ＰＳＬＸプラットフォーム計画

C#版

共通コンポーネント実装マニュアル

第 3 部

PPS メッセージングサービス

バージョン 1.0

2009 年 6 月

NPO 法人ものづくり APS 推進機構

改訂履歴

日付	内容	備考
2009/03/12	バージョン 1.0 ベータ版	
2009/06/07	バージョン 1.0	

もくじ

1. はじめに

◆ 目的

PPS 共通コンポーネントは、PSLX プラットフォーム対応のソフトウェア構築において、異なるアプリケーションプログラムおよびソフトウェア環境で相互接続性を保証するための共通の実装環境です。PPS 共通コンポーネントは、OASIS PPS 技術委員会が定めた国際標準に沿って実装されています。またメッセージ通信に関する詳細な知識および XML に関する詳細な知識を必要とせずに容易に PSLX プラットフォーム対応のシステム構築が可能となるように設計されています。

本マニュアルは、実装マニュアル第 3 部「PPS メッセージングサービス C#版」です。本マニュアルは、.NET Framework における PSLX プラットフォームによる業務アプリケーション間のメッセージの送受信に関するプログラミングを支援するためのコンポーネントの利用方法を解説します。メッセージキューを利用したメッセージの送受信や、Web サーバを利用した SOAP および HTTP／POST を利用した送受信を行うプログラムの実装方法について解説します。

◆ 対象とする読者

（1） 資格

この仕様書は、PSLX プラットフォーム計画プロジェクトに参加している企業の従業員に対して、PSLX プラットフォーム対応ソフトウェアを開発するために公開している文章です。PSLX プラットフォーム計画プロジェクトのメンバー以外であっても、この仕様書を閲覧することは可能ですが、NPO 法人ものづくり APS 推進機構の許可なく複製や再配布を行うことは禁止されています。

（2） 必要とする知識・技術

ソフトウェア開発の一般知識を有する人を対象にした文章です。特に、下記の項目についての知識が必要です。

— C#によるプログラミング
— Visual Studio によるソフトウェア開発

◆ 稼動環境

（1） 必要ソフトウェア環境

区分	内容
オペレーティングシステム	Windows XP Service Pack 3 以降 Windows 2003 Server 以降
コンポーネント	.NET Framework 2.0 以降
Web ブラウザ	Microsoft Internet Explorer 6,7 および 8 Mozilla Firefox 3.0
開発ツール	Visual Studio 2005 ServicePack1、または Visual Studio 2008

（2） 実行環境

本マニュアルに記載されているすべてのプログラムを実行するためには、次のメッセージキューサーバのいずれか 1 つが、実行するコンピュータか、あるいは実行するコンピュータから LAN 経由でアクセス可能な他のコンピュータ上にインストールされている必要があります。メッセージキューサーバのインストール方法については、次の各項目に対応した URL を参照してください。

名称	開発元	URL
Apache ActiveMQ	Apache	http://activemq.apache.org/
WebSphere MQ	IBM	http://www.ibm.com/developerworks/jp/websphere/category/wmq/
MSMQ	Microsoft	http://msdn.microsoft.com/ja-jp/msmq/

2. 通信方式の基本概念

◆ さまざまな通信方式の対応

PPS メッセージングサービスは、業務アプリケーション間での情報のやりとりを実際に行うためにさまざまな通信手段と業務アプリケーションとをつなぐための共通コンポーネントです。業務アプリケーション間との通信手段としては、SOAP やメッセージキューなどが挙げられ、より単純な手段としては HTTP や SMTP や FTP などが挙げられます。

PPS メッセージングサービスを利用することで、下位の通信手段を意識することなく、業務アプリケーション間での情報のやりとりを行うアプリケーションプログラムの開発が可能となります。下位の通信手段が変更になった場合でも、プログラムの作り直しを避けることができます。

通信手段としてメッセージキューサーバを利用することで、業務アプリケーション間をピア・ツー・ピア方式で通信することが可能です。また通信手段として SOAP や HTTP/POST を利用する場合は、サーバ・クライアント方式で通信することが可能です。SOAP や HTTP/POST を利用するためには、ASP.NET(IIS)や Servlet/Axis(Tomcat)、PHP などのアプリケーションサーバを利用します。

◆ 送信と受信

PPS メッセージサービスを提供する共通コンポーネントとして、Pps.Messaging.dll が容易されています。これは、現時点では、メッセージキュー（MQ）や SOAP、HTTP/POST 等の通信手段によって、送信者から受信者へテキストデータを送信するためのものです。メッセージキューを利用する場合には、送信者は、メッセージキューサーバにあらかじめ割り当てられたキューの名称を指定し、そこにメッセージを送信します。一方、受信者は、キューの名称を指定し、そこにある最も古いメッセージから順に取り出します。この結果、メッセージキューサーバを経由して、送信者から受信者へメッセージが送信されることになります。なお、メッセージキューの特徴として、キュー読み出しは、書き込んだ順となります。また一度キューから読み出しがら、その時点でキューには該当データが存在しなくなります。

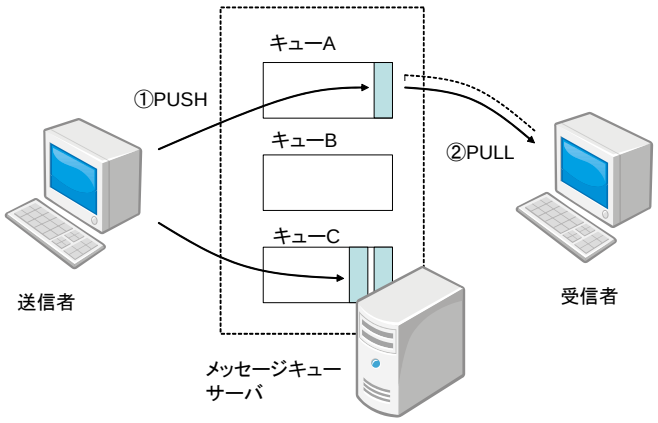


図３－１　メッセージキューの基本機能

◆　トランザクションの構造

メッセージキューを利用した通信は、基本的には非同期で行われます。つまり、送信側が送信したタイミングと受信側が受信したタイミングはずれがあり、送信側は、受信側がメッセージを受信したかどうかにかかわらず次の処理を行うことが可能です。

