FUJITSU

WS-POS 環境セットアップ・アプリケーション開発ガイド

富士通アイソテック株式会社

Revision 1.0.0.0

目次

1. 概要

本ドキュメントは、FIT 製 OPOSドライバを利用した WS-POS 環境のセットアップ方法とアプリケーションの開発方法について記述します。
本ドキュメントでは、下記の表 1-1 の環境を例として解説しています。環境が異なる場合には、適宜読み替えてください。 また、ダウンロード URL は 2014 年 6 月時点のものです。

WS-POS(Web Services for Point of Service)は、Web ベースの POS システムでデバイス制御を行うものです。
WS-POS の仕様については、下記 URL を参照してください。
https://nrf.com/resources/retail-technology-standards/unifiedpos

1.1 環境構築に必要なソフトウェア

WS-POS 構築については以下のソフトウェアが必要となります。

表 1-1 環境構築に必要なソフトウェア

種類	説明
WS-POS 参照実装	WS-POS サービスの参照実装です。 以下からダウンロードします。 https://nrf.com/resources/retail-library/ws-pos-version-12-reference-implementation-c 日本語のドキュメント等は下記 URL を参照してください。 http://www.microsoft.com/ja-jp/business/industry/retail/wspos/default.aspx
Microsoft Point of Service for .NET v1.12 (POS for .NET 1.12)	WS-POS が利用します。 以下からダウンロードします。 http://www.microsoft.com/en-us/download/details.aspx?id=5355
Common Control Object 1.12	上記モジュールが参照する CO です。 以下からダウンロードします。 http://monroecs.com/oposccos_current.htm
FP-1100 OPOSドライバ	FP-1100 OPOSドライバー式 以下からダウンロードします。 http://jp.fujitsu.com/group/fit/services/printers/downloads/driver/fp1100/#FP-1100OPOS
.NET Framework	WS-POS の動作には、Microsoft .NET Framework 3.5 以降が必要になります。 Microsoft のサイトからダウンロードしてください。 ・Microsoft .NET Framework 3.5 http://www.microsoft.com/ja-jp/download/details.aspx?id=21 ・Microsoft .NET Framework 3.5 Service Pack 1 http://www.microsoft.com/ja-jp/download/details.aspx?id=25150 ・Microsoft .NET Framework 4.0 http://www.microsoft.com/ja-jp/download/details.aspx?id=17718 ・Microsoft .NET Framework 4.5 http://www.microsoft.com/ja-jp/download/details.aspx?id=30653

1.2 機器構成について

クライアント PC とサーバーPC の機能配置について、以下に示します。
クライアント PC とサーバーPC は LAN/WAN で通信できる状態としてください。

