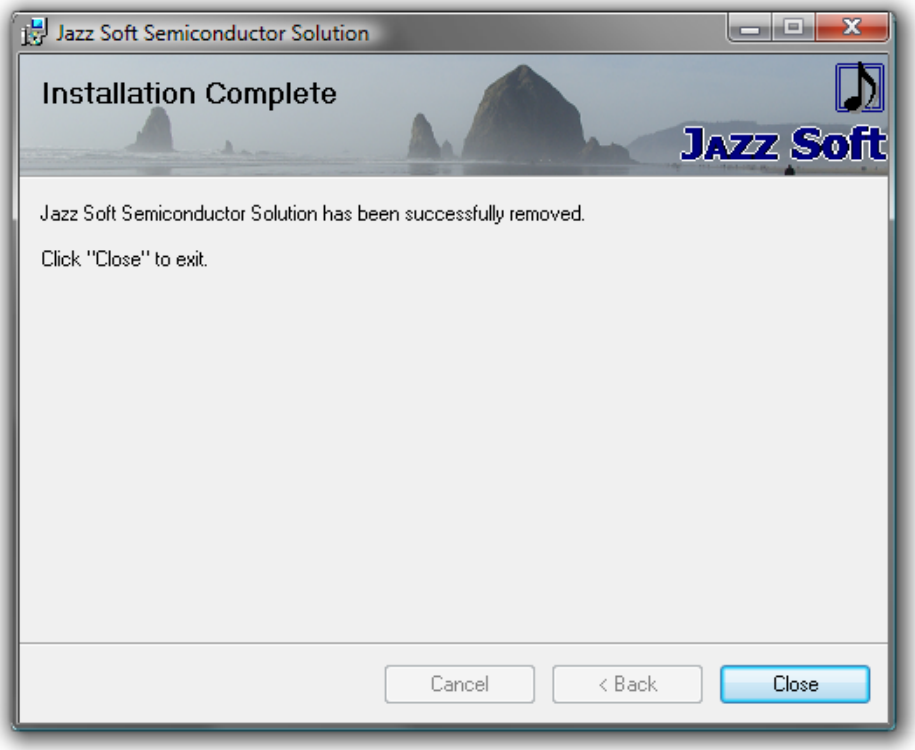
アンインストールはコントロールパネルの「Programs and Features」から行えます。また Setup.exe を実行しても行えます。

1 インストールしたときの Setup.exe を実行します。修復インストールするか、アンインストールするか聞いてきますので、「Remove Jazz Soft Semiconductor Solution」を選択し、Finish ボタンをクリックします。



2 アンインストールが始まると進捗状況が表示されます。

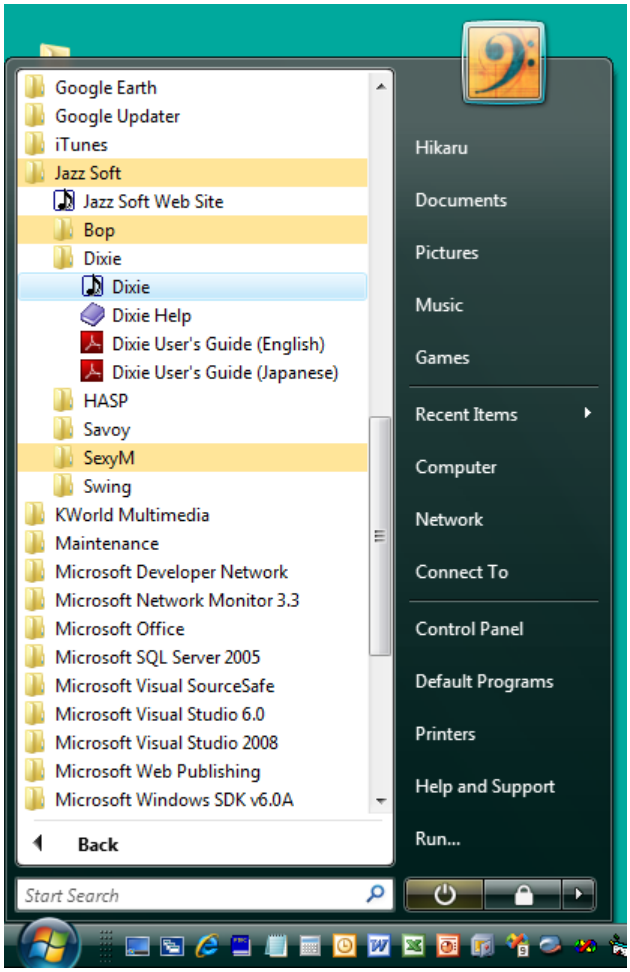
3 アンインストールが完了しました。Close ボタンをクリックします。



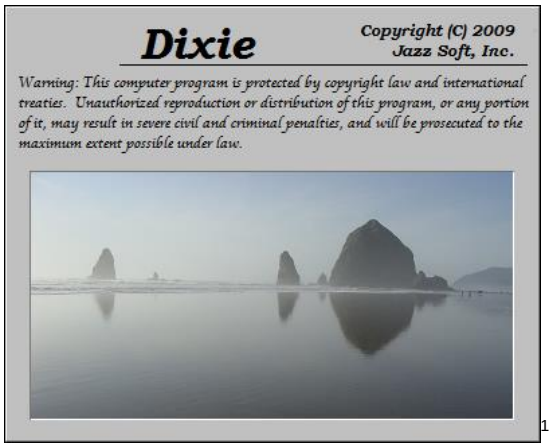
5. 操作の概要

1.5 起動のしかた

1 スタートメニューの「Jazz Soft」 – 「Dixie」から Dixie をクリックします。



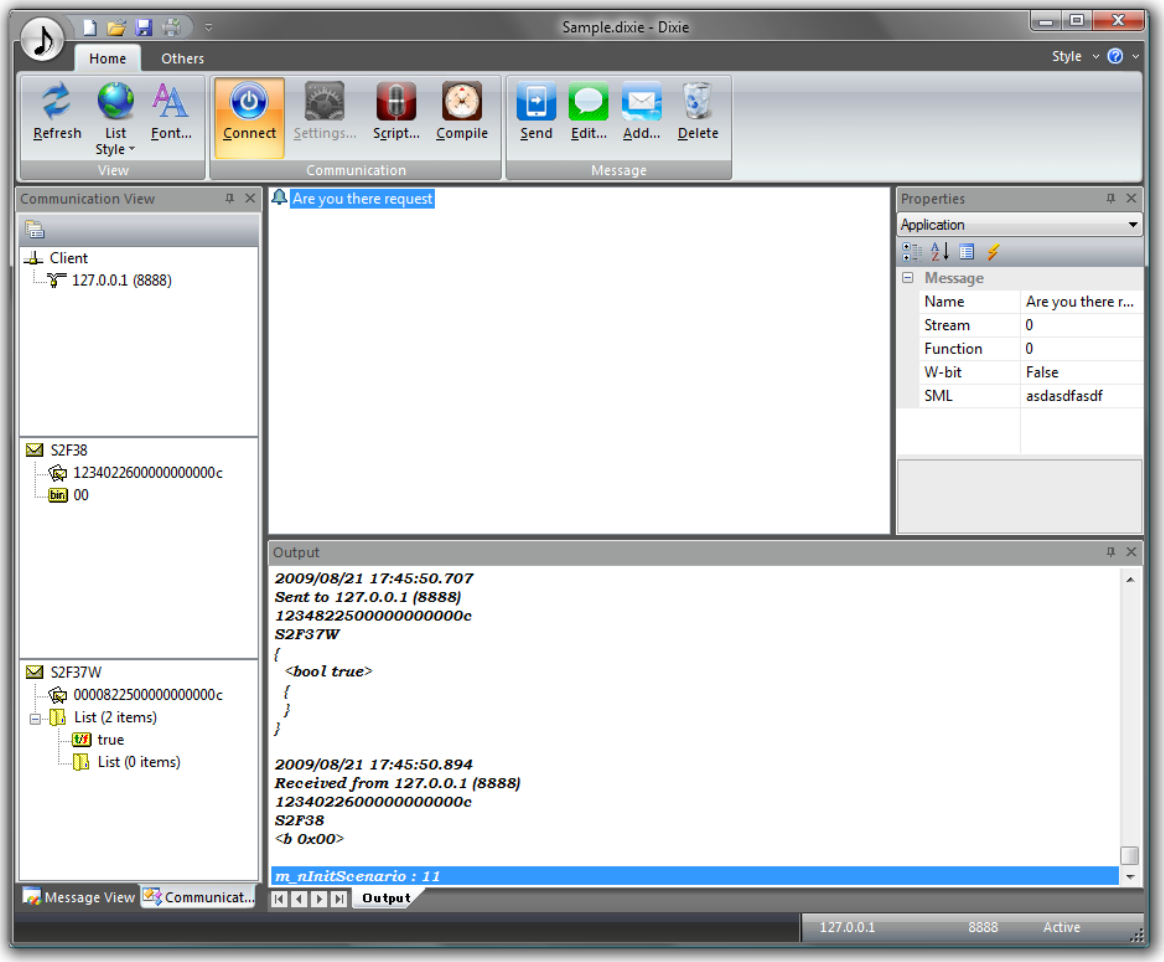
2 スプラッシュ・スクリーンが表示され、続いて Dixie のメイン画面が現れます。



[1]

[1] この写真は映画「グーニーズ」の舞台となったアメリカのオレゴン州キャノンビーチで撮影されました。

3 Microsoft Office 2007 ユーザインターフェースに基づいたメイン画面が表示されます。



1.6 リボン

Dixie は Microsoft Office 2007 ユーザインターフェースに準拠しています。このため従来のソフトウェアにはありがちなメニューがなく、代わりにリボンと呼ばれるタブが表示されます。

1.6.1 Home カテゴリ

View

アイコン	項目	説明
	Refresh	メッセージ一覧を再描画します。
	List Style	メッセージ一覧の表示形式を指定します。List または Detail のいずれかを選択します。
	Font	アウトプット欄で使用するフォントを選択します。固定ピッチのフォントを推奨します。