一方で、業務情報の照会を行う場合や、受信確認をおこないたい場合など、受信者からの返信メッセージを、送信者へ返すようなトランザクションも存在します。このような場合には、次の図のように、メッセージキューサーバを介して、往信メッセージと返信メッセージが交換されます。

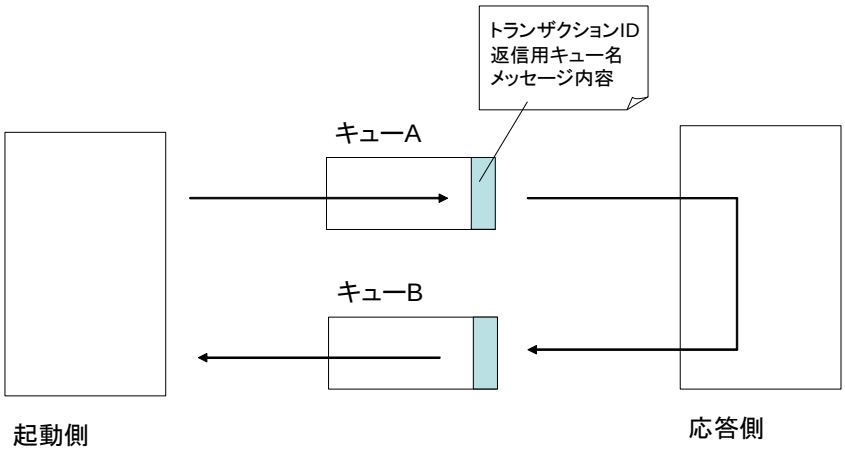


図 1 トランザクションの構造

ここで、起動側（クライアント）は、送信メッセージに、トランザクション ID と、自身

が管理している返信用キュー名を、送信内容と合わせて設定し、応答側が管理しているキューに送信します。応答側（サーバ）は、受信したメッセージに返信が必要なものがある場合には、そこに設定している返信用キューに対して返信メッセージを送信します。この際に、返信メッセージのトランザクション ID は、受信メッセージのトランザクション ID と同一にします。起動側は、当初のメッセージ送信後、返信用キューを監視し、そこに到着したメッセージをしらべ、送信メッセージのトランザクション ID と同じ返信メッセージがあれば、その内容を送信メッセージと対応づけ、応答メッセージにある内容を処理します。

これらの一連の処理は、共通コンポーネントによって実施され、業務アプリケーション・プログラマは、それらをほとんど認識せずにプログラミングを行うことができます。

3. 必要なファイルとモジュール構成

◆ 必要なファイル

本マニュアルで解説する機能を実行するためには、次の実行ファイルおよび定義ファイルが必要となります。

（1） 実行ファイル

ファイル名	説明
Pps.Messaging.dll	この仕様書のプログラムを実行するために必要な共通コンポーネントです。
Pps.Messaging.SoapService.dll	SOAP クライアントとして WWW サーバにアクセスするためのコンポーネントです。
Pps.Messaging.HttpService.dll	HTTP/POST によるクライアントとして WWW サーバにアクセスするためのコンポーネントです。
Pps.Messaging.ActiveMQ.dll	Apache ActiveMQ サーバを利用する場合に必要となるコンポーネントです。
Pps.Messaging.WebSphereMQ.dll	IBM WebSphere MQ サーバを利用する場合に必要となるコンポーネントです。
Pps.Messaging.MSMQ.dll	Microsoft MSMQ サーバを利用する場合に必要となるコンポーネントです。
Apache.NMS.dll	Apache から提供されている実行モジュールです。
Apache.NMS.ActiveMQ.dll	Apache から提供されている実行モジュールです。
amqmdnet.dll	WebSphereMQ 用に IBM 社から提供されている実行モジュールです。
amqmdxcs.dll	WebSphereMQ 用に IBM 社から提供されている実行モジュールです。

（2） 定義ファイル

ファイル名	説明
Pps.Messaging.config	この仕様書のプログラムを実行するために必要な定義ファイルです。
Pps.Messaging.config.SoapService	SOAP クライアントを実装する場合に必要となる

	設定ファイルです。ファイル名を Pps.Messaging.config に変更して利用します。
Pps.Messaging.config.HttpService	HTTP/POST を用いた Web クライアントを実装する場合に必要となる設定ファイルです。ファイル名を Pps.Messaging.config に変更して利用します。
Pps.Messaging.config.ActiveMQ	Apache ActiveMQ サーバを利用する場合に必要となる設定ファイルです。ファイル名を Pps.Messaging.config に変更して利用します。
Pps.Messaging.config.WebSphereMQ	IBM WebSphere MQ サーバを利用する場合に必要となる設定ファイルです。ファイル名を Pps.Messaging.config に変更して利用します。
Pps.Messaging.config.MSMQ	Microsoft MSMQ サーバを利用する場合に必要となる設定ファイルです。ファイル名を Pps.Messaging.config に変更して利用します。

◆ モジュール構成

PPS メッセージングサービス(以下、本コンポーネント)を業務アプリケーションで利用する場合は、利用する通信手段によって、モジュール構成が異なります。どの通信手段を利用するかは、定義ファイルによって指定します。

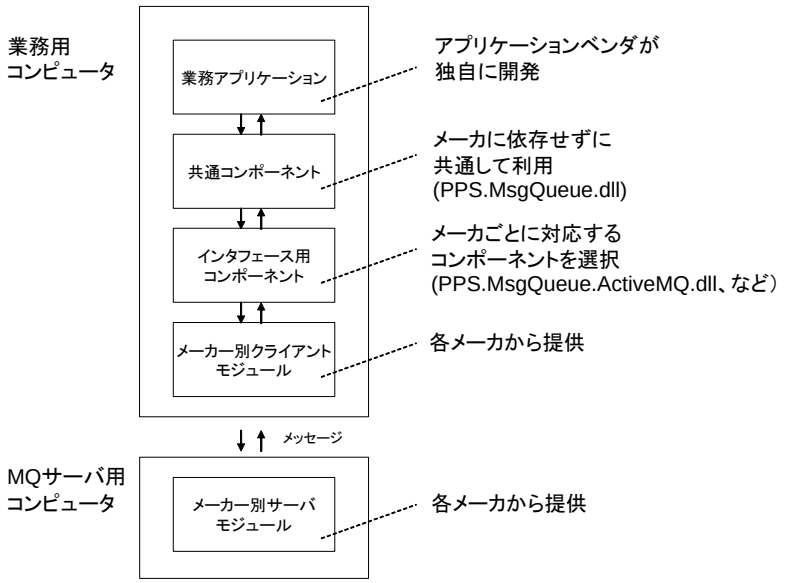


図 2 共通コンポーネントを利用した業務アプリケーションの構成

◆ 定義ファイルの設定

本コンポーネントの定義ファイルは、Pps.Messaging.config というファイル名で、業務アプリケーションの実行ファイルが存在するフォルダにおく必要があります。定義ファイルの内容は、次のような形式で書きます。

```
<?xml version="1.0" encoding="utf-8"?>
<Settings>
    <QueueEngine>
        <QueueEngineAssembly>Pps.Messaging.ActiveMQ.dll</QueueEngineAssembly>
        <QueueEngineClassName>Pps.Messaging.ActiveMQ</QueueEngineClassName>
        <Location>tcp://localhost:61616</Location>
    </QueueEngine>
</Settings>
```