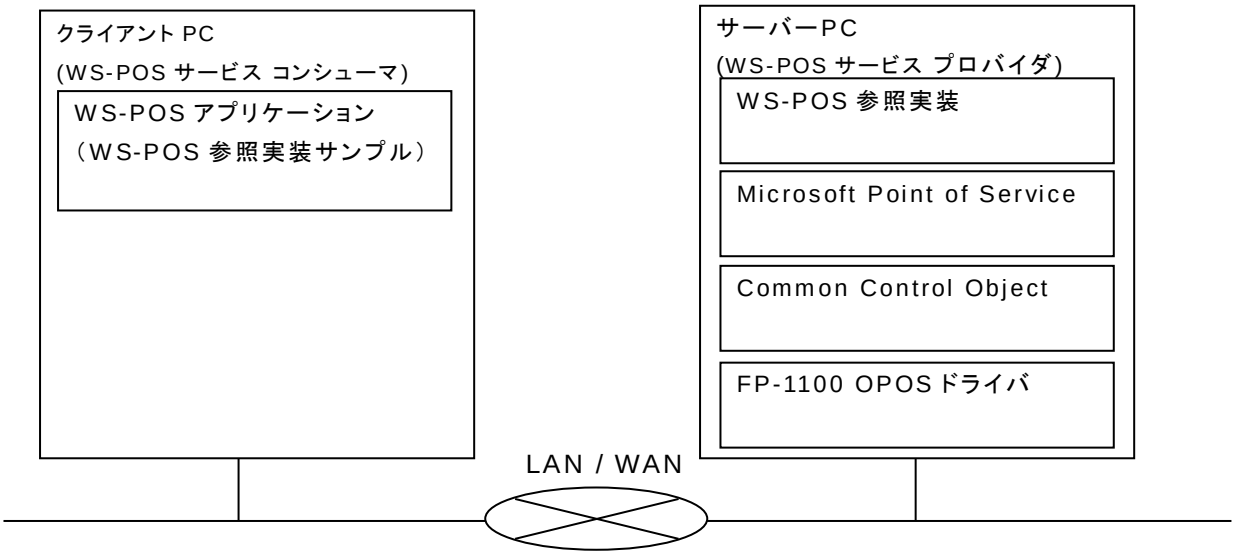


図1-1 構成図

1.3 商標について

記載されている会社名、製品名は各社の登録商標または商標です。

2. 環境セットアップと設定について

本章では、WS-POS 環境セットアップについて説明します。

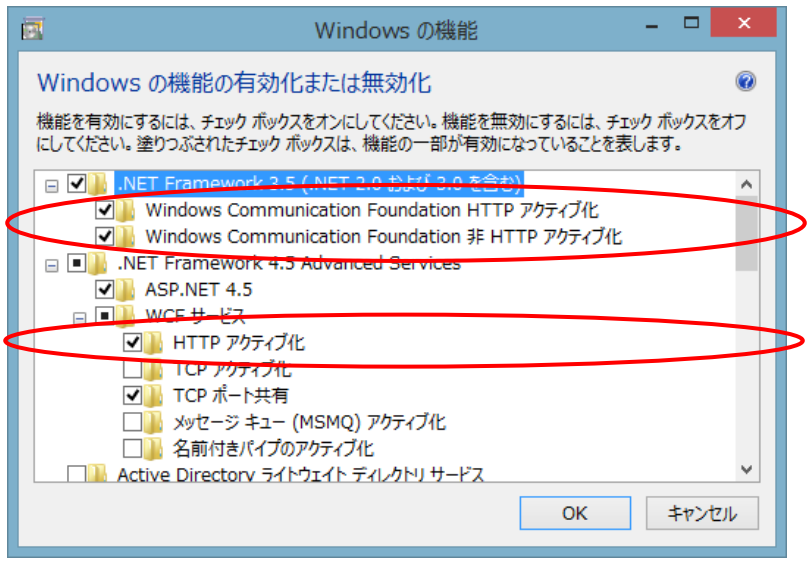
※セットアップは管理者権限で行ってください。

2.1 インストール

以下の手順で、WS-POS 環境に必要なソフトウェアをインストールします。WS-POS サービスプロバイダを構築するPC（サーバー側）にインストールします。

2.1.1 .NET Framework

WS-POSの動作に .NET Framework 3.5以降が必要になります。
Windows7 以降の OS では、有効化（下記参照）することでインストールされ使用可能になります。

手順	内容	詳細
1	.NET Framework 3.5 以降 のダウンロード	.NET Framework 3.5以降がインストールされていない場合、 以下のURLからダウンロードしてインストールしてください。 ・Microsoft .NET Framework 3.5 http://www.microsoft.com/ja-jp/download/details.aspx?id=21 ・Microsoft .NET Framework 3.5 Service Pack 1 http://www.microsoft.com/ja-jp/download/details.aspx?id=25150 ・Microsoft .NET Framework 4.0 http://www.microsoft.com/ja-jp/download/details.aspx?id=17718 ・Microsoft .NET Framework 4.5 http://www.microsoft.com/ja-jp/download/details.aspx?id=30653
2	.NET Framework の有効化	コントロールパネル > プログラムと機能 > Windows の機能の有効かまたは無効化 を選択して以下の項目にチェックを入れます。  ・Windows Communication Foundation HTTP アクティブ ・Windows Communication Foundation 非 HTTP アクティブ ・HTTP アクティブ化

2.1.2　OPOS ドライバ

ここでは、FP-1100 OPOS ドライバを例に説明します。

<table>
<tr><th>手順</th><th>内容</th><th>詳細</th></tr>
<tr><td>1</td><td>OPOS ドライバのインストール</td><td>
<32bit OSの場合>
1.　以下のURLからダウンロードします。
http://jp.fujitsu.com/group/fit/services/printers/downloads/driver/fp1100/#FP-1100OPOS

2.　ダウンロードした実行ファイルをダブルクリックして解凍します。

3.　解凍して作成されたフォルダ内（1つ下の階層のフォルダの場合があります）のSetup.exeを起動します。

4.　以下の画面が表示されます。画面の指示に従いインストールを行ってください。


<64bit OSの場合>
1.　以下のURLからダウンロードします。
http://jp.fujitsu.com/group/fit/services/printers/downloads/driver/fp1100/#FP-1100OPOS

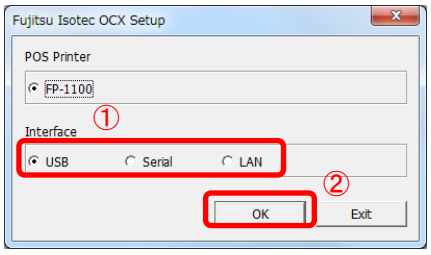
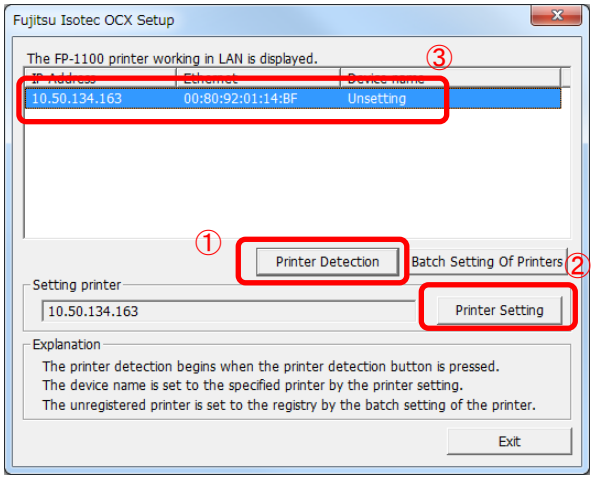
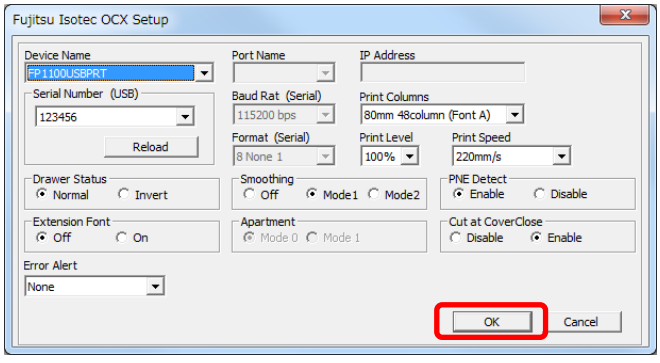
2.　ダウンロードした実行ファイルをダブルクリックして解凍します。