Communication

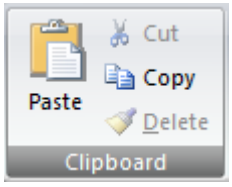
アイコン	項目	説明
	Connect	通信を接続または切断します。
	Settings	環境設定画面を開きます。
	Script	スクリプトの編集を行います。
	Compile	スクリプトをコンパイルします。

Message

アイコン	項目	説明
	Send	選択されているメッセージを送信します。
	Edit	選択されているメッセージを編集します。
	Add	新しくメッセージを追加します。
	Delete	選択されているメッセージを削除します。

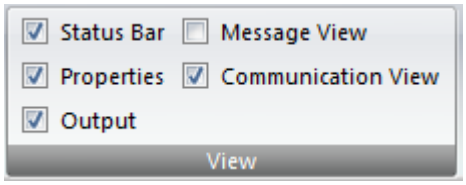
1.6.2 Others カテゴリ

Clipboard



アイコン	項目	説明
	Paste	メッセージをクリップボードから貼り付けます。
	Cut	（未使用）
	Copy	メッセージをクリップボードにコピーします。
	Delete	（未使用）

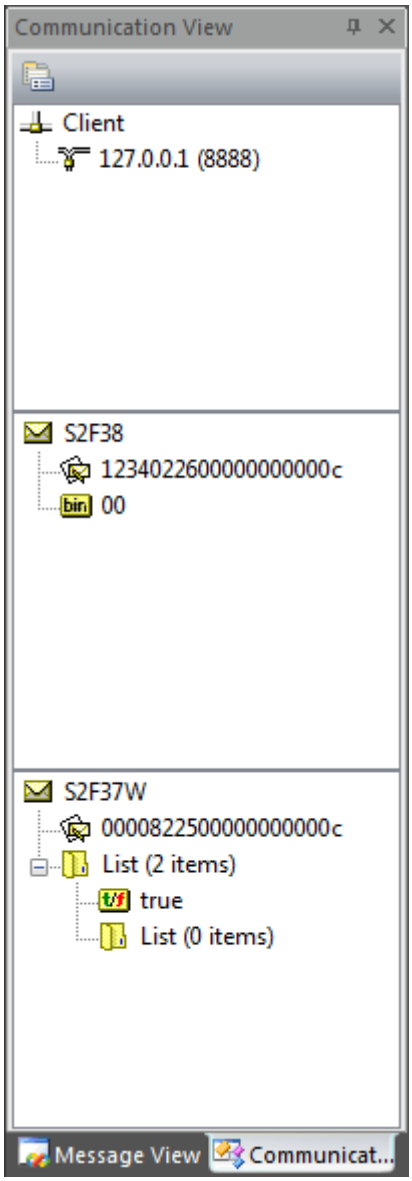
View



アイコン	項目	説明
	Status Bar	ステータスバーを表示または非表示にします。
	Properties	プロパティ欄を表示または非表示にします。
	Output	アウトプット欄を表示または非表示にします。
	Message View	メッセージビューを表示または非表示にします。
	Communication View	コミュニケーションビューを表示または非表示にします。

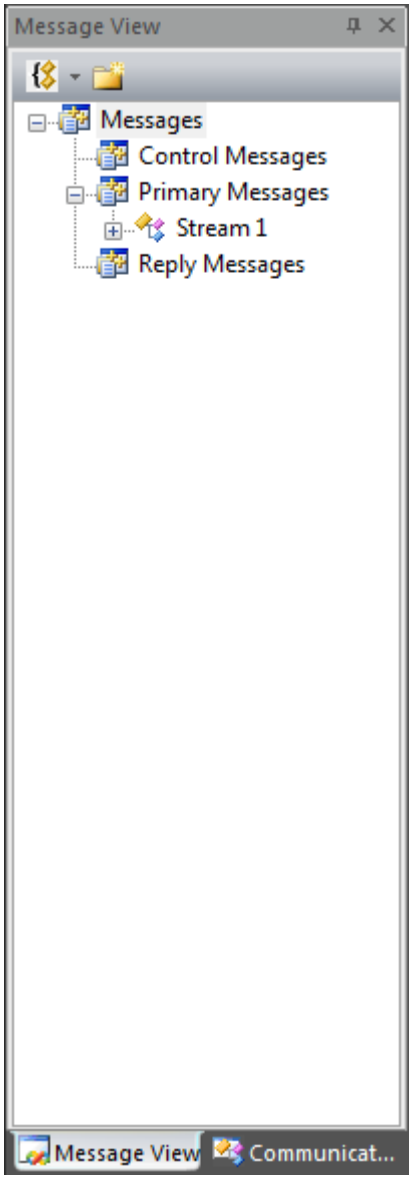
1.7　コミュニケーションビュー

通信の状態を表示します。一番上が SavoyHsms、下二つは SavoySecsII を使用しています。



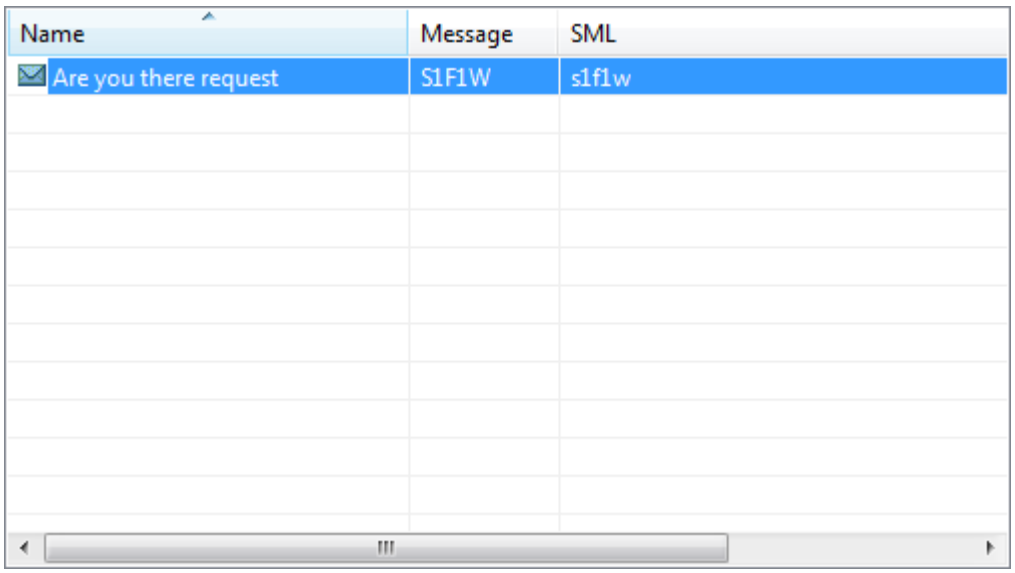
1.8 メッセージビュー

登録されているメッセージのサマリーを表示します。例えば「Primary Messages」を選択すると、メッセージ一覧にはプライマリメッセージだけが表示されるようになります。全てのメッセージを表示したい場合は、一番上にある「Messages」を選択します。



1.9 メッセージ一覧

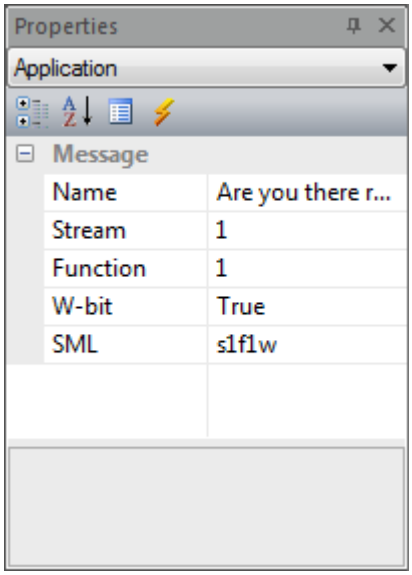
メッセージの一覧を表示します。この画面は List Style の設定によって、表示のしかたが異なります。「Detail」が選択されている場合は、以下のように詳細情報まで表示されます。