ここで、各要素は次のように指定してください。なお、Apache ActiveMQ、IBM WebSphereMQ、MSMQ、そして Web 対応として SOAP や HTTP/POST それぞれについて、次の値を設定したファイルがそれぞれ Pps.Messaging.config.ActiveMQ、Pps.Messaging.config.WebSphereMQ、Pps.Messaging.config.MSMQ、Pps.Messaging.config.SoapService、Pps.Messaging.config.HttpService というファイル名称で用意されています。これらのファイル名称を Pps.Messaging.config に変更することで、次の形式となります。

（1） ConnectorEngineAssembly 要素

ConnectorEngineAssembly 要素には、次の表のように、メッセージキューサーバを提供によって異なるインタフェースコンポーネントのファイル名を指定します。これは、固定値です。ここで指定したファイルは、業務アプリケーションの実行ファイルがあるフォルダに置く必要があります。

サーバ種類	インタフェースコンポーネント名	備考
Apache ActiveMQ	Pps.Messaging.ActiveMQ.dll	固定値です。
IBM WebSphere MQ	Pps.Messaging.WebSphereMQ.dll	固定値です。
Microsoft MSMQ	Pps.Messaging.MSMQ.dll	固定値です。

SOAP サーバ	Pps.Messaging.SoapService.dll	固定値です。
HTTP サーバ（POST）	Pps.Messaging.HttpService.dll	固定値です。

（２） ConnectorEngineClassName 要素

ConnectorEngineClassName 要素には、次の表のように、メッセージキューサーバによって異なるインタフェースコンポーネントのクラス名を指定します。これは、固定値です。

サーバ種類	値	備考
Apache ActiveMQ	Pps.Messaging.ActiveMQ	固定値です。
IBM WebSphere MQ	Pps.Messaging.WebSphereMQ	固定値です。
Microsoft MSMQ	Pps.Messaging.MSMQ	固定値です。
SOAP サーバ	Pps.Messaging.SoapService	固定値です。
HTTP サーバ（POST）	Pps.Messaging.HttpService	固定値です。

（３） Location 要素

Location 要素には、メッセージキューサーバの位置を指定します。指定する方法は、メッセージキューサーバによって異なります。

サーバ種類	値（例）	備考
Apache ActiveMQ	tcp://localhost:61616	設定例です。
IBM WebSphere MQ	localhost:1414¥TestQM	設定例です。
Microsoft MSMQ	.¥Private$	設定例です。
SOAP サーバ	http://localhost:8080/PpsMessaging.asmx	設定例です。
HTTP サーバ（POST）	http://localhost/Default.aspx	設定例です。

サーバ種類が ActiveMQ の場合、上記のように、サーバマシンの IP アドレスおよび MQ サーバのポート番号を指定します。記述方法は、“tcp://”が固定値で、その後の“:”までにサーバの IP アドレスを指定し、最後にポート番号を指定します。

サーバ種類が WebSphere MQ の場合、あらかじめ定義したサーバ名を指定します。もしキューマネージャがローカルコンピュータに存在する場合には、ここにはキューマネージ

ャ名を指定します。キューマネージャがリモートコンピュータに存在する場合には、“サーバの IP アドレス”＋“：”＋ポート番号＋“：”＋チャネル名＋“¥”＋キューマネージャ名とします。なお、チャネル名、あるいはポート番号とチャネル名を省略することが可能です。省略値は、ポート番号は 1414、チャネル名は“SYSTEM.DEF.SVRCONN”となります。

サーバ種類が MSMQ の場合、サーバマシン名を指定します。文字列“.¥”はローカルコンピュータを表します。なお、ローカルコンピュータにおけるプライベートなキューを利用する場合には、上記の MSMQ の設定例のように、“Private$”を付加します。この場合には、外部のコンピュータからのアクセスはできません。MSMQ を利用してリモートのキューにアクセスするためには、Windows ドメインコントローラの管理下の中で、あらかじめサーバ名が登録されている必要があります。

Web サーバの場合として､SOAP サーバの場合と通常の HTTP サーバの場合については、それぞれ該当する URL および該当サービスのあるアドレスを指定します。

4. メッセージの送信と受信

◆ 準備作業

本コンポーネントに対する業務アプリケーションとの API は、Pps.Messaging.dll によって提供されています。まず、プロジェクトの参照設定において、“参照の追加”を選択し、“参照”タブの中で、このコンポーネントを選択します。また、次のように、using の設定を各プログラムファイルの先頭で指定します。

```
using Pps.Messaging;
```

また、PPS メッセージサービスのマネージャクラス（MessageManager）のオブジェクトを定義します。定義位置は、メッセージの送信、受信等のメソッドから利用可能なように、共通の変数領域にするとよいでしょう。

```
// PPSメッセージマネージャの定義
MessageManager manager;
```

業務プログラムの初期化の処理の中で、次のように、マネージャオブジェクトを生成するとともに、必要なプロパティを設定します。ここでは、業務アプリケーションの名称およびメッセージキューサーバのアドレスを指定しています。メッセージキューサーバのアドレスの指定方法は、メッセージキューサーバを提供するメーカによって異なりますので、前章を参照してください。なお、設定しない場合には、設定ファイルにある記述がそのまま使用されます。

```
// PPSメッセージマネージャを生成します。
manager = new MessageManager();
manager.SenderId = "HOSEI";
manager.Location = "tcp://localhost:61616";
```