3.　管理者権限でコマンドプロンプトを起動し、解凍して作成されたフォルダ内のSetup.exeがあるフォルダに移動します。

4.　Setup.exeに起動オプション”/32bit”を付けて実行します。
コマンドプロンプトから下記を実行してください。
　＞ Setup.exe /32bit

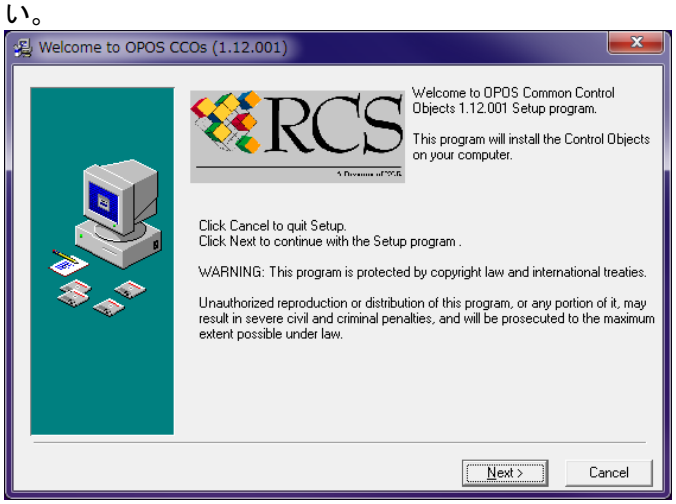
5.　以下の画面が表示されます。画面の指示に従いインストールを行ってください。


</td></tr>
</table>

手順	内容	詳細
2	OPOS ドライバとプリンタのセットアップ	1. 使用するプリンタをPCに接続し、電源を入れます。 2. OPOSのセットアップを行います。Windows8以降の場合は、アプリー覧画面から、その他のOSの場合は、スタートメニューから「FP Printer OPOS Setup」を起動します。 3. インターフェースを選択し、[OK]をクリックします。  **※ LAN I/Fを選択した場合** LAN接続されたプリンタを選択する画面が表示されます。 [Printer Detection]をクリックし、検索されたプリンタを選択後、[Printer Setting]をクリックします。  4. プリンタの設定を任意に変更し、[OK]をクリックします。 この画面で表示されている”Device Name”は、2.3、2.4章で使用する”デバイス名”です。 

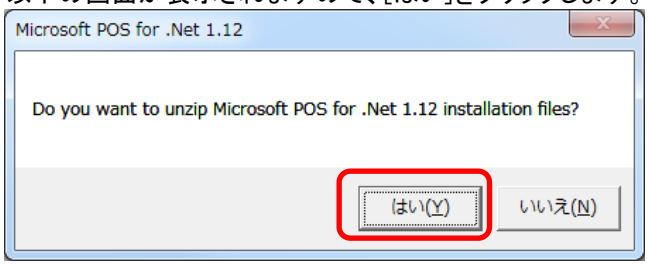
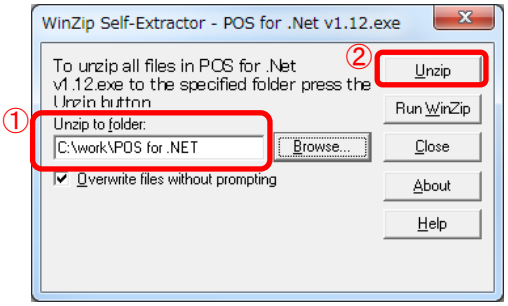
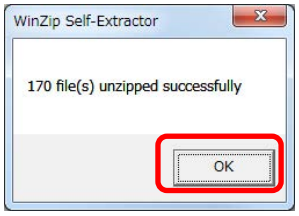
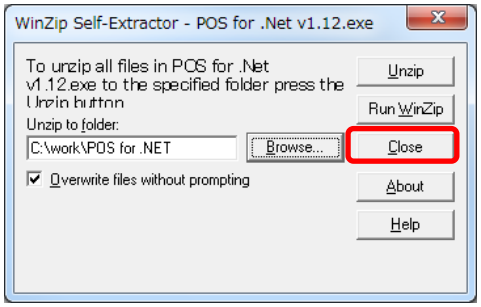
2.1.3 Common Control Object (共通 CO)

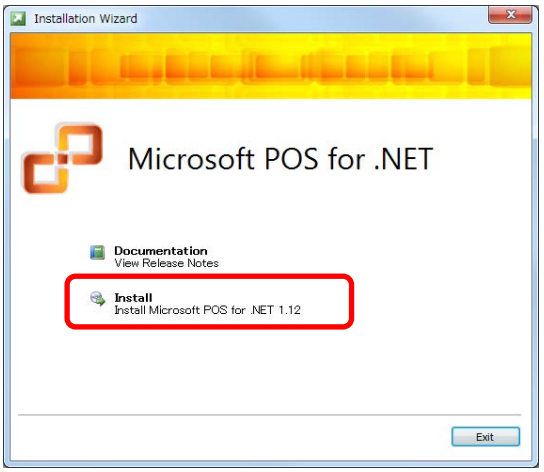
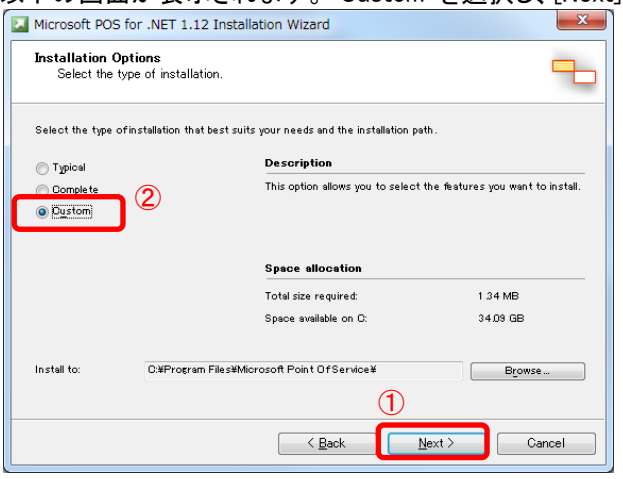
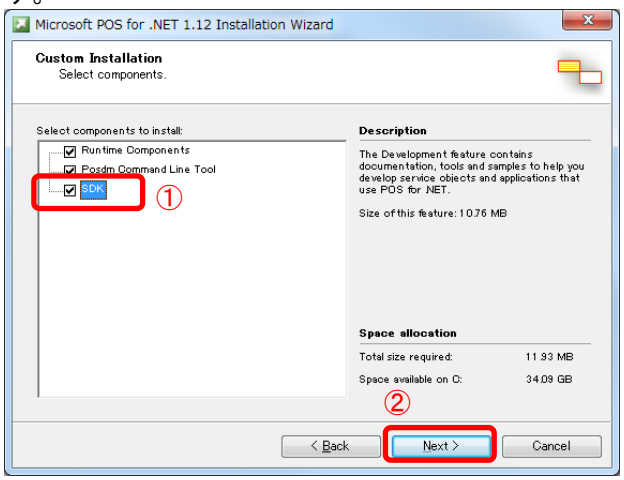
Common Control Object(共通 CO)をインストールします。