「List」が選択されている場合は、名称だけが表示されます。メッセージの登録数が増えた場合は、この表示の方が便利なこともあります。

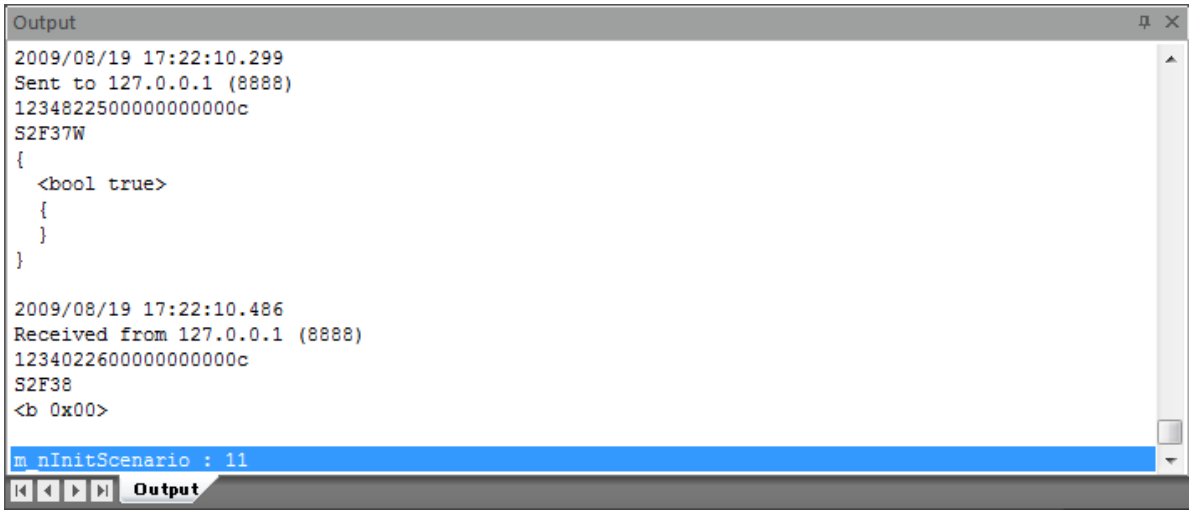
1.10 プロパティ欄

メッセージのプロパティが表示されます。



1.11 アウトプット欄

通信ログやコンパイル状況などが表示されます。



一番最後の行が選択されている場合は、新たな事象が発生した場合に次々と更新されます。

1.12 ステータスバー

接続状態などを表示します。

6. 画面の説明

1.13 Home – View パネル

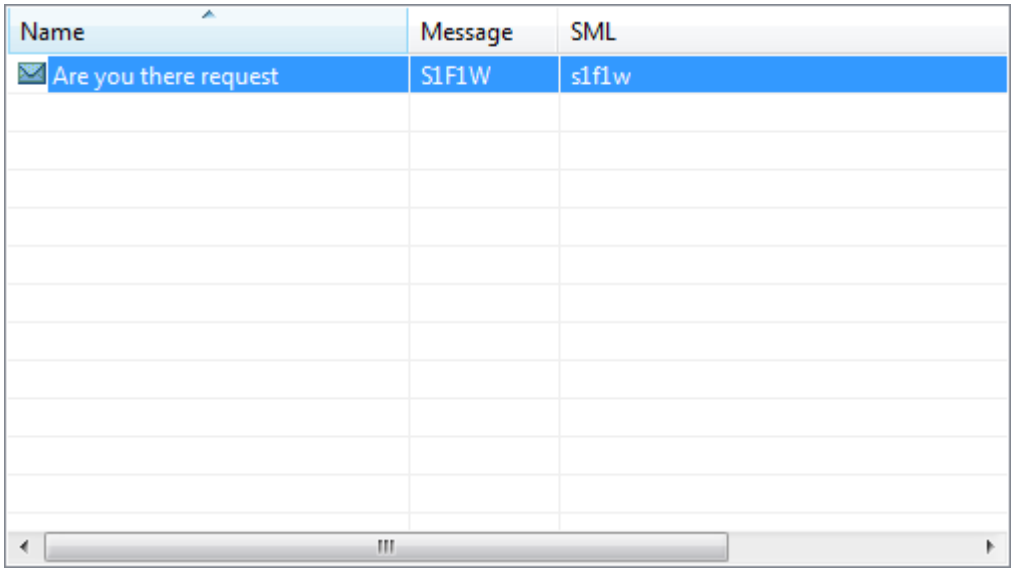
1.13.1 Refresh

メッセージ一覧を再描画します。

1.13.2 List Style

メッセージ一覧の表示形式を指定します。List または Detail のいずれかを選択します。List を選択すると、以下のように名称だけが表示されます。

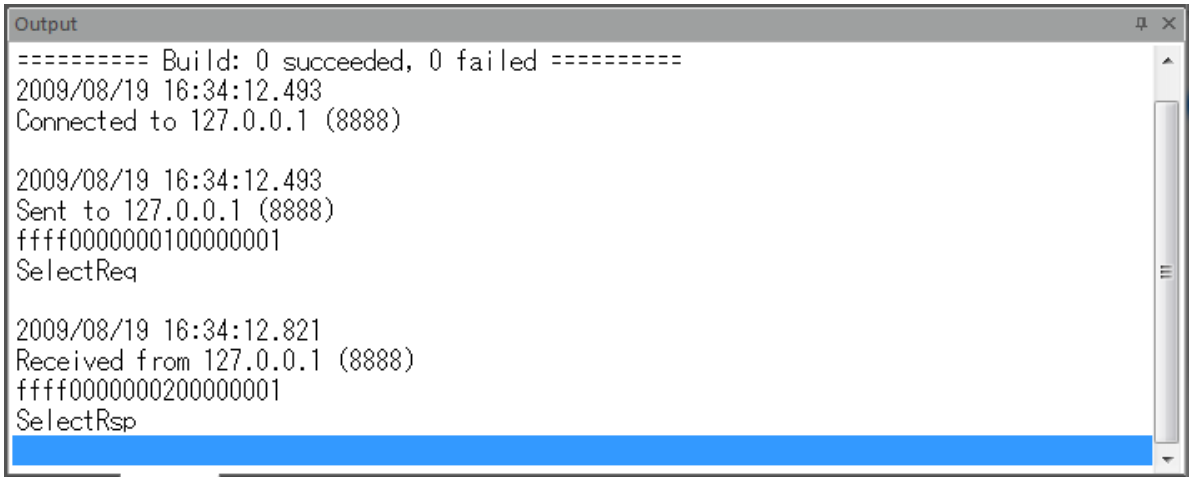
Detail を選択すると、詳細情報まで表示されます。



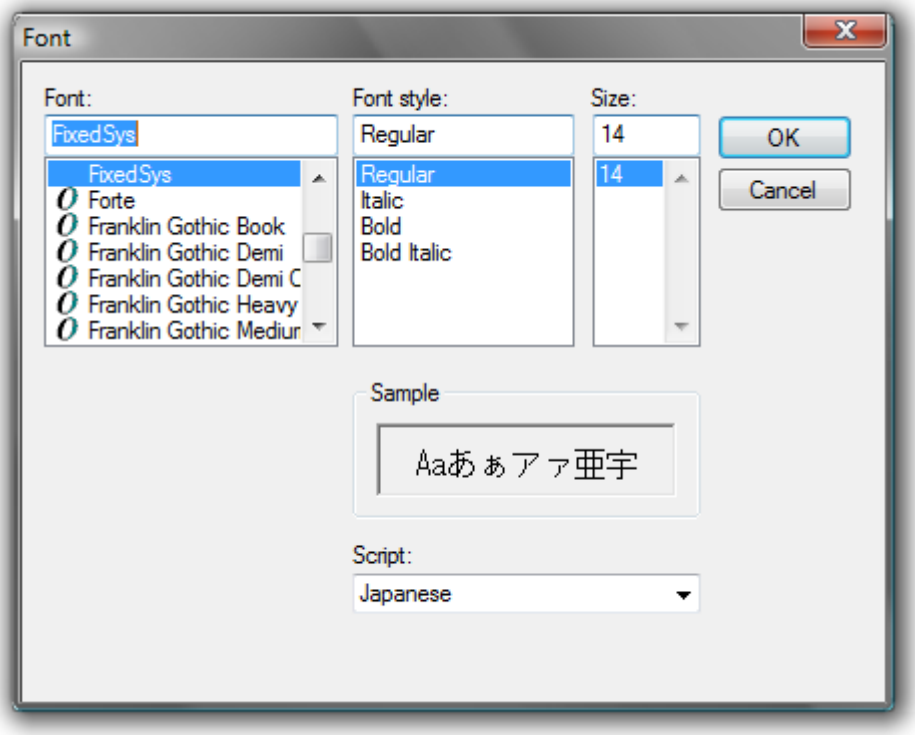
1.13.3 Font

アウトプット欄で使用するフォントを選択します。メッセージ編集画面の SML 文字列およびスクリプト編集画面のスクリプト文字列でもこのフォントが使われます。

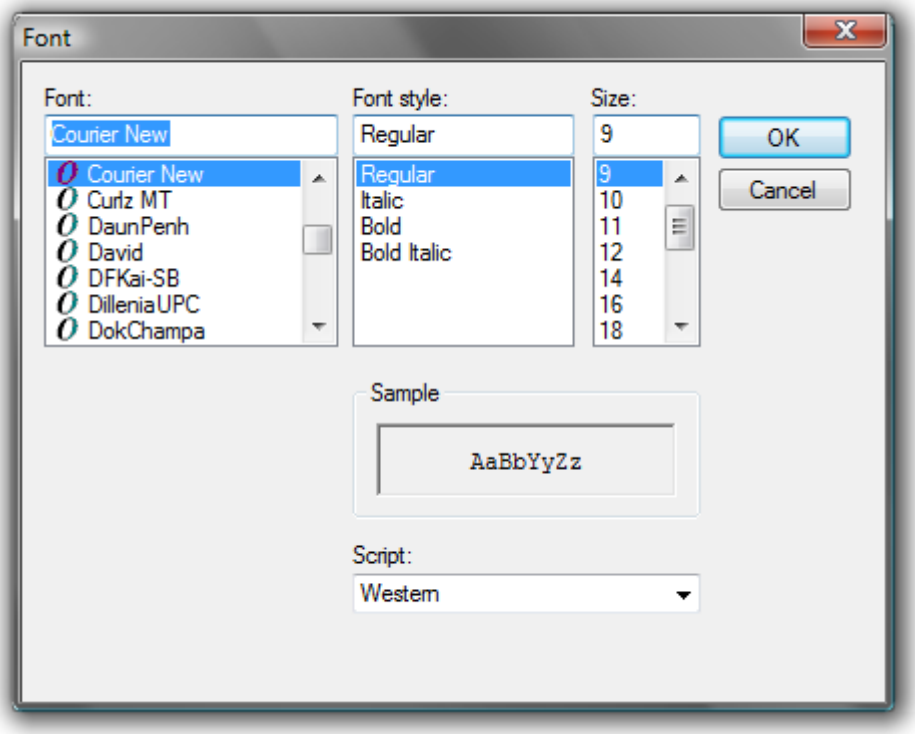
起動時は以下のように FixedSys フォントが選択されています。



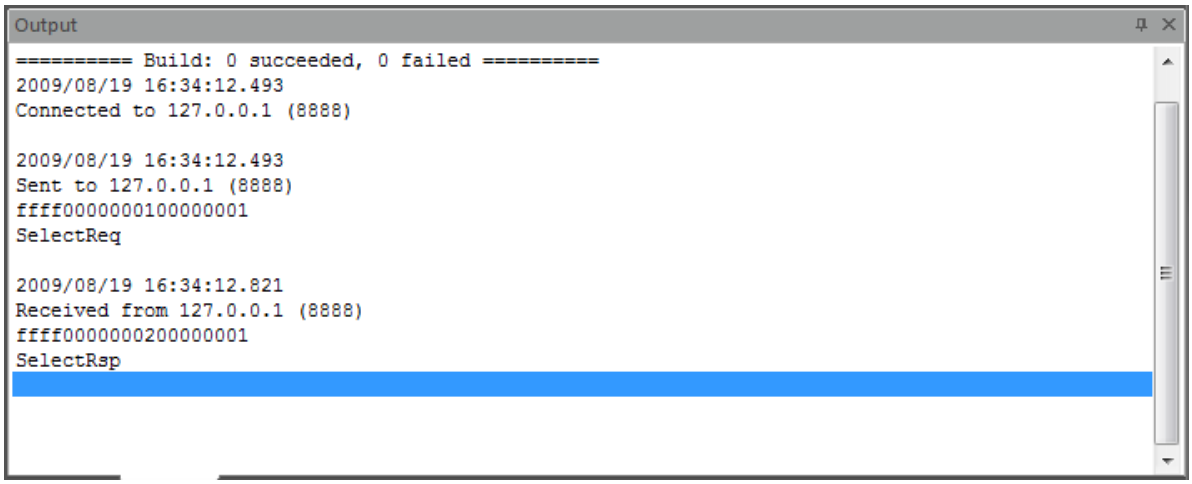
Font 選択ダイアログボックスを開きます。



これを 9 ポイントの Courier New フォントに変更してみます。SML 文字列等は等幅のフォントで表示したほうが読みやすいため、固定ピッチのフォントを推奨します。