◆ メッセージの送信

メッセージを送信するためには、SendTextMessage メソッドを利用します。次のように、このメソッドに、送信するテキストおよび送信先キューの名称を指定して実行してください。

```
try // 送信します。エラーの場合はその内容を表示します。
{
    manager.SendTextMessage(message, queueName);
    MessageBox.Show("送信しました。");
}
catch(Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

また、次のように、送信メッセージのオブジェクトを生成して、それを SendMessage によって送信することも可能です。この場合には、次のようにメッセージオブジェクトを生成するときにメッセージ内容を指定してください。

```
    TransportMessage msg = manager.CreateMessage(message);
    manager.SendMessage(msg, queueName);
```

◆ メッセージの受信

メッセージの受信は、送信者からではなく、メッセージキューサーバから受け取ります。同期メッセージ受信の場合には、受信側プログラムは、次のように、自らサーバのキューを指定してそこにあるメッセージを受け取ります。メッセージを読み込んだ時点で、対象となるキューからは、そのメッセージが取り除かれます。

```
try // 受信キューからメッセージを読み込みます。
{
    string msg = manager.ReceiveTextMessage(queueName);
    Message_richTextBox.Text = msg;
}
catch (Exception ex)
```



```
    {
        MessageBox.Show(ex.Message);
    }
```

この場合、もし指定したキューにメッセージが存在している場合には、その先頭のメッセージの内容が戻り値として設定されます。もし、メッセージがキューにひとつも存在しない場合には、戻り値に null が返されます。

送信の場合と同様に、ReceiveTextMessage メソッドは、ReceiveMessage によって次のように代替することができます。この場合には、戻り値であるメッセージオブジェクトの Message プロパティに内容が設定されています。もし、キューにメッセージが存在しない場合には、ReceiveMessage メソッドの戻り値は null となります。

```
    TransportMessage msg = manager.ReceiveMessage(queueName);
    if(msg!=null) Message_richTextBox.Text = msg.Message;
```

あらかじめ、キューの中にメッセージが存在しているかどうかを確認するためには、次のように ReceiveCount メソッドを利用します。このメソッドは、その時点で、指定したキューにあるメッセージの数を返します。

```
    if (manager.ReceiveCount(queueName) > 0)
    {
        string message = manager.ReceiveTextMessage(queueName);
        Message_richTextBox.Text = message;
    }
```

ReceiveMessage、ReceiveTextMessage、ReceiveCount の各メソッドは、受信キューを毎回指定する方法と、あらかじめ設定してあるキュー名を利用する場合があります。後者の場合には、マネージャクラスの QueueName プロパティに受信キュー名を設定し、上記３つのメソッドではキュー名を省略することができます。

◆ 非同期のメッセージ受信

メッセージを受信する方法として、メッセージが到着したかどうかを毎回確認する方法ではなく、メッセージが到着した場合に、非同期でそのメッセージが読み込まれるようなプログラムを作成することができます。そのためには、次のように、まずレシーバクラスのオブジェクトを生成します。ただし、ここで指定する非同期受信用のキュー名は、マネージャオブジェクトの ConnectorName プロパティとは同一にできません。そして、そのオブジェクト（受信キューオブジェクト）の ReceiveMessage イベントに対してコールバックメソッドを設定します。次の例では、msgQueue_ReceiveMessage メソッドが定義されています。このように ReceiveMessage イベントでイベントハンドラを設定することで、メッセージを受信するとすぐに、このメソッドが実行されるようになります。

```
// PPSメッセージサービスマネージャを設定します。
manager = new MessageManager();
manager.SynchronizingObject = this;

// 受信キューを生成します。
msgQueue = manager.CreateReceiver(queueName);
msgQueue.ReceiveMessage += new ReceiveMessageHandler(msgQueue_ReceiveMessage);
```

msgQueue_ReceiveMessage メソッドの引数は、ReceiveMessageHandler デリゲートによって次のように定義されています。ここで、最初の引数はメッセージを受信した受信キューであり、2 つ目の引数がメッセージオブジェクトになります。したがって、次のように、このメッセージオブジェクトの Message プロパティを取得することで、受信したメッセージの内容を確認することができます。

```
void msgQueue_ReceiveMessage(MessageConnector sender, TransportMessage m)
{
    // このメソッドは、メッセージを受信した時点で呼ばれるコールバックメソッド

    Console.Beep();
    Message_richTextBox.Text = m.Message;
}
```

このように、受け取ったメッセージを非同期で処理する場合には、この例のように、マ

ネージャクラスの SynchronizingObject の値に、表示のために利用する Windows の Form クラスまたはそのサブクラスのオブジェクトを指定してください。これによって、別のスレッドで実行されている受信処理と、業務アプリケーションの処理とが同期可能となります。

```
try // 受信キューの読込を開始
{
    msgQueue.Open();
    this.Text = "受信プログラム（起動）";
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

非同期メッセージの受信は、受信キューオブジェクトの Open メソッドを実行するまで開始されません。また、受信キューをオープンした場合には、必ずプログラム終了時にクローズしてください。受信キューオブジェクトの IsConnected プロパティの値が true の場合には、すでにオープンしている状態なので、次のように Close メソッドを利用してクローズします。

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    // 終了時に受信キューをクローズする
    if (msgQueue.IsConnected) msgQueue.Close();
}
```


5. クライアントとサーバの連携

◆ クライアントの処理

送信および受信を行うシンプルなプログラムに対して、クライアントとサーバのプログラムは、多少複雑になります。そこでは、シンプルな送信および受信の組み合わせによって、メッセージが一往復することになります。まず、クライアントでは、まずなんらかの依頼メッセージをサーバが管理するキューに送信します。そして、こちらから指定したキューに対するサーバからの返信を待ち、そこにメッセージが到着した後、そのメッセージが依頼したメッセージの返信であるかを確認して、もしそうであれば終了します。

このような一連のクライアント処理は、本コンポーネントの SyncMessage メソッドあるいは SyncTextMessage メソッドを一度呼び出すだけで行うことが可能です。SyncTextMessage メソッドを使った同期処理によるメッセージ送受信を行うプログラムを次に示します。