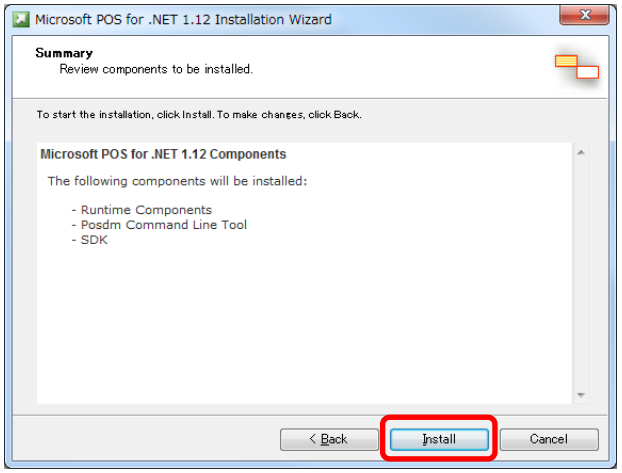
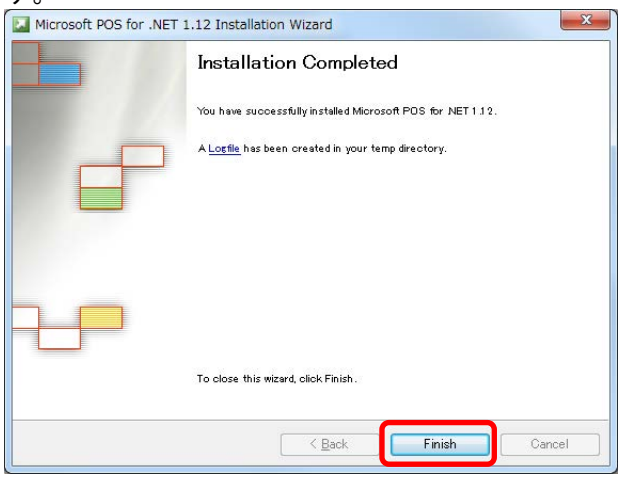
手順	内容	詳細
1	Common Control Object のインストール	1. 以下のURLから「1.12.001 CCO Runtime (Wize Install)」をダウンロードします。 http://monroecs.com/oposccos_current.htm 2. ダウンロードした OposCCOs-1_12_001.exe をダブルクリックします。 以下の画面が表示されます。画面の指示に従いインストールを行ってください。  **※共通COはFP-1100 OPOSより後にインストールしてください。** 共通COのインストール後にFP-1100 OPOSドライバをインストールすると、POS for .NET からデバイスが使用できなくなります。これはレジストリのOPOSデバイスのCO設定が、共通COからFP-1100 OPOSのCOに上書きされるためです。 この場合は、共通CO をアンインストール＆再インストールするか、RegSvr32.exe を使用して、共通CO の各OCX を再度レジストリに登録し直しすると使用できるようになります。

2.1.4　POS for .NET

POS for .NET 1.12をインストールします。
※インストールの際は、**POS for .NET SDK**も選択してください。

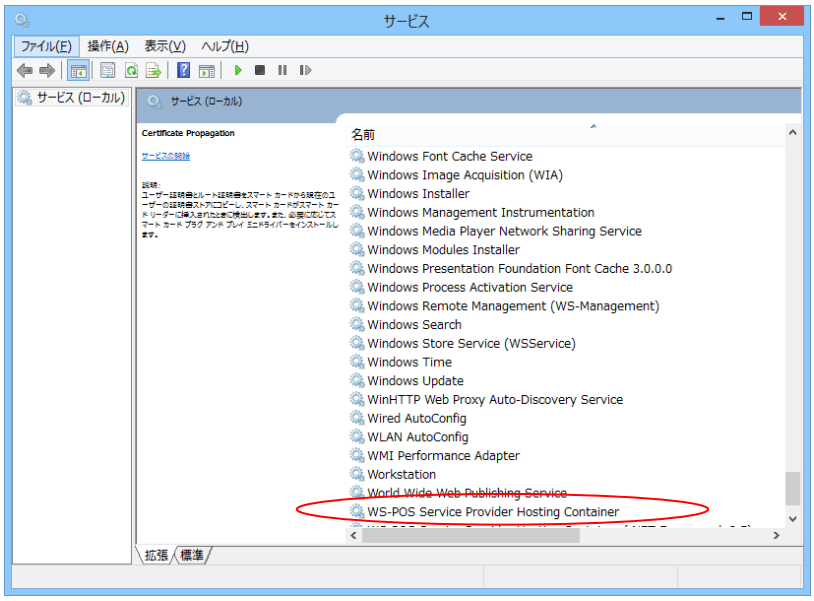
手順	内容	詳細
1	POS for .NET 1.12 のダウンロードと解凍	1. 以下の URL からダウンロードします。 http://www.microsoft.com/en-us/download/details.aspx?id=5355 2. ダウンロードした POS for .NET v1.12.exe をダブルクリックします。 以下の画面が表示されますので、[はい]をクリックします。  3. 以下の画面が表示されます。"Unzip to folder"に、任意の解凍先フォルダを指定して、[Unzip]をクリックします。  4. 解凍が完了すると、以下の画面が表示されます。[OK]をクリックします。  5. [Close]をクリックします。 