新しく選択されたフォントで画面が自動的に更新されます。



1.14 Home – Communication パネル

1.14.1 Connect

通信を接続または切断します。通信が切断されている状態のときは以下のように灰色で表示されます。

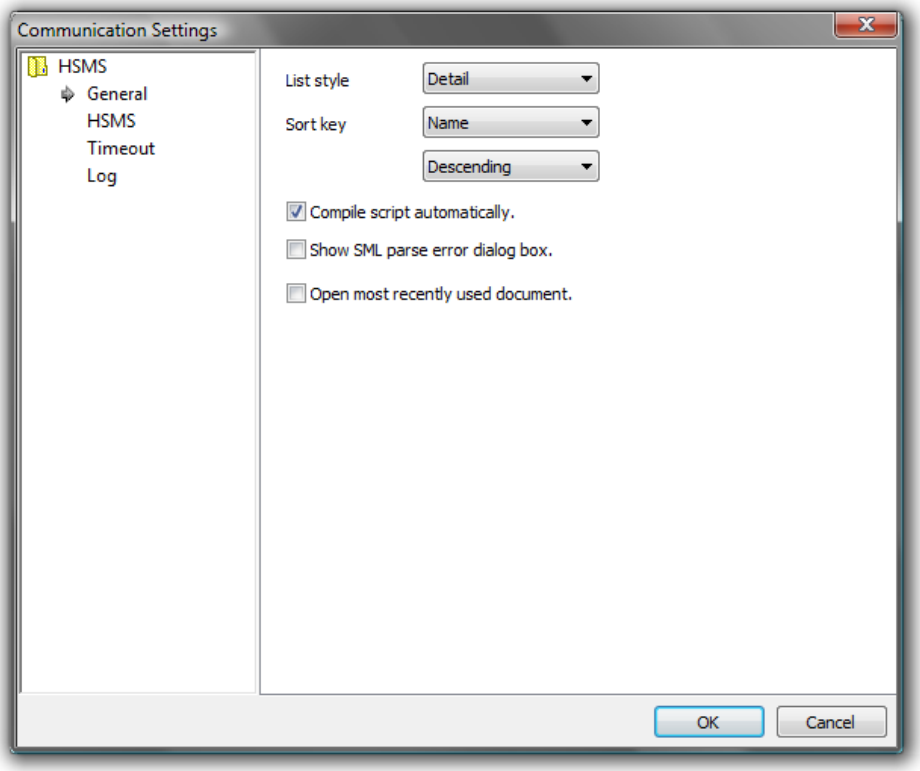
通信が接続状態になると以下のようにオレンジ色に変わります。[2]

[2] パッシブエンティティの場合、アクティブエンティティからの接続が成立していなくても「接続待ち状態」になればこのように表示されます。

1.14.2 Settings

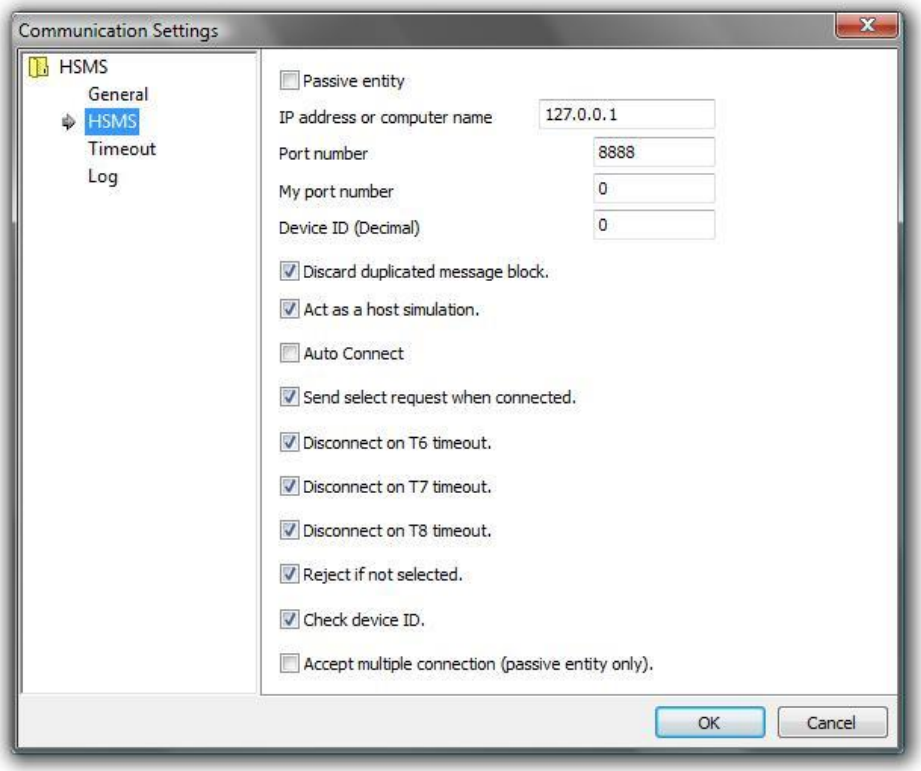
環境設定画面を開きます。通信が接続状態のときは、このボタンは無効になります。

General



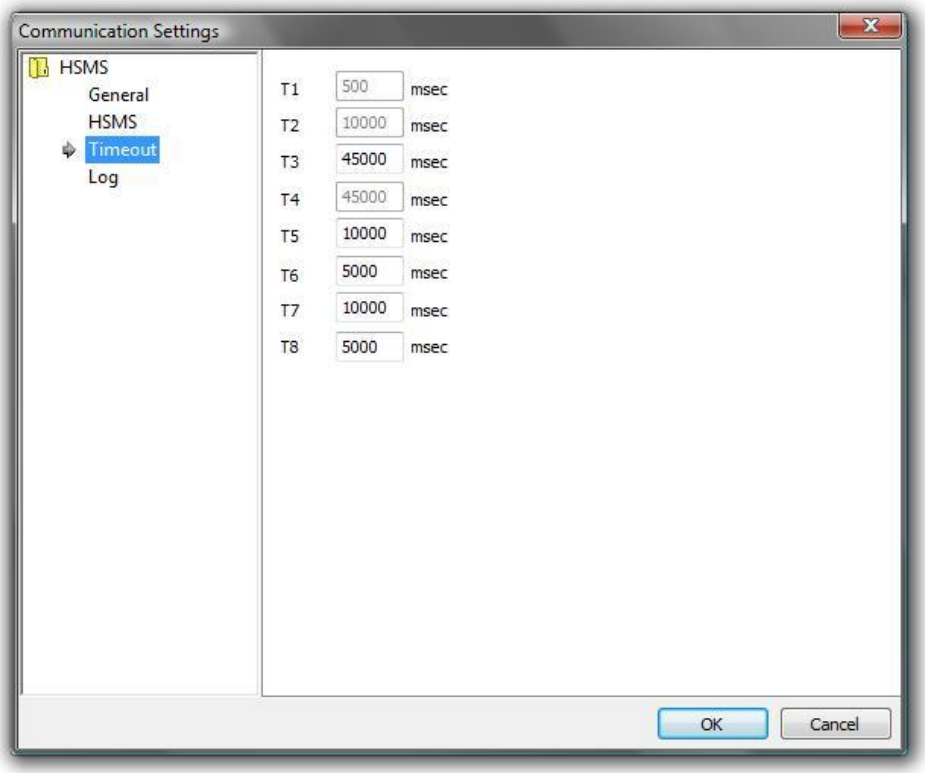
項目	説明
List style	メッセージ一覧の表示形式を指定します。List または Detail のいずれかを選択します。List を選択すると名称だけが表示され、Detail を選択すると詳細情報まで表示されます。
Sort key	メッセージ一覧のソートキーを指定します。Name、Message、SML のいずれかを選択します。
	メッセージ一覧のソートの順序を指定します。Ascending または Descending のいずれかを選択します。
Compile script automatically.	スクリプトを変更した場合に自動的にコンパイルします。なおこの設定のいかんに関わらず、Dixie 起動時には自動的にコンパイルされます。
Show SML parse error dialog box.	メッセージを変更した場合にSML 文字列をチェックし、文法的な誤りがあればダイアログボックスでエラーを表示します。
Open most recently used document.	最後に使った.dixie ファイルを次回起動時に自動的にオープンするかどうかを指定します。この設定は Windows のレジストリに記録されます。