```
try // メッセージを送信し回答を待ってすぐに受信画面に表示します。
{
    string result = manager.SyncTextMessage(message, sendQueueName);
    Receive_richTextBox.Text = result;
}
catch (TimeoutException ex) // エラー（タイムアウト）の場合
{
    MessageBox.Show(ex.Message);
    Receive_richTextBox.Text = "";
}
```

このプログラムで、MessageManager オブジェクトの SyncTextMessage メソッドを呼び出すことによって、サーバへメッセージを送信しています。最初の引数（message）が送信メッセージを表し、2 つ目の引数が送信先のキューの名称を表します。サーバから返される返信内容は、SyncTextMessage メソッドの戻り値（result）として返されます。

SyncMessage メソッドあるいは SyncTextMessage メソッドによるクライアント処理は、同期して行われるため、これらメソッドが呼び出されると、サーバから返信があるまで待ち、その間は制御が戻りません。サーバから一定時間以上経っても返信メッセージが受信できない場合には、タイムアウトの例外が発生します。

タイムアウトの時間は、次のように、SyncTimeout プロパティで設定することができます。単位はミリ秒です。また、SamplingInterval プロパティは、タイムアウトになるまでの間、受信があるかどうかについて受信キューを確認する間隔を指定することができます。この値は、タイムアウト時間よりも短くしなければなりません。

```
manager.SyncTimeout = 3000;
manager.SamplingInterval = 500;
```

◆ 非同期クライアント

クライアント処理を非同期で行いたい場合には、次のように、まず受信用のキューの監視を開始させた後に、送信メッセージを送ります。受信メッセージを受け取った場合には、そのメッセージが送信メッセージの返信なのかどうかを、トランスポート ID によって確認します。ここで、トランスポート ID とは、送信メッセージとそれに対応する返信メッセージを対応づけるものであり、サーバは返信時に、送信メッセージに設定されているトランスポート ID を返信メッセージに設定します。

```
// 送信したメッセージのトランスポートID
string TransportId;

// 受信用キュー
MessageConnector receiveQueue;
```

トランスポート ID と受信用キューを共通領域に定義した後に、次のように、ます受信用キューを生成します。受信用キューの名称を設定しない場合には、MessageManager クラスの ConnectorName プロパティで設定された名称の受信キューが使われます。次のプログラムは、受信用キューの監視を開始するプログラムです。

```
// 非同期で結果を受信するためのキューを設定します。
receiveQueue = manager.CreateReceiver();
receiveQueue.ReceiveMessage += new
    ReceiveMessageHandler(receiveQueue_ReceiveMessage);
```


上記のコールバックメソッドである receiveQueue_ReceiveMessage メソッドのプログラムは、次のようになります。このメソッドでは、受信したメッセージのトランスポート ID の照合が行われています。なお、コールバックメソッドを受け取るためには、MessageManager クラスの SynchronizingObject プロパティを設定する必要があります。

```
void receiveQueue_ReceiveMessage(MessageConnector sender,
    TransportMessage message)
{
    if (message.TransportId == TransportId)
    {
        // 返信メッセージの処理を記述します。
        Receive_richTextBox.Text = message.Message;
    }
    else
    {
        // その他の受信メッセージの処理を記述します。
    }
}
```

送信メッセージに対する返信メッセージを受け取るためのプログラムを上記のように設定したうえで、次のように送信メッセージを送ります。メッセージを送信するまえに、受信用キューをオープンしておきます。また、送信前に、トランスポート ID を保存しておき、受信時の対応付けに利用します。なお、プログラム終了時には、この受信キューを必ず閉じて、監視を終了するようにしてください。

```
try // 非同期でメッセージを送信します。
{
    if (!receiveQueue.IsConnected) receiveQueue.Open();
    TransportMessage transactionMessage = manager.CreateMessage(message);
    TransportId = transactionMessage.TransportId;
    manager.SendMessage(transactionMessage, sendQueueName);
}
catch (Exception ex) // その他のエラーの場合
{
    MessageBox.Show(ex.Message);
    Receive_richTextBox.Text = "";
}
```


◆ サーバの処理

サーバでは、クライアントからの要求メッセージを処理し、必要に応じて返信メッセージをクライアントが指定したキューに対して送信します。クライアントに対して、あらかじめ受信用キューの名称を公開しておきます。実際に受信キューにメッセージが届いた場合の処理の前半部分は、非同期の受信処理と同じです。

```
// PPSメッセージサービスマネージャを設定します。
manager = new MessageManager();
manager.SynchronizingObject = this;

// 受信用キューを設定します。
receiveQueue = manager.CreateReceiver(receiveQueueName);
receiveQueue.ReceiveMessage += new
    ReceiveMessageHandler(msgQueue_ReceiveTextMessage);
```

サーバでクライアントからの要求メッセージを受け付けるために、サーバの受信用キューの監視を監視します。キューの監視を開始するには、Open メソッドを使います。次のサンプルでは、一つのボタンで起動と停止が切り替えられるようにするために、IsConnected プロパティで、あらかじめすでに監視が開始されているかどうかを確認しています。

```
// 起動／停止ボタンがおされたときに呼ばれるコールバックメソッド
if (!receiveQueue.IsConnected)
{
    try // 開始していない場合は、サーバーを起動します。
    {
        receiveQueue.Open();
        this.Text = "PPSサーバ（起動）";
    }
    catch
    {
        MessageBox.Show("起動できません。");
        return;
    }
}
else // すでに開始している場合は、サーバを停止します。
{
    receiveQueue.Close();
    this.Text = "PPSサーバ（停止）";
```



```
}
```

受信キューにメッセージが到着した場合に呼ばれるコールバックメソッドでは、次のように受信メッセージを解析し、返信メッセージを返信してください。返信メッセージの返信は、次のように、ResponseMessage メソッドを利用して行います。この場合に、返信先に関する情報として受信メッセージを第二引数として渡してください。これによって、返信メッセージの送信先は受信メッセージの ReturnAddress プロパティの値に、トランスポート ID は、受信メッセージの TransportId と等しくなるように設定されます。

```
void msgQueue_ReceiveTextMessage(MessageConnector sender, TransportMessage m)
{
    // 受信イベントに対応して呼び出されるコールバックメソッド
    Console.Beep();

    // 返信メッセージの内容を作成します。
    Message_richTextBox.Text = m.Message;
    string responseMsg = "これは[" + m.Message + "]に対する回答です。";