手順	内容	詳細
2	POS for .NET 1.12 のインストール	1. 解凍先フォルダ内の Setup.exe を起動します。 2. 以下の画面が表示されます。[Install]をクリックします。  3. 次の画面で[Next]をクリックし、更に次の画面で[Accept]をクリックします。 4. 以下の画面が表示されます。”Custom”を選択し、[Next]をクリックします。  5. 以下の画面が表示されます。”SDK”にもチェックを入れ、[Next]をクリックします。 

手順	内容	詳細
2 つづき	POS for .NET 1.12 の インストール	6. 以下の画面が表示されます。[Install]をクリックします。 インストールが開始されます。  7. インストールが完了すると、以下の画面が表示されます。[Finish]をクリックします。 

2.1.5　WS-POS 環境

WS-POS 参照実装をダウンロードし、配置します。

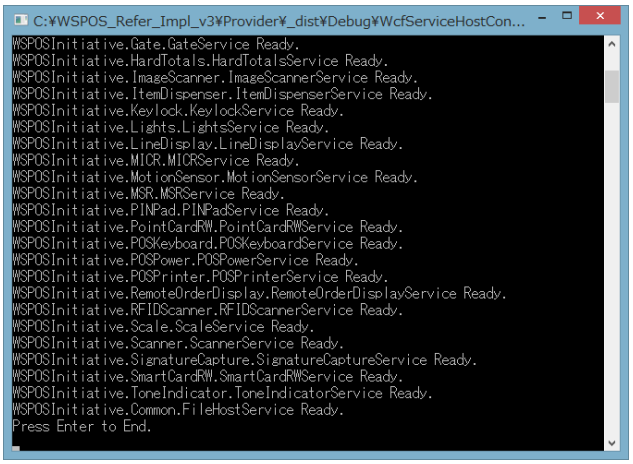
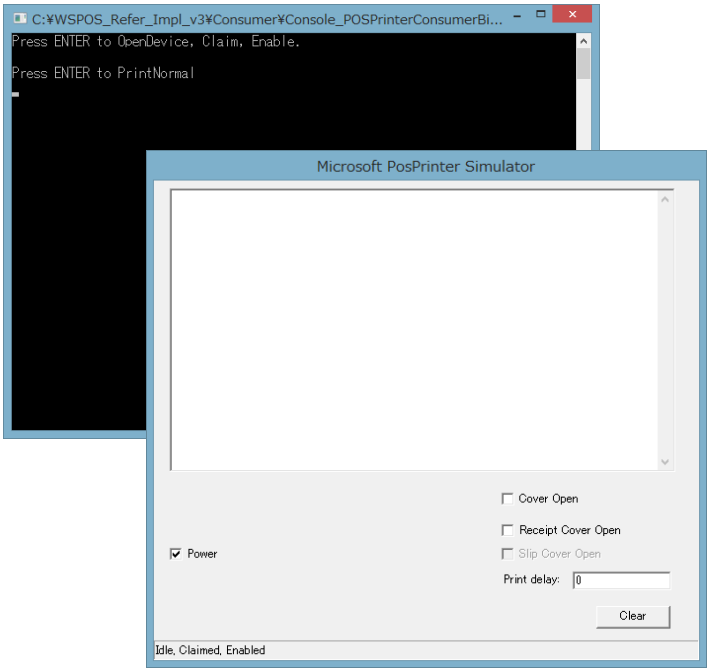
手順	内容	詳細
1	WS-POS 参照実装のダウンロードと配置	1. 以下からダウンロードします。 https://nrf.com/resources/retail-library/ws-pos-version-12-reference-implementation-c 2. ダウンロードした ARTS WS-POS 1.2 Reference Implementation_0.zip を解凍します。 3. 解凍したフォルダから、WS-POS 参照実装のフォルダ(WSPOS_Refer_Impl_v3)を、任意のフォルダにコピーします。(以降コピーしたフォルダを[WS-POS フォルダ]と記述します。)
2	WS-POS サービスプロバイダを Windows サービスとしてインストール	1. コマンドプロンプトを**管理者権限**で起動します。 2. 以下のコマンドを入力し、WS-POS 参照実装のフォルダに移動します。 > cd [WS-POS フォルダ]¥WSPOS_Refer_Impl_v3¥Provider¥_dist¥Release 3. C:¥Windows¥Microsoft.NET¥Framework¥v2.0.50727 配下にある InstallUtil を利用して WS-POS サービスプロバイダを Windows サービスとしてインストールします。以下のコマンドを入力します。 > C:¥Windows¥Microsoft.NET¥Framework¥v2.0.50727¥ InstallUtil WcfServiceHostWindowsService35.exe [ENTER] 4. コントロールパネルからサービスを表示し、"WS-POS Service Provider Hosting Container"のサービス名が表示されていることを確認します。  サービスは停止している状態にしておいてください。 次手順で、コンソールアプリケーションによるサービスプロバイダ起動確認を行いますので、サービスが動作しているとコンソールアプリケーションと衝突してしまうためです。

2.1.6　その他の設定

手順	内容	詳細
1	ファイアウォールの設定	参照実装のデフォルトでは、WS-POS サービスプロバイダは8087 ポートにて WS-POS サービスコンシューマからのリクエストを待ちます。 このため、サービスプロバイダがポートを開いて待ち受け可能なように、オペレーティングシステムやセキュリティソフトウェアにて上記ポートの通信をブロックしている場合は、設定変更が必要です。