HSMS



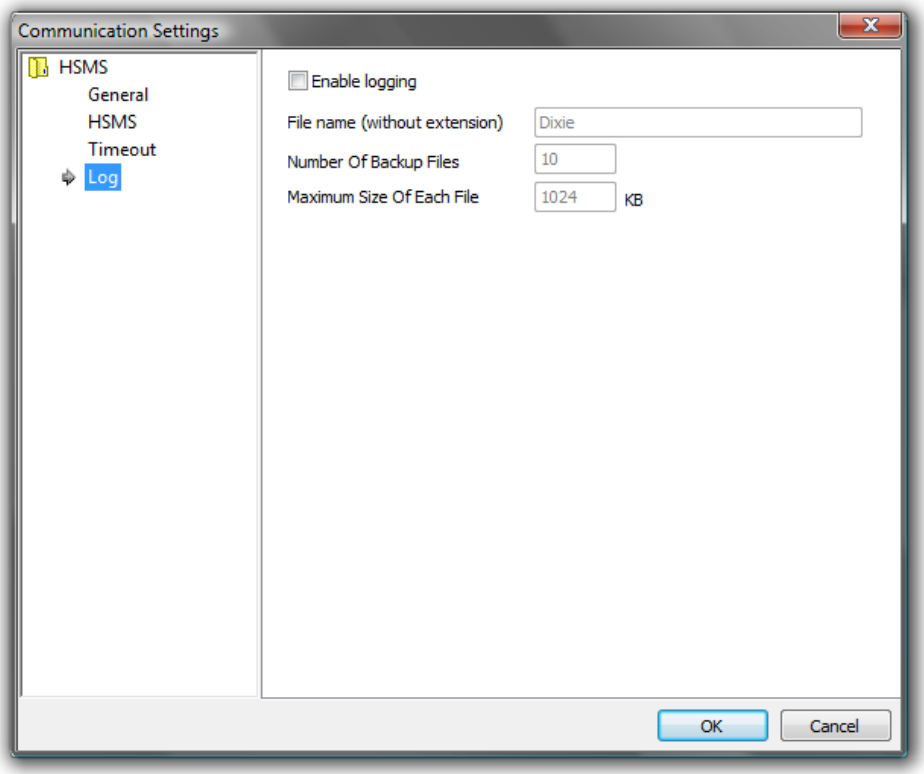
項目	説明
Passive entity	パッシブエンティティかアクティブエンティティかを指定します。
IP address or computer name	IP アドレスまたはコンピュータ名を指定します。この設定を空欄にした場合は、自分自身（ローカルホスト）と解釈されます。パッシブエンティティの場合はこの設定は無効となります。
Port number	接続するポート番号を設定します。パッシブエンティティの場合はこの設定は無効となります。
My port number	自分自身のローカルポート番号を指定します。アクティブエンティティの場合には 0 を指定することを強く推奨します。
Device ID (Decimal)	デバイス ID を指定します。
Discard duplicated message block.	二重ブロックを検出するかどうかを指定します。
Act as a host simulation.	ホストシミュレータとして動作するか、装置シミュレータとして動作するかを指定します。
Auto Connect	起動時に自動的に接続するかどうかを指定します。
Send select request when connected.	TCP/IP の接続に成功したら自動的にセレクト要求を送信するかどうかを指定します。HSMS では通常はアクティブエンティティがセレクト要求を送信します。
Disconnect on T6 timeout.	T6 タイムアウトが発生したら切断するかどうかを指定します。
Disconnect on T7 timeout.	T7 タイムアウトが発生したら切断するかどうかを指定します。
Disconnect on T8 timeout.	T8 タイムアウトが発生したら切断するかどうかを指定します。
Reject if not selected.	セレクトされていない場合に全てのデータメッセージを拒否するかどうかを指定します。
Check device ID.	デバイス ID が異なるメッセージを受信したらメッセージを破棄するかどうかを指定します。装置シミュレータとして設定されている場合には S9F1 を送信します。
Accept multiple connections.	複数の接続を受け入れるかどうかを指定します。アクティブエンティティの場合はこの設定は無効となります。

Timeout



項目	説明
T1	（未使用）
T2	（未使用）
T3	T3 タイムアウトをミリ秒単位で指定します。
T4	（未使用）
T5	T5 タイムアウトをミリ秒単位で指定します。
T6	T6 タイムアウトをミリ秒単位で指定します。
T7	T7 タイムアウトをミリ秒単位で指定します。
T8	T8 タイムアウトをミリ秒単位で指定します。

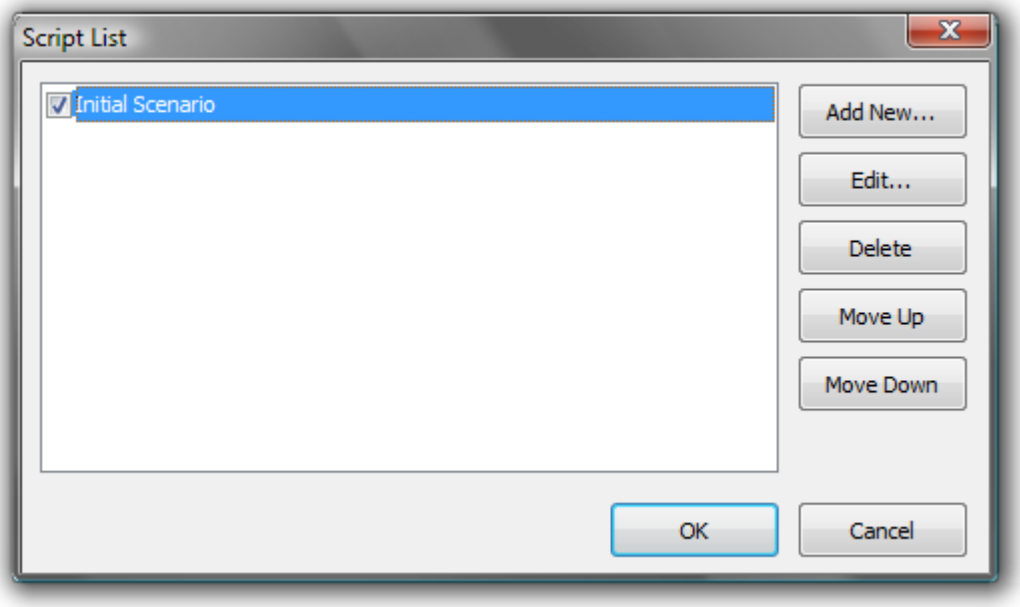
Log



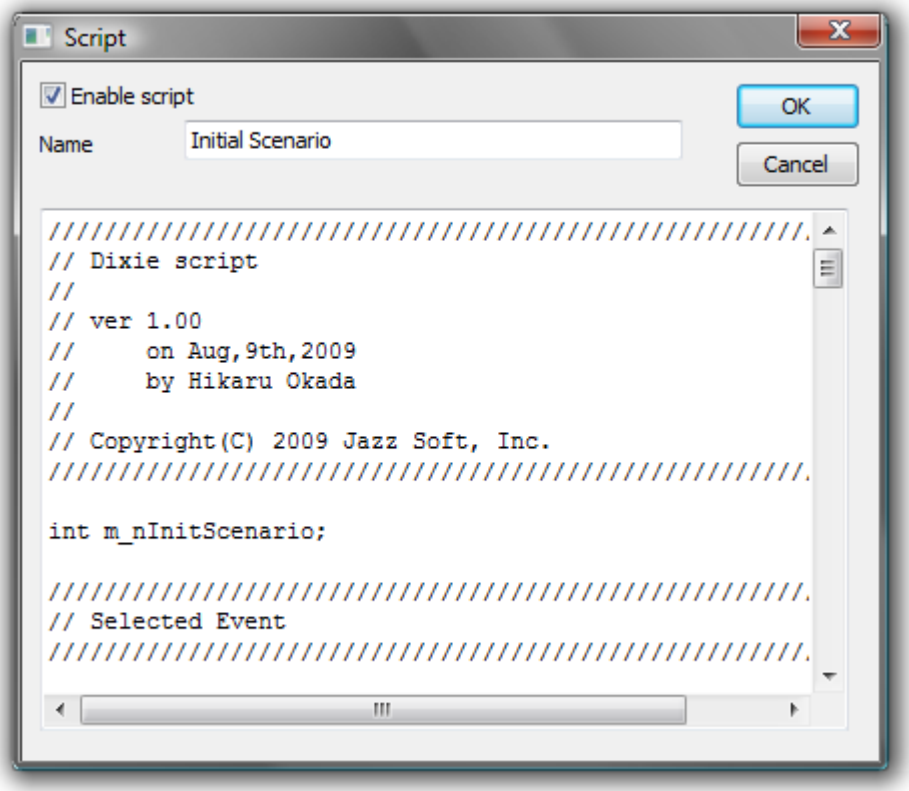
項目	説明
Enable logging	通信内容をログに記録するかどうかを指定します。
File name (without extension)	ログファイルの名前を指定します。ファイル名には拡張子をつけないでください（Dixie によって自動的に付加されます）。
Number of backup files	バックアップファイルの個数を指定します。バックアップファイルの個数がこの設定を超えると古いものから順に削除されます。
Maximum size of each file	一つのログファイルのファイルサイズをキロバイト単位で指定します。ログファイルがこのサイズを超えるとバックアップファイルを作成します。

1.14.3 Script

スクリプトの編集を行います。スクリプトには優先順位があり、リストの上位にランクされたものが優先的に実行されます。



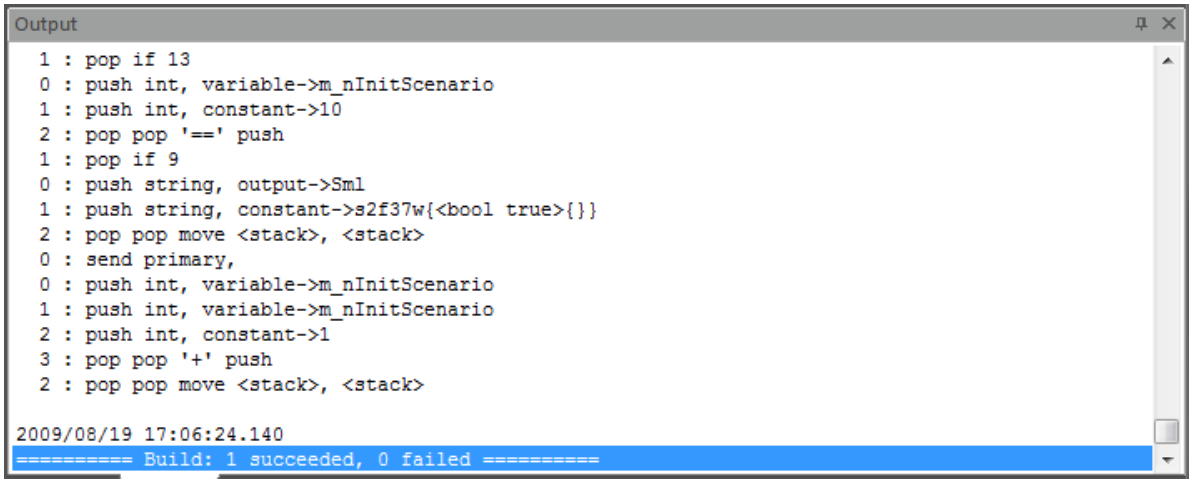
スクリプトのソースコードを編集します。ダイアログボックスのサイズは変更することが可能です。この画面には特に高度な編集機能はありませんので、Visual Studio 2008 などでソースコードを記述し、コピー＆ペーストされることを推奨します。