    // 返信メッセージを生成し返信します。
    try
    {
        manager.ResponseMessage(responseMsg, m);
        Send_richTextBox.Text = responseMsg;
    }
    catch (Exception ex) // 返信時のエラーの場合
    {
        MessageBox.Show(ex.Message);
        Send_richTextBox.Text = "";
    }
}
```


6. Web サーバの構築

◆ SOAP によるサーバの構築

PPS メッセージサービスを利用して、SOAP 形式でメッセージ通信を行う場合のサーバ構築方法について説明します。ここでは、IIS および ASP.NET を用いて Web サービスを作成し、要求メッセージに対する応答メッセージを返すサーバの構築例を挙げます。クライアントは、PPS メッセージサービスを通じて、ここで構築したサーバに対して SOAP 形式でメッセージを送受信します。

SOAP 形式に対応したサーバを作るサンプルは、“Pps_SimpleServer_SOAP”フォルダに含まれていますので参考にしてください。Visual Studio 2005 または Visual Studio 2008 を用いて、プロジェクトを新規作成する場合は、“新しいプロジェクト”画面において“ASP.NET Web サービス アプリケーション”を選択して作成します。自動的にプロジェクトに作成される“Service.asmx”を開き、次のような Request メソッドを定義します。ここで定義したメソッドは、Web サービスとして公開されます。

```
[WebService(Namespace = "http://webservice.pslx.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    public Service()
    {
    }

    [WebMethod]
    public string Request(string xml, string senderId, string sendTo)
    {
        return "Response by SOAP from '" + senderId + "' to '" + sendTo + "':¥n" + xml;
    }
}
```

Request メソッドは、クライアントから PPS メッセージサービスによって送られたすべてのリクエストを処理します。引数 xml には、クライアントからの要求が含まれるメッセージが代入されます。そのメッセージに従って処理を行い、メソッドの戻り値として応答メッセージを文字列で返してください。引数 senderId は、要求を送ったアプリケーションを識別するための名称が代入されます。また引数 sendTo には、要求の送り先を表すコネク

ション名が代入されます。なお、上記のように、Web サービスの名前空間には、"http://webservice.pslx.org"を指定してください。

実際に業務アプリケーション開発において、要求メッセージを処理し応答メッセージを返すサーバを実装する場合には、Request メソッドにおいて、PPS ドキュメントサービスなどを用いて、処理を行うロジックを実装します。Request メソッドの引数として受け取った XML の内容を解釈し、PPS の規約にしたがって、返信メッセージを返してください。返信メッセージが必要ない場合には、長さ 0 の文字列を返してください。

上記の手順で Web サービスを作成し、Visual Studio 上で実行すると、簡易 ASP.NET サーバが起動し“http://localhost:1196/Service.asmx”などといったアドレスから Web サービスへアクセスできるようになります(ポート番号は、ランダムに設定されます)。

実際に、他のコンピュータにあるクライアントからこの Web サービスに対して要求メッセージを送ることができるようにするためには、このプロジェクトを IIS 等の Web サーバへ公開し、外部からアクセス可能な状態にする必要があります。クライアントは、その Web サービスにアクセスするため設定をクライアント側の PPS メッセージサービスの設定ファイル(Pps.Messaging.config)に書きます。Web サービスのアドレスは、設定ファイルの Location 要素の値に指定します。

SOAP 形式によるメッセージ通信は、これまで説明したメッセージキューによるメッセージ通信と同じ仕組みで利用することができます。ただし、この場合、クライアントで利用できる MessageManager クラスの送受信のメソッドは、次に示す制約があります。

SOAP 形式によるメッセージ通信では、クライアントからの要求メッセージに対して、サーバがその応答メッセージを返す流れのみとなります。メッセージキューの場合と異なり、サーバからクライアントに対してメッセージが通知されることはありません。

表 1 SOAP 形式によるメッセージ通信における制約

区分	対象メソッド	対応内容
送信	SendMessage SendTextMessage	対応しています。
受信	ReceiveMessage ReceiveTextMessage	対応していません。サーバは、必ず要求メッセージを送信する必要があります。

送受信（同期）	SyncMessage SyncTextMessage	対応しています。メッセージキューによるサーバと同様に利用することが可能です。
送受信 （非同期１）	SendMessage ReceiveMessage	対応していません。
送受信 （非同期２）	SendMessage MessageConnector.ReceiveMessage	対応していません。

◆ WSDL の定義

PPS メッセージサービスを使った SOAP 形式によるメッセージ通信では、次に挙げる SOAP サービス定義である WSDL ファイルに従った SOAP 通信を行います。ASP.NET 以外の環境で、SOAP 形式のサーバを実装する場合は、この WSDL の定義に沿った Web サービスを実装してください。

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://webservice.pslx.org/" xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
targetNamespace="http://webservice.pslx.org/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://webservice.pslx.org/">
      <s:element name="Request">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="target" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="xml" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="RequestResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="RequestResult"
type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
```



```
    </s:schema>
  </wsdl:types>
  <wsdl:message name="RequestSoapIn">
    <wsdl:part name="parameters" element="tns:Request" />
  </wsdl:message>
  <wsdl:message name="RequestSoapOut">
    <wsdl:part name="parameters" element="tns:RequestResponse" />
  </wsdl:message>
  <wsdl:portType name="ServiceSoap">
    <wsdl:operation name="Request">
      <wsdl:input message="tns:RequestSoapIn" />
      <wsdl:output message="tns:RequestSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="ServiceSoap" type="tns:ServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="Request">
      <soap:operation soapAction="http://webservice.pslx.org/Request"
style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="ServiceSoap12" type="tns:ServiceSoap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="Request">
      <soap12:operation soapAction="http://webservice.pslx.org/Request"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="Service">
    <wsdl:port name="ServiceSoap" binding="tns:ServiceSoap">
      <soap:address
location="http://localhost:4383/Pps_SimpleServer_Soap/Service.asmx" />
    </wsdl:port>
    <wsdl:port name="ServiceSoap12" binding="tns:ServiceSoap12">
      <soap12:address
location="http://localhost:4383/Pps_SimpleServer_Soap/Service.asmx" />
    </wsdl:port>
  </wsdl:service>
```



```
</wsdl:definitions>
```

◆ HTTP POST によるサーバの構築

PPS メッセージサービスを利用して、HTTP POST 形式でメッセージ通信を行う場合のサーバ構築方法について説明します。ここでは、IIS および ASP.NET を用いて単純な動的 Web ページを作成し、要求メッセージに対する応答メッセージを返すサーバの構築例を挙げます。なお HTTP POST によるサーバは、IIS+ASP.NET に限らず、Tomcat+Servlet や Apache+PHP など構成で構築することができます。クライアントは、PPS メッセージサービスを通じて、ここで構築したサーバに対して HTTP POST 形式でメッセージを送受信します。