2.2 インストール確認

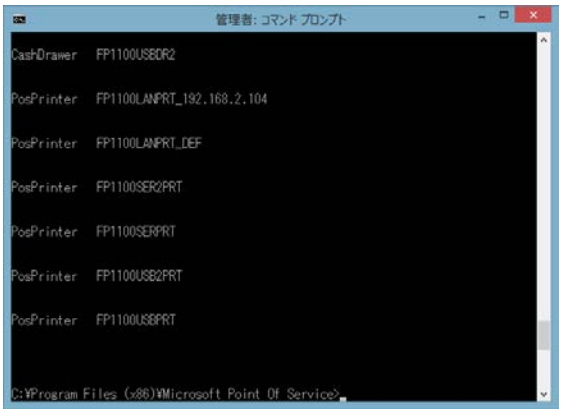
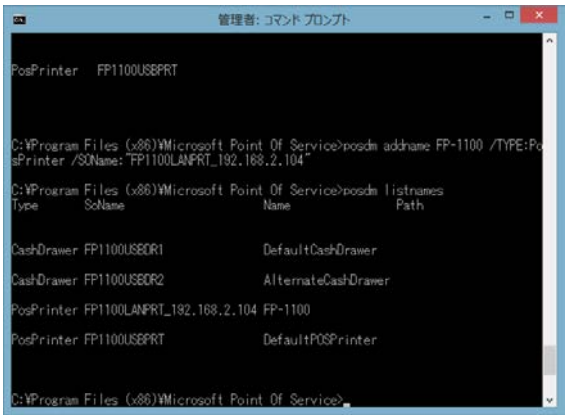
インストールが正常に行われたかどうかを確認します。下記手順をサーバーで行ってください。

手順	内容	詳細
1	コンソールアプリケーションによるサービスプロバイダ起動確認	[WS-POS フォルダ]¥WSPOS_Refer_Impl_v3¥Provider¥_dist¥Release¥WcfServiceHostConsoleExe35.exe を起動します。 成功するとコマンドプロンプトが表示され、待ち受け状態となります。 
2	サービスコンシューマサンプルの実行	[WS-POS フォルダ]¥WSPOS_Refer_Impl_v3¥Consumer¥Console_POSPrinterConsumerBiDirectionSample¥bin¥ Debug¥Console_POSPrinterConsumerBiDirectionSample.exe を起動します。 成功するとコマンドプロンプトが表示され、待ち受け状態となります。  Enter キーを押すと、POSPrinter シミュレータが動作します。 Enter キーを押していくと、終了します。

これらの動作が確認できれば、インストールは正常に完了しています。
これらアプリケーションは動作確認後、終了してください。

2.3 実デバイスを用いた実行のための設定（POS プリンタ）

環境の構築が完了したら、FP-1100 向けに固有の設定を行います。

手順	内容	詳細
1	OPOS デバイス名確認	コマンドプロンプトを管理者権限で起動します。 C:¥Program Files¥Microsoft Point Of Service　に移動(64bit OS の場合は、C:¥Program Files (x86) ¥Microsoft Point Of Service)し、以下のコマンドで、対象のドライバが出力されるかを確認します。 > posdm listsos 以下のような画面が表示されます。FP-1100 用のデバイス名が表示されることを確認します。 
2	OPOS デバイス登録	以下のコマンドでデバイスを登録します > posdm addname FP-1100 /TYPE:PosPrinter /SOName:"FP1100LANPRT_192.168.2.104" この例では、"FP-1100"というエイリアスについて、 デバイス名"FP1100LANPRT_192.168.2. 104"を登録します。
3	登録デバイス確認	以下のコマンドでデバイスを登録します > posdm listnames FP-1100 が登録されていることを確認します。 

<table>
<tr><th>手順</th><th>内容</th><th>詳細</th></tr>
<tr><td>4</td><td>サービスコンシューマ設定ファイルの変更
（クライアント側）</td><td>[WS-POS フォルダ] ¥WSPOS_Refer_Impl_v3¥Consumer¥
Console_POSPrinterConsumerBiDirectionSample¥bin¥Debug¥
Console_POSPrinterConsumerBiDirectionSample.exe.config
の以下の部位を修正します。

◆修正前
<endpoint address=
"http://localhost:8087/POSPrinter.svc/POSPrinterSimulator"
binding="basicHttpBinding"
contract="UnifiedPOS.POSPrinter.V1_2.POSPrinter"
name="POSPrinterPort"/>
↓
◆修正後
<endpoint address=
"http://localhost:8087/POSPrinter.svc/POSPrinterFP1100"
binding="basicHttpBinding"
contract="UnifiedPOS.POSPrinter.V1_2.POSPrinter"
name="POSPrinterPort"/></td></tr>
<tr><td>5</td><td>サービスプロバイダ設定ファイルの変更（サーバー側）</td><td>[WS-POS フォルダ]¥WSPOS_Refer_Impl_v3¥Provider¥_dist¥Release¥ フォルダにて、下記ファイルの以下に示す部位を修正します。
・WcfServiceHostConsoleExe35.exe.config

◆修正前
<behavior name="POSPrinter_Simulator">
<wsposDevice deviceName="Microsoft PosPrinter Simulator"
providerSessionTimeout="180" eventPollingTimeout="-1"
eventResponseTimeout="-1"/>
</behavior>
↓
◆修正後
<behavior name="POSPrinter_FP1100">
<wsposDevice deviceName="FP-1100"
providerSessionTimeout="180" eventPollingTimeout="-1"
eventResponseTimeout="-1"/>
</behavior>

◆修正前
<endpoint address="POSPrinterSimulator"
behaviorConfiguration="POSPrinter_Simulator"
binding="basicHttpBinding"
contract="UnifiedPOS.POSPrinter.V1_2.POSPrinter"
bindingNamespace=
"http://www.nrf-arts.org/UnifiedPOS/POSPrinter/"/>
↓
◆修正後
<endpoint address="POSPrinterFP1100"
behaviorConfiguration="POSPrinter_FP1100"
binding="basicHttpBinding"
contract="UnifiedPOS.POSPrinter.V1_2.POSPrinter"
bindingNamespace=
"http://www.nrf-arts.org/UnifiedPOS/POSPrinter/"/></td></tr>
</table>