1.14.4 Compile

スクリプトをコンパイルします。複数のスクリプトが定義されている場合も全てコンパイルされます。ただし途中でコンパイルエラーが発生した場合は、そこでコンパイルを中断します。コンパイル結果に「0 failed」と表示されていれば、正常にコンパイルが完了したことを示します。

Output 欄にはコンパイルされたバイナリのニモニックが表示されます。一番左側の数字はスタック深度を表します。



1.15 Home – Message パネル

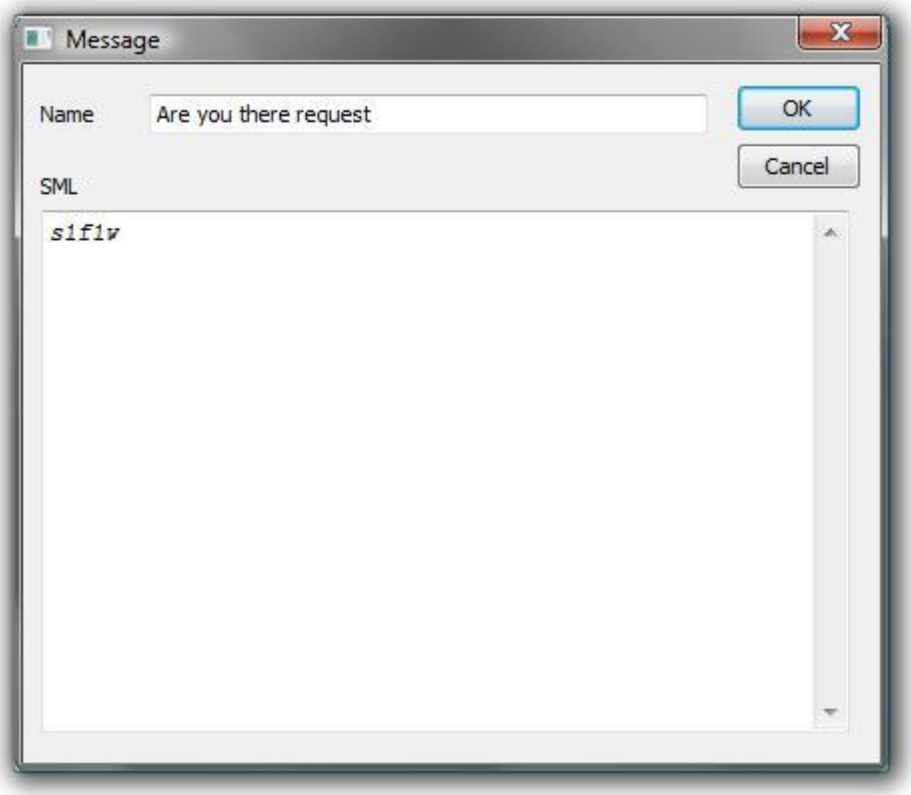
1.15.1 Send

選択されているメッセージを送信します。メッセージが選択されていないときは、このボタンは無効となります。

1.15.2 Edit

選択されているメッセージを編集します。メッセージが選択されていないときは、このボタンは無効となります。

ダイアログボックスのサイズは変更することが可能です。この画面には特に高度な編集機能はありませんので、Visual Studio 2008 などで SML 文字列を記述し、コピー＆ペーストされることを推奨します。



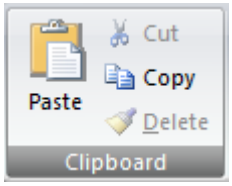
1.15.3 Add

新しくメッセージを追加します。

1.15.4 Delete

選択されているメッセージを削除します。メッセージが選択されていないときは、このボタンは無効となります。

1.16 Others – Clipboard パネル



1.16.1 Paste

クリップボードで選択されているメッセージをペーストします。クリップボードに Dixie メッセージが選択されていない場合は、このボタンは無効となります。

1.16.2 Cut

この機能は現時点では未使用です。

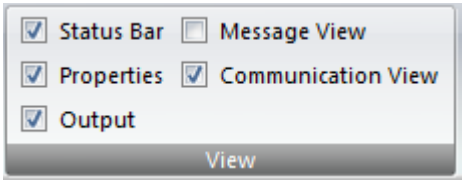
1.16.3 Copy

選択されているメッセージをクリップボードにコピーします。メッセージが選択されていないときは、このボタンは無効となります。

1.16.4 Delete

この機能は現時点では未使用です。

1.17 Others – View パネル



1.17.1 Status Bar

ステータスバーを表示します。

1.17.2 Properties

プロパティ欄を表示します。

1.17.3 Output

アウトプット欄を表示します。

1.17.4 Message View

メッセージビューを表示します。

1.17.5 Communication View

コミュニケーションビューを表示します。

7. チュートリアル

1.18 初期化スクリプト

ここでは GEM の初期化を行うホストシミュレータを作成しながらスクリプトについて解説していくことにします。

1.18.1 初期化シナリオの仕様

作成するホストシミュレータの初期化シナリオ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。Dixie を起動して装置と接続した際に、この一連のシナリオが自動的に流れるようにするのが目標です。

#	Dixie		装置
1	S1F13	→→→→→	
1a		←←←←←	S1F14
2	S1F1	→→→→→	
2a		←←←←←	S1F2
3	S2F37（イベント無効）	→→→→→	
3a		←←←←←	S2F38
4	S2F33（レポート消去）	→→→→→	
4a		←←←←←	S2F34
5	S2F33（レポート定義）	→→→→→	
5a		←←←←←	S2F34
6	S2F35（リンク）	→→→→→	
6a		←←←←←	S2F36
7	S2F43（スプール中止）	→→→→→	
7a		←←←←←	S2F44
8	S5F3（アラーム有効）	→→→→→	
8a		←←←←←	S5F4
9	S2F15（装置定数変更）	→→→→→	
9a		←←←←←	S2F16
10	S1F3（状態変数）	→→→→→	
10a		←←←←←	S1F4
11	S2F37（イベント有効）	→→→→→	
11a		←←←←←	S2F38

1.18.2 状態遷移

このシナリオの一行一行が「状態」であると考えることができます。例えば S1F13 を送信したら、「S1F14 を待っている状態」に遷移します。S1F14 を受け取ったら S1F1 を送信して「S1F2 を待っている状態」に遷移します。このように個々の状態を知ることが重要です。この手法はスクリプト処理以外にも応用ができ、Savoy を使って装置やホストの通信ソフトを記述する際に非常に役に立ちます。

シナリオの一番左のカラムに「1」や「2a」のような記号がついていますが、これらが状態番号に相当します。今回作成するシナリオでは 11 の状態に、「何もしない状態」を加えた 12 の状態が存在することになります。

状態を管理するには変数を使います。状態は整数なので int 型の変数を使用します。Dixie の命名規約として、全ての変数は「m_」で始める必要があります。

Dixie Script

```
int m_nInitScenario;
```

1.18.3 Selected イベント処理

シナリオを進めていくには「装置と接続が完了した」とか、「S1F2 を受信した」などのイベントを捕らえ、それに応じて状態を遷移させるかどうかを決めて行きます。

シナリオを開始するトリガーとなるのは、HSMS 通信が「セレクト状態になった」というイベントです。これを得るには OnSelected() イベントハンドラ関数を記述します。シナリオの推移が分かるよう、コメントを出力すると分かりやすいです。

Dixie Script

```
bool OnSelected()
{
    cout << "Init scenario started";
}
```

最初の状態は「S1F13 を送信し、S1F14 を受信するのを待つ」というものです。S1F13 の SML 文字列をセットし Send()関数を呼んで送信します。そして m_nInitScenario に 1 をセットします。

Dixie Script

```
    out.SML = "s1f13w{}";
    Send();
    m_nInitScenario = 1;
```

1.18.4 Received イベント処理

次に S1F14 を受信したイベントを得る必要があります。それには OnReceived()イベントハンドラ関数を記述します。コメントとして m_nInitScenario 変数の内容も表示することにします。

Dixie Script

```
bool OnReceived()
{
    cout << "m_nInitScenario : " + m_nInitScenario;
}
```

if 文を使って受信メッセージのストリーム番号とファンクション番号をチェックします。これが S1F14 に一致していれば目的のメッセージだと判断できます。何らかの原因で予測しない S1F14 を受け取ることがあるかもしれませんので、m_nInitScenario 変数が 1 のときだけ処理することにします。

次の状態は「S1F1 を送信し、S1F2 を待つ」というものです。S1F1 の SML 文字列をセットし Send()関数を呼んで送信します。そして m_nInitScenario を一つインクリメントします。

Dixie Script

```
    if(in.Stream == 1 && in.Function == 14)
    {
        // S1F14
        if(m_nInitScenario == 1)
        {
            out.SML = "s1f1w";
            Send();
            m_nInitScenario = m_nInitScenario + 1;
        }
    }
```

1.18.5 同一メッセージの受信

シナリオをよく見ると S2F37 と S2F33 のトランザクションは2回ずつあることが分かります。装置からは S2F38 および S2F34 が返ってきますが、これらをどのように区別したらいいのでしょうか？それにはやはり m_nInitScenario 変数が一役買います。

例えば S2F34 を受信する局面としては、状態４のときと状態５のときがあります。これを if 文を使って場合分けすればいいのです。

Dixie Script

```
if(in.Stream == 2 && in.Function == 34)
{
	// S2F34
	if(m_nInitScenario == 4)
	{
		out.SML = "s2f33w{<bool true>}";
		Send();
		m_nInitScenario = m_nInitScenario + 1;
	}
	else
	if(m_nInitScenario == 5)
	{
		out.SML = "s2f35w{}";
		Send();
		m_nInitScenario = m_nInitScenario + 1;
	}
}
```

1.18.6 全ソースコード

以上の手順で機械的にそれぞれの状態の処理を記述していきます。これだけで初期化スクリプトの完成です。ソフトウェア開発と同様に、スクリプトの途中に分かりやすいコメントを入れるといいでしょう。以下は今回作成したスクリプトの全ソースコードです。