HTTP POST 形式に対応したサーバを作るサンプルは、“Pps_SimpleServer_Http”フォルダに含まれていますので参考にしてください。Visual Studio 2005 または Visual Studio 2008 を用いて、プロジェクトを新規作成する場合は、“新しいプロジェクト”画面において“ASP.NET Web アプリケーション”を選択して作成します。自動的にプロジェクトに作成される“Default.aspx.cs”を開き、Page_Load メソッドに次のようなプログラムを書きます。ここで定義したメソッドは、クライアントから要求があった際に実行されます。

```
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (Request.RequestType.ToUpper() == "POST")
        {
            //パラメータの解釈
            string xml;
            string senderId;
            string sendTo;

            Response.ContentEncoding = Encoding.UTF8;
            if (Request.ContentType == "application/x-www-form-urlencoded")
            {
                senderId = Request.Params["senderid"];
                sendTo = Request.Params["sendto"];
                xml = Request.Params["xml"];
            }
            else if (Request.ContentType == "application/xml")
            {
                senderId = Request.Headers["X-SenderId"];
```



```
                sendTo = Request.Headers["X-SendTo"];

                StreamReader reader =
                    new StreamReader(Request.InputStream, Request.ContentEncoding);
                xml = reader.ReadToEnd();
            }
            else
            {
                Response.Write("リクエストが不正です。");
                Page.Visible = false;
                return;
            }
            Response.Write(
                "Response by HTTP POST from '" + senderId + "' to '" + sendTo + "':¥n"
            );
            Response.Write(xml);
            Page.Visible = false;
        }
    }
}
```

Page_Load メソッドは、クライアントから PPS メッセージサービスによって送られたすべてのリクエストを処理します。上記のプログラムでは、ローカル変数 xml に、クライアントからの要求が含まれるメッセージが代入されます。そのメッセージに従って処理を行い、Response.Write メソッドで応答メッセージを出力してください。ローカル変数 senderId は、要求を送ったアプリケーションを識別するための名称が代入されます。またローカル変数 sendTo には、要求の送り先を表すコネクション名が代入されます。なお、PPS メッセージサービスの HTTP POST 形式によるメッセージ通信では、ContentType が“application/x-www-form-urlencoded”または“application/xml”のどちらかで送られます。サーバは、どちらの ContentType でも受け付けることができるように実装してください。

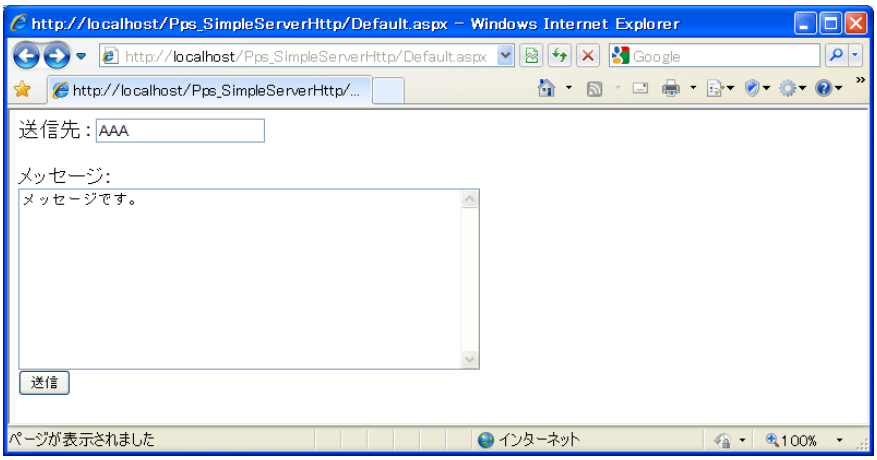
実際に業務アプリケーション開発において、要求メッセージを処理し応答メッセージを返すサーバを実装する場合には、Page_Load メソッドにおいて、PPS ドキュメントサービスなどを用いて、処理を行うロジックを実装します。クライアントから受け取った XML メッセージの内容を解釈し、PPS の規約にしたがって、返信メッセージを出力してください。返信メッセージが必要ない場合には、長さ 0 の文字列を出力してください。

HTTP POST 形式に対応したサーバの場合は、PPS メッセージサービスからに限らず、Web ブラウザからメッセージを送ることができます。ここでは、作成した動的 Web ページ

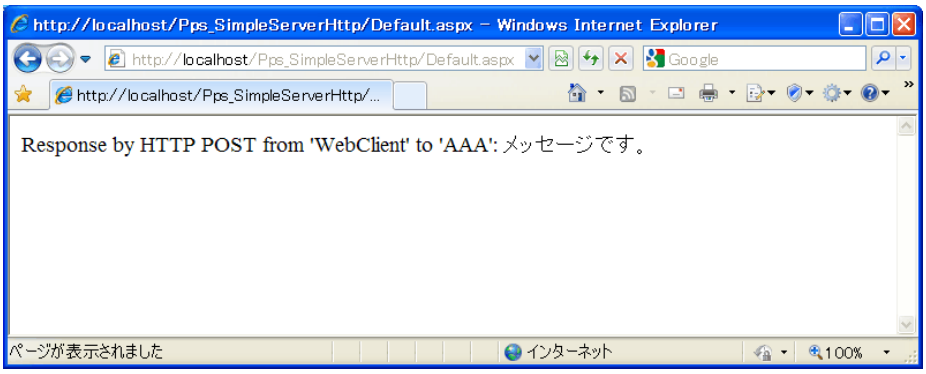
を Web ブラウザからアクセスした場合に、サーバに対してメッセージを送ることができるように“Default.aspx”に次のような HTML を書きます。