2.4 実デバイスでの実行（コンシューマとプロバイダが同一 PC）

コンシューマとプロバイダを同一 PC で動作させ、実デバイスの動作を実行します。

プリンタを PC に接続し、「2.2. インストール確認」の手順を実施すると、プリンタから印刷が行われます。
印刷が行われない場合は、「2.3 実デバイスを用いた実行のための設定（POS プリンタ）」に間違いがないか確認してください。

2.5 実デバイスでの実行（コンシューマとプロバイダが別 PC）

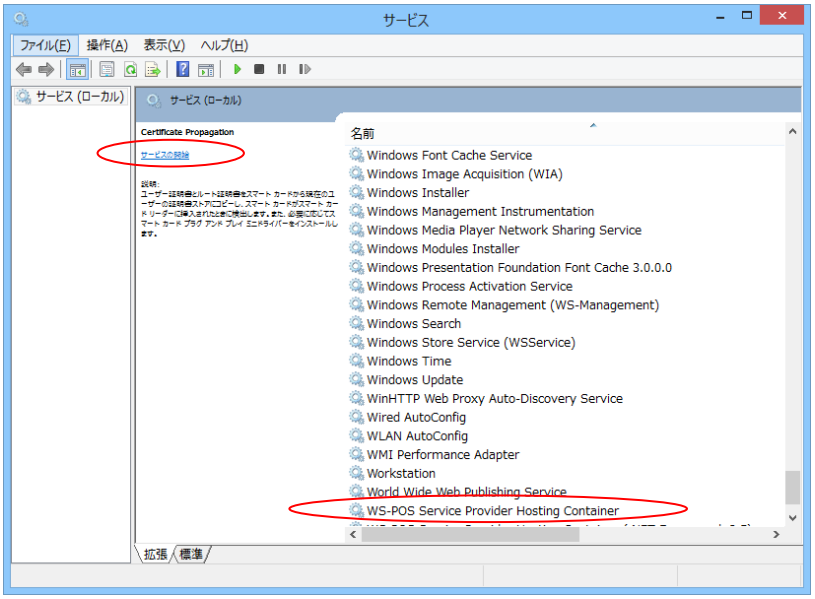
コンシューマ(クライアント)とプロバイダ(サーバー)をそれぞれ別々の PC で動作させ、実デバイスの動作を実行します。

手順	内容	詳細
1	サービスコンシューマファイルをクライアントPCにコピー	サーバーPCのフォルダ、 [WS-POS フォルダ] ¥WSPOS_Refer_Impl_v3¥Consumer¥ Console_POSPrinterConsumerBiDirectionSample¥bin¥Debug¥ をクライアントPCにコピーします。
2	サービスコンシューマ設定ファイルのサービスプロバイダ URL の変更 （クライアント PC 側）	サービスプロバイダ URL を変更します。 クライアントPCで、 [WS-POS フォルダ] ¥WSPOS_Refer_Impl_v3¥Consumer¥ Console_POSPrinterConsumerBiDirectionSample¥bin¥Debug¥ Console_POSPrinterConsumerBiDirectionSample.exe.config の以下の部位を修正します。 ◆修正前 <endpoint address= "http://localhost:8087/POSPrinter.svc/POSPrinterFP1100" binding="basicHttpBinding" contract="UnifiedPOS.POSPrinter.V1_2.POSPrinter" name="POSPrinterPort"/> ↓ ◆修正後 <endpoint address= "http://**10.50.13x.xxx**:8087/POSPrinter.svc/POSPrinterFP1100" binding="basicHttpBinding" contract="UnifiedPOS.POSPrinter.V1_2.POSPrinter" name="POSPrinterPort"/> 「10.50.13x.xxx」はサーバーPC の IP アドレスです。

手順	内容	詳細
3	サービスコンシューマ設定ファイルのサービスコンシューマ URL の変更 （クライアント PC 側）	サービスプロバイダに通知するサービスコンシューマの URL を変更します。 クライアントPCで、 [WS-POS フォルダ] ¥WSPOS_Refer_Impl_v3¥Consumer¥ Console_POSPrinterConsumerBiDirectionSample¥bin¥Debug¥ Console_POSPrinterConsumerBiDirectionSample.exe.config の以下の部位を修正します。 ◆修正前 <host> <baseAddresses> <add baseAddress="http://localhost:8001/POSPrinterEvent"/> </baseAddresses> </host> ↓ ◆修正後 <host> <baseAddresses> <add baseAddress="http://**10.50.13x.yyy**:8001/POSPrinterEvent"/> </baseAddresses> </host> 「10.50.13x.yyy」はクライアント PC の IP アドレスです。
4	実デバイスでの動作確認	プリンタを PC に接続し、「2.2. インストール確認」の手順を実施してください。

2.6 WS-POS サービスプロバイダを Windows Service で実行

ここまでは、サービスプロバイダはコンソールアプリケーションを使用していましたが、「2.1 インストール」の手順 5 で登録した Windows Service での実行手順を説明します。

手順	内容	詳細
1	サービスプロバイダ設定ファイルの変更（サーバー側）	[WS-POS フォルダ]¥WSPOS_Refer_Impl_v3¥Provider¥_dist¥Release¥ フォルダの WcfServiceHostWindowsService35.exe.config を 2.2 の手順 5 および 2.3 の手順 5 と同様に変更します。
2	サービスの起動	コントロールパネルからサービスを表示します。 ”WS-POS Service Provider Hosting Container”のサービスを選択し、 “サービス起動“をクリックします。  編集した.config ファイルに間違いがなければ、サービスが正常に開始します。
3	サービスコンシューマの起動	Console_POSPrinterConsumerBiDirectionSample.exe などのコンシューマを起動します。動作はコンソールアプリケーションのサービスプロバイダと変わりません。