Dixie Script

```
////////////////////////////////////////////////////////
// Dixie script
//
// ver 1.00
//	on Aug,9th,2009
//	by Hikaru Okada
//
// Copyright(C) 2009 Jazz Soft, Inc.
////////////////////////////////////////////////////////

int m_nInitScenario;

////////////////////////////////////////////////////////
// Selected Event
////////////////////////////////////////////////////////
bool OnSelected()
{
	cout << "Init scenario started";

	out.SML = "s1f13w{}";
	Send();
	m_nInitScenario = 1;
}

////////////////////////////////////////////////////////
// Received Event
////////////////////////////////////////////////////////
bool OnReceived()
{
	cout << "m_nInitScenario : " + m_nInitScenario;

	if(in.Stream == 1 && in.Function == 14)
	{
		// S1F14
		if(m_nInitScenario == 1)
		{
			out.SML = "s1f1w";
```

```
            Send();
            m_nInitScenario = m_nInitScenario + 1;
        }
    }
    else
    if(in.Stream == 1 && in.Function == 2)
    {
        // S1F2
        if(m_nInitScenario == 2)
        {
            out.SML = "s2f37w{<bool false>{}}";
            Send();
            m_nInitScenario = m_nInitScenario + 1;
        }
    }
    else
    if(in.Stream == 2 && in.Function == 38)
    {
        // S2F38
        if(m_nInitScenario == 3)
        {
            out.SML = "s2f33w{<bool false>}";
            Send();
            m_nInitScenario = m_nInitScenario + 1;
        }
        else
        if(m_nInitScenario == 11)
        {
            m_nInitScenario = 0;
        }
    }
    else
    if(in.Stream == 2 && in.Function == 34)
    {
        // S2F34
        if(m_nInitScenario == 4)
        {
            out.SML = "s2f33w{<bool true>}";
            Send();
            m_nInitScenario = m_nInitScenario + 1;
        }
        else
        if(m_nInitScenario == 5)
        {
            out.SML = "s2f35w{}";
            Send();
            m_nInitScenario = m_nInitScenario + 1;
        }
    }
    else
    if(in.Stream == 2 && in.Function == 36)
    {
        // S2F36
        if(m_nInitScenario == 6)
        {
            out.SML = "s2f43w{}";
            Send();
            m_nInitScenario = m_nInitScenario + 1;
        }
    }
    else
    if(in.Stream == 2 && in.Function == 44)
    {
        // S2F44
        if(m_nInitScenario == 7)
        {
            out.SML = "s5f3w{<b 0x80><u4>}";
            Send();
```

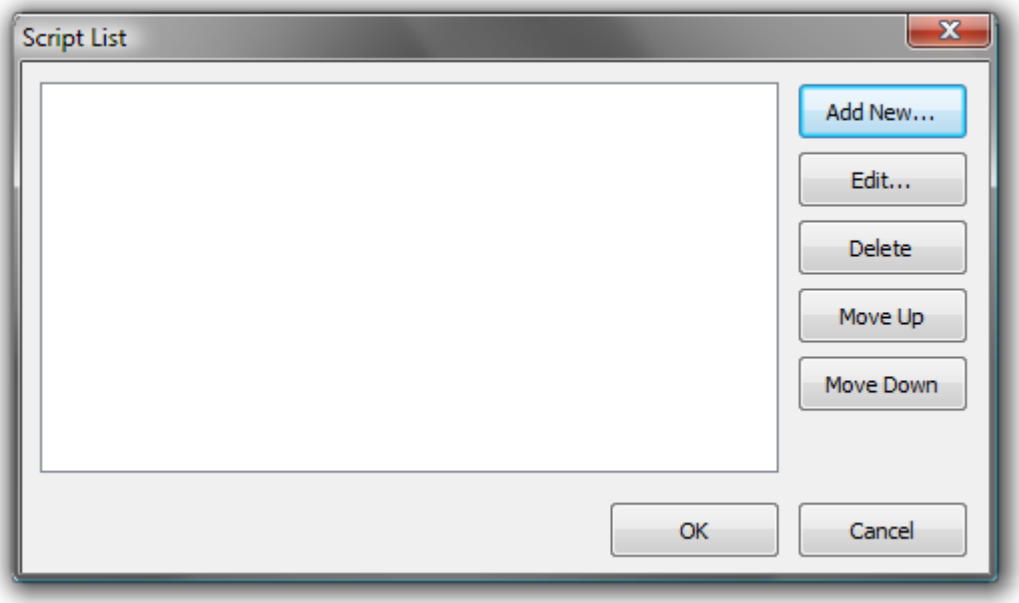
```
                m_nInitScenario = m_nInitScenario + 1;
            }
        }
        else
        if(in.Stream == 5 && in.Function == 4)
        {
            // S5F4
            if(m_nInitScenario == 8)
            {
                out.SML = "s2f15w{}";
                Send();
                m_nInitScenario = m_nInitScenario + 1;
            }
        }
        else
        if(in.Stream == 2 && in.Function == 16)
        {
            // S2F16
            if(m_nInitScenario == 9)
            {
                out.SML = "s1f3w{}";
                Send();
                m_nInitScenario = m_nInitScenario + 1;
            }
        }
        else
        if(in.Stream == 1 && in.Function == 4)
        {
            // S1F4
            if(m_nInitScenario == 10)
            {
                out.SML = "s2f37w{<bool true>{}}";
                Send();
                m_nInitScenario = m_nInitScenario + 1;
            }
        }
}
```

1.18.7 新規スクリプトの追加

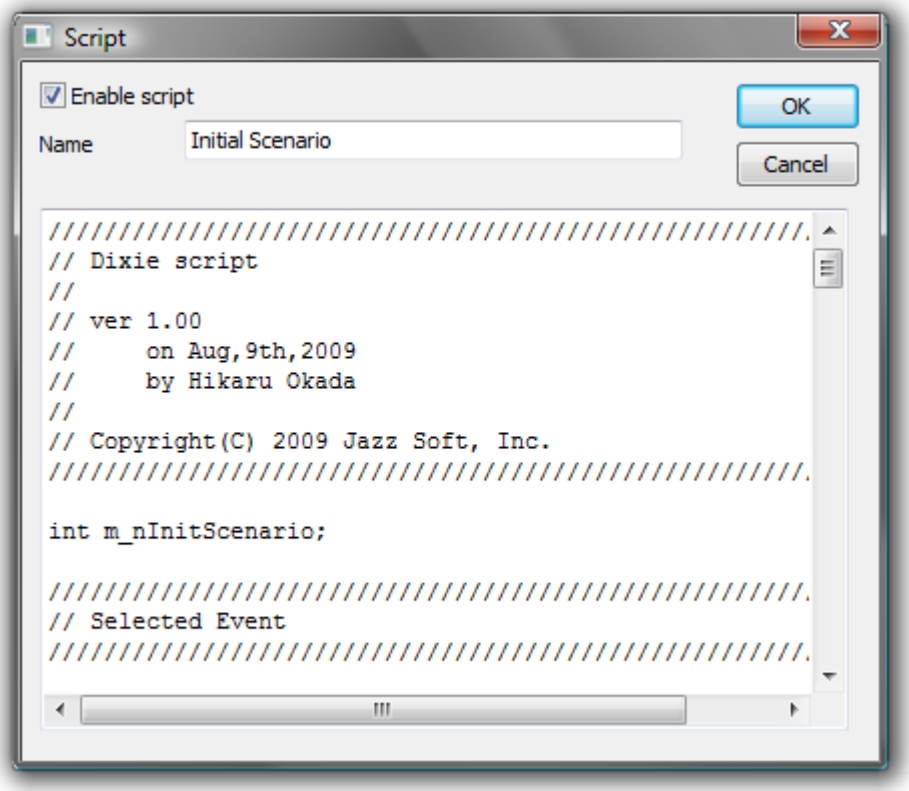
上記のスクリプトを Visual Studio 2008 などのテキストエディタを使用して入力し、Dixie のスクリプト編集画面でペーストするのが簡単です。テキストエディタとしてはメモ帳などを使用することもできますし、何もなければ Dixie のスクリプト編集画面で手入力しても構いません。

1 Dixie を起動し、リボンの Home – Communication から Script…ボタンをクリックします。

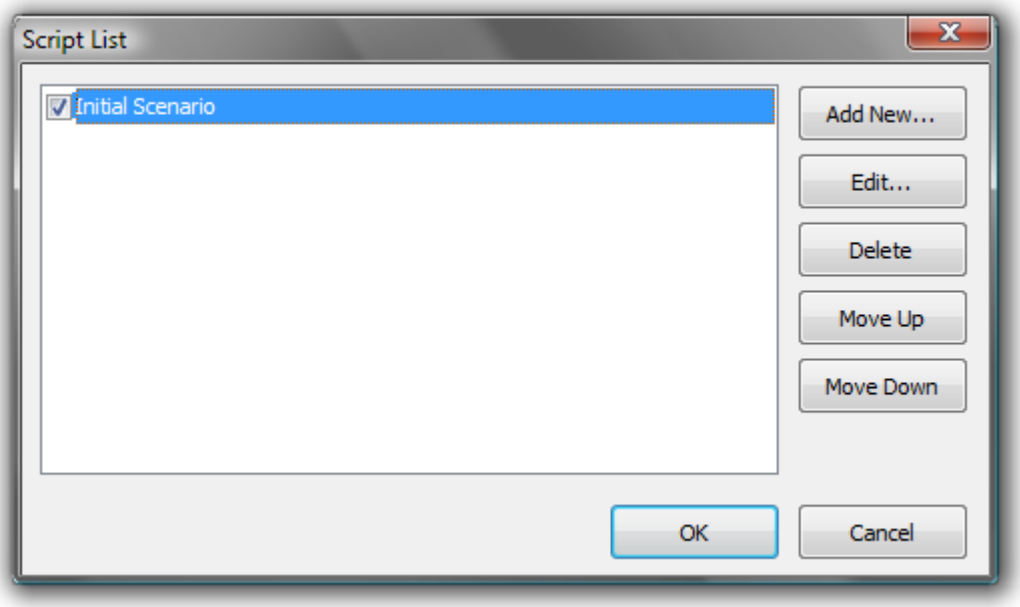
2 Add New…ボタンをクリックします。



3 スクリプトを入力し、適当な名前をつけて OK ボタンをクリックします。



4 新規のスクリプトが作成されました。



1.18.8 動作検証

スクリプトが正常に動作するかどうかを調べるには実際に装置につないでテストしてみるのが確実です。ただし Dixie を使って一方をホストシミュレータ、他方を装置シミュレータに設定してテストすることも可能です。