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <form action="Default.aspx" method="POST">
        <input type="hidden" name="senderid" value="WebClient"/>
        送信先 :
        <input type="text" name="sendto" value="AAA"/><br />
        <br />
        メッセージ:<br />
        <textarea name="xml" cols="50" rows="10"></textarea><br />
        <input type="submit" value="送信" />
    </form>
</body>
</html>
```

上記の Default.aspx を IIS に発行し、アクセスすると、HTTP のサンプルにあるプログラムを実行すると、次のような送信用のフォームが表示されます。



ここで、任意のメッセージを入力し、送信ボタンを押すと、Web ブラウザがサーバに対して POST 形式でメッセージを送信され、サーバから次のような応答メッセージが得られます。



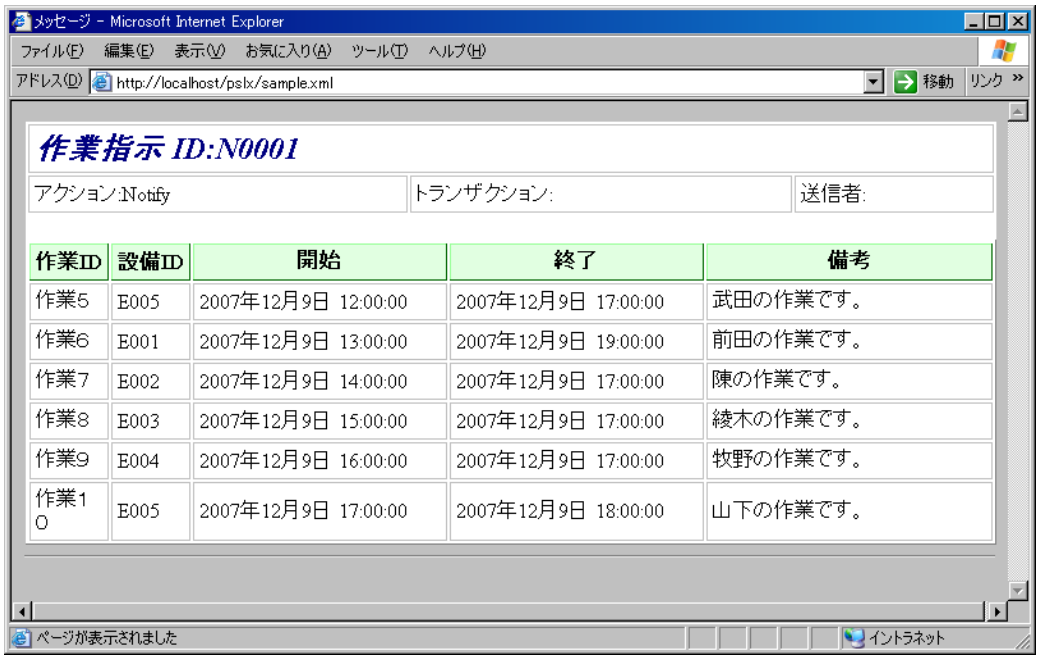
◆ XML スタイルシートによる表示

Web ブラウザをクライアントとして使用する場合は、提供されている共通コンポーネントをクライアントで使用することができせん。Web ブラウザをクライアントとして使う場合は、一般に、XML スタイルシート(XSLT)を用いて XML 形式の業務ドキュメントを、帳票形式などに変換し、ブラウザに表示する方法が考えられます。ここでは、XSLT を用いて XML 形式の業務ドキュメントを表形式で表示するサンプルを示します。

この場合、返信メッセージに XML スタイルシートの指定が Web サーバによって挿入されることになります。スタイルシートを任意に切り替えるために、必要に応じて POST メッセージに引数を追加することも可能です。

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="pslx-style.xsl"?>
<Message id="0" sender="DefaultSender" xmlns="http://docs.oasis-open.org/pps/2009">
...以下省略
```

Web ブラウザには、次のような帳票が表示されます。Web ブラウザ上に表示された表は、上記の XML を XML スタイルシートによって整形し表示したものです。




```
<xsl:template match="pslx:Document[@name='OperationSchedule']">
  <div class="doc">
    <table>
      <tr>
        <td colspan="4">
          <span class="title">
            作業指示 ID:<xsl:value-of select="@id"/>
          </span>
        </td>
      </tr>
      <tr>
        <td>
          アクション:<xsl:value-of select="@action"/>
        </td>
        <td>
          トランザクション:<xsl:value-of select="@transaction"/>
        </td>
        <td>
          送信者:<xsl:value-of select="@sender"/>
        </td>
      </tr>
    </table>
    <p>
    </p>
    <table border="1">
      <tr>
        <th>作業ID</th>
        <th>設備ID</th>
```



```
          <th>開始</th>
          <th>終了</th>
          <th>備考</th>
        </tr>
        <xsl:for-each select="pslx:Operation">
          <tr>
            <td>
              <xsl:value-of select="@id" />
            </td>
            <td>
              <xsl:value-of select="pslx:Assign[@type='pps:equipment']/@resource" />
            </td>
            <td class="datetime">
              <xsl:apply-templates select="pslx:Start" />
            </td>
            <td class="datetime">
              <xsl:apply-templates select="pslx:End" />
            </td>
            <td class="remark">
              <xsl:value-of
                  select="pslx:Description[@type='pps:comment']/pslx:Char/@value" />
            </td>
          </tr>
        </xsl:for-each>
      </table>
    </div>
    <hr />
  </xsl:template>
```


付録　サンプル実装プログラム

PPS メッセージングサービス C#版のコンポーネントには、サンプルプログラムが含まれています。本マニュアルで解説した内容に合わせて、次のサンプルプログラムが用意されています。これらのサンプルは、Visual Studio 2005 で動作確認することができます。これらのサンプルプログラムが実際のアプリケーションプログラムの開発にあたり、参考になれば幸いです。

プロジェクト名	概要
Pps_SimpleSender	任意のテキストメッセージを送信するアプリケーションプログラムのサンプルです。
Pps_SimpleReceiver	任意のテキストメッセージを要求したタイミングで受信するアプリケーションプログラムのサンプルです。
Pps_SimpleReceiver_Async	任意のテキストメッセージを受信したタイミングで受取るアプリケーションプログラムのサンプルです。
Pps_SimpleClient	任意のテキストメッセージを送信し、その直後に返信を受取るアプリケーションプログラムのサンプルです。
Pps_SimpleClient_Async	任意のテキストメッセージを送信した後、その返信を受信したタイミングで受取るアプリケーションプログラムのサンプルです。
Pps_SimpleServer	任意のテキストメッセージを受信し、送信者に対して返信メッセージを送信するアプリケーションプログラムのサンプルです。
Pps_SimpleServer_Soap	SOAP サーバ（IIS）で動作する Web アプリケーションプログラムです。送信したクライアントに対して、返信メッセージを返します。
Pps_SimpleServer_Http	Web サーバ（IIS）で動作する Web アプリケーションプログラムです。送信したクライアントに対して、返信メッセージを返します。HTTP/POST を利用しています。