8. スクリプト仕様

スクリプトは、Dixie を制御するためのオブジェクト指向の簡易言語です。

1.19 数値

数値はすべて整数です。十進法表現、「0x」で始まる 16 進法表現、「true」(=1)、「false」(=0)が使用可能です。

例

Dixie Script

```
out.Stream = 1;
out.Function = 0x0d;
out.Wbit = true;
```

1.20 文字列

文字列はダブルクォーテーションで囲まれたリテラル文字の集合です。文字列は連結することが可能です。

例

Dixie Script

```
out.Sml =
    “s1f13w”
    “{“
    “    <b 0>”
    “    {“
    “    }”
    “}”
    ;
```

1.21 コメント

Dixie スクリプトのコメントは C++言語とほぼ同じです。コメントには日本語を記述することができます。

1.21.1 //

「//」から行末まではコメントとなります。

Dixie Script

```
// HSMS connection has been selected
```

1.21.2 /* */

「/*」から「*/」まではコメントとなります。

Dixie Script

```
/*
    HSMS connection has been selected
*/
```

1.22 イベント関数

ある特定の条件を満たすと、Dixie はイベントを発生します。スクリプト中にイベント関数を定義しておくと、Dixie に指定した動作をさせることができます。

1.22.1 OnSelected()

HSMS のセレクト要求に対して正常な応答が返ってきた場合に発生します。

文法

Dixie Script

```
bool OnSelected( bEnabled )
```

bEnabled の値は下記のいずれかとなります。

値	説明
true	関数が有効
false	関数が無効
なし	true と同じ

関数が無効に設定されている場合、スクリプトは実行されません。

例

Dixie Script

```
bool OnSelected()
{
    cout << "OnSelected()";

    // S1F13
    out.Sml="s1f13w{<a'Script'><a'1.00'>}";
    Send();
    return true;
}
```

特記事項

参照

1.22.2 OnReceived()

データメッセージ（S タイプが 0 のメッセージ）を受信した場合に発生します。

文法

Dixie Script

```
bool OnReceived( bEnabled )
```

bEnabled の値は下記のいずれかとなります。

値	説明
true	関数が有効
false	関数が無効
なし	True と同じ

関数が無効に設定されている場合、スクリプトは実行されません。

例

Dixie Script

```
bool OnReceived()
{
	cout << "OnReceived()";

	if(in.Stream==1)
		if(in.Function==1)
			cout << "stream is 1 and function is also 1";
		else
			cout << "stream is 1 but function is not 1";
	else
		if(in.Function==1)
			cout << "stream is not 1 but function is 1";
		else
			cout << "stream is not 1 and function is also not 1";
}
```

特記事項

参照

1.22.3 OnProblem()

エラーが起こった場合に発生します。

文法

Dixie Script

```
bool OnProblem( bEnabled )
```

bEnabled の値は下記のいずれかとなります。

値	説明
true	関数が有効
false	関数が無効
なし	true と同じ

関数が無効に設定されている場合、スクリプトは実行されません。

例

Dixie Script

```
bool OnProblem()
```

```
{
    cout << "OnProblem()";

    return false;
}
```

特記事項

参照

1.23 組み込み関数

Dixie スクリプトには、あらかじめ組み込まれている命令があります。

1.23.1 Send()

メッセージを送信します。

文法

Dixie Script

```
void Send( string strMessageName )
```

strMessageName は、あらかじめ登録されているメッセージの名前です。省略した場合は、現在 out オブジェクトが保持している内容が送信されます。

例

Dixie Script

特記事項

参照

1.23.2 Reply()

メッセージを返信します。ヘッダ情報は in オブジェクトの二次メッセージとして更新されます。

文法

Dixie Script

```
void Reply( string strMessageName )
```

strMessageName は、あらかじめ登録されているメッセージの名前です。省略した場合は、現在 out オブジェクトが保持している内容が送信されます。

例

Dixie Script

特記事項

参照

1.24 変数型

変数の型には int 型と string 型があります。代入や比較の際には、演算子の左右で型が一致している必要があります。

1.25 組み込みオブジェクト

いくつかのオブジェクトが組み込まれています。

1.25.1 in

入力メッセージを処理する SecsII オブジェクトです。

整数系

Member	Description
BlockNumber	ブロック番号。
DeviceID	デバイス ID。
Function	ファンクション番号。
NodeCount	ノードの個数。読み出し専用。
NodeType	ノードのタイプ。読み出し専用。
Ptype	P タイプ。
Stype	S タイプ。
SessionID	セッション ID。
SourceID	ソース ID。
Stream	ストリーム番号。
SystemBytes	システムバイト。セッション ID とトランザクション ID を連結したもの。
TransactionID	トランザクション ID。

ブーリアン系

Member	Description
Ebit	エンドビット。
Rbit	リバースビット。
Wbit	ウェイトビット。

文字列系

Member	Description
Msg	メッセージの 16 進法表現。
Node	ノード指定子。
NodeValue	ノードの値。読み出し専用。
NodeValueHex	ノードの値の 16 進法表現。読み出し専用。
Sml	メッセージの SML 表現。
Suggested	推奨される返信メッセージの 16 進法表現。読み出し専用。
Xml	メッセージの XML 表現。現在は未対応。

1.25.2 out

出力メッセージを処理する SecsII オブジェクトです。メンバ変数は in オブジェクトと同じです。

1.25.3 arg

イベント関数に渡される引数です。

メンバ

Member	Description
IPAddress	相手の IP アドレスの文字列です。
PortNumber	相手のポート番号をあらわす整数です。
ErrorCode	エラーコードをあらわす整数です。OnProblem()イベントでのみ有効です。
AdditionalInfo	追加情報の文字列です。OnProblem()イベントの一部のエラーコードでのみ有効です。

例

Dixie Script

```
cout << "IP Address:          " + arg.IPAddress;
cout << "Port Number:         " + arg.PortNumber;
cout << "Error Code:          " + arg.ErrorCode;
cout << "Additional Info: " + arg.AdditionalInfo;
```

1.25.4 cout

画面の output 欄に文字列を表示します。

文法

Dixie Script

```
void cout::operator =( string strText )
```

例

Dixie Script

```
cout << "This is a sample.";
```

1.26 ユーザ定義変数

ユーザが自由に変数を定義することができます。変数名は「m_」で始める必要があります。

1.26.1 int 型

符号付き整数型です。

文法

Dixie Script

```
```

```
int VariableName ;
```

例

```
Dixie Script
int m_nTest;
```

1.26.2 string 型

文字列型です。

文法

```
Dixie Script
string VariableName ;
```

例

```
Dixie Script
string m_strTest;
```

1.27 ステートメント

ステートメントとはスクリプトを構成する命令セットです。

1.27.1 if

if 文は C/C++と同じように if に続けて条件式を記述し、その後ステートメントが続きます。条件に一致しない場合の処理として else ブロックを記述することも可能です。

文法

```
Dixie Script
if ( Condition )
    Statement
```

```
Dixie Script
if ( Condition )
    Statement
else
    Statement
```

```
Dixie Script
if ( Condition )
{
    Statements
}
```

Dixie Script

```
if ( Condition )
{
    Statements
}
else
{
    Statements
}
```

例

Dixie Script

```
    if(in.Stream==1)
    {
        // 条件に一致した場合の処理
    }

    if(in.Stream==1)
    {
        // 条件に一致した場合の処理
    }
    else
    {
        // 条件に一致しなかった場合の処理
    }
```

ステートメントが一つだけの場合は、ステートメントブロックを{ }で囲むのを省略することができます。

例

Dixie Script

```
    if(in.Stream==1)
        if(in.Function==1)
            Reply(“s1f2”);
```

else は直前の if ステートメントに結びつきます。

例

Dixie Script

```
bool OnReceived()
{
    cout << "OnReceived()";

    if(in.Stream==1) ---------------------------------------------- (A)
        if(in.Function==1) ---------------------------------------- (B)

        else ------------------------------------------------------- (1)

    else ------------------------------------------------------------ (2)
        if(in.Function==1) ---------------------------------------- (C)
```

```
        else ---------------------------------------------------------- (3)
}
```

上記の例では 3 回 else が登場しますが、それぞれ(1)は(B)、(2)は(A)、(3)は(C)に対応します。

1.27.2 return

return はスクリプト処理を抜けて Dixie に制御を戻す場合に使用します。戻り値として true または false を返すことができます。

文法

Dixie Script

return *ReturnValue* ;

例

Dixie Script

```
return true;
return false;
```

1.27.3 代入

代入とは組み込みオブジェクトのメンバ変数、またはユーザ定義変数に値をセットすることです。記号としては=を使用します。代入の左辺（被代入側）と右辺（代入側）では、型が一致していなければなりません。

文法

Dixie Script

Receptor = *Nominator* ;

例

Dixie Script

```
out.Stream = 3;
out.Sml = “s1f1w”;
```

1.27.4 その他

組み込み関数や組み込みオブジェクトをステートメントに記述することができます。

例

Dixie Script

```
Reply(“s1f2”);
cout << "This is a sample.";
```

1.28 演算子

演算子は基本的に左結合で、以下のものが定義されています。演算子の優先順位の低いものから順に記述してあります。

Operator	Description
=	代入
&&	AND 条件
||	OR 条件
==	一致条件
!=	不一致条件
>	大小比較条件
>=	大小比較条件
<	大小比較条件
<=	大小比較条件
+	加算
-	減算
*	乗算
/	除算
%	剰余
&	論理 AND
|	論理 OR
-	マイナス符号

演算子の優先順位規則に基づいて処理が実行されます。

例

Dixie Script

```
out.Function = 1 + 2 * 3 + 4;
```

この例では「=」「+」「*」の 3 種類の演算子が登場します。優先順位が明示的に分かるよう、以下のように記述しても同じです。

例

Dixie Script

```
out.Function = ( 1 + ( 2 * 3 ) + 4 );
```