2.7　サービスプロバイダの各種設定

ここでは、サービスプロバイダの設定可能な項目について説明します。

2.7.1　Behavior 名、デバイス名の変更

以下の部位を変更します。

```
<behavior name="POSPrinter_Simulator">
  <wsposDevice deviceName="Microsoft PosPower Simulator" providerSessionTimeout="180"
    eventPollingTimeout="-1" eventResponseTimeout="-1" />
</behavior>
```

Key	説明
behaviour　name	使用する behavior 名を記載します。
wsposDevice deviceName	使用するデバイス名を記載します。
providerSessionTimeout	セッションのタイムアウト時間を秒単位で指定します。
eventPollingTimeout	ロングポーリングのタイムアウト時間を秒単位で指定します。 -1 を指定した場合はタイムアウトしません。
eventResponseTimeout	ロングポーリング/自己ホストのイベント処理終了待ちのタイムアウト時間を秒単位で指定します。 -1 を指定した場合はタイムアウトしません。

2.7.2　サービスプロバイダ URL の変更

以下の部位を変更します。

```
<service name="WSPOSInitiative.POSPrinter.POSPrinterService"
        behaviorConfiguration="WS-POS_ServiceProvider">
  <host>
    <baseAddresses>
      <add baseAddress="http://localhost:8087/POSPrinter.svc"/>
    </baseAddresses>
  </host>
  <endpoint address="POSPrinterSimulator"
          behaviorConfiguration="POSPrinter_Simulator"
          binding="basicHttpBinding" contract="UnifiedPOS.POSPrinter.V1_2.POSPrinter"
          bindingNamespace="http://www.nrf-arts.org/UnifiedPOS/POSPrinter/"/>
</service>
```

Key	説明
add baseAddress	サービスプロバイダ URL を下記形式で指定します。 http://[使用するプロバイダの IP アドレス:ポート番号]/POSPrinter.svc 例） http://192.168.1.120:8087/POSPrinter.svc

2.7.3　endpoint の変更

以下の部位を変更します。

```
<service name="WSPOSInitiative.POSPrinter.POSPrinterService"
        behaviorConfiguration="WS-POS_ServiceProvider">
  <host>
    <baseAddresses>
      <add baseAddress="http://localhost:8087/POSPrinter.svc"/>
    </baseAddresses>
  </host>
  <endpoint address="POSPrinterSimulator"
        behaviorConfiguration="POSPrinter_Simulator"
        binding="basicHttpBinding" contract="UnifiedPOS.POSPrinter.V1_2.POSPrinter"
        bindingNamespace="http://www.nrf-arts.org/UnifiedPOS/POSPrinter/"/>
</service>
```

Key	説明
endpoint address	使用する endpoint 名を記載します。
behaviorConfiguration	使用する behavior 名を記載します。

※同じデバイス（例えば POSPrinter）を複数使用する場合は、<endpoint>の定義をコピーして複数のデバイス定義を記載します。

例）

```
<service name="WSPOSInitiative.POSPrinter.POSPrinterService"
        behaviorConfiguration="WS-POS_ServiceProvider">
  <host>
    <baseAddresses>
      <add baseAddress="http://localhost:8087/POSPrinter.svc"/>
    </baseAddresses>
  </host>
  <endpoint address="POSPrinter1"
      behaviorConfiguration="POSPrinter_Behavior1"
      binding="basicHttpBinding" contract="UnifiedPOS.POSPrinter.V1_2.POSPrinter"
      bindingNamespace="http://www.nrf-arts.org/UnifiedPOS/POSPrinter/"/>
  <endpoint address="POSPrinter2"
      behaviorConfiguration="POSPrinter_Behavior2"
      binding="basicHttpBinding" contract="UnifiedPOS.POSPrinter.V1_2.POSPrinter"
      bindingNamespace="http://www.nrf-arts.org/UnifiedPOS/POSPrinter/"/>
</service>
```

2.7.4　セッションタイムアウトの確認間隔の変更

以下の部位を変更します。

```
<appSettings>
      …
  <add key="POSPrinterService_ProviderSessionTimeoutCheckPeriod" value="1"/>
      …
</appSettings>
```

add key の値	Key	説明
POSPrinterService_ProviderSessionTimeoutCheckPeriod	value	セッションのタイムアウトを定期的に確認する間隔(秒)を指定します。

2.7.5　通信ポートの最大同時接続数の変更

以下の部位を変更します。

```
<system.net>
  <connectionManagement>
    <add address = "*" maxconnection = "48" />
  </connectionManagement>
</system.net>
```

Key	説明
maxconnection	サービスプロバイダとサービスコンシューマ間の通信で同時に使用する最大ポート数を指定します。 スレッド毎にポートを開くため、スレッドの数を考慮した値を指定してください。例えば、参照実装のサンプルでは、main スレッド、Polling 処理スレッド、KeepAlive 処理スレッドの3つが必要になります。

2.7.6　使用する.NET Framework バージョンの変更

以下の部位を変更します。

```
<startup>
  <supportedRuntime version="v2.0.50727"/>
</startup>
```

Key	説明
supportedRuntime version	使用する Microsoft .NET Framework のバージョンを指定します。 ・Microsoft .NET Framework 4.0 : "v4.0" ・Microsoft .NET Framework 3.5 : "v2.0.50727" ・Microsoft .NET Framework 3.0 : 〃

2.8　サービスコンシューマの各種設定

ここでは、サービスコンシューマの設定可能な項目について説明します。

2.8.1　endpoint の変更

以下の部位を変更します。

```
<client>
  <endpoint address="http://localhost:8087/POSPrinter.svc/POSPrinterSimulator"
          binding="basicHttpBinding"
          contract="UnifiedPOS.POSPrinter.V1_2.POSPrinter"
          name="POSPrinterPort"/>
</client>
```

Key	説明
endpoint address	接続するサービスプロバイダの URL と endpoint 名を下記形式で指定します。 http://[使用するプロバイダの IP アドレス:ポート番号]/POSPrinter.svc/[endpoint 名] 例） http://192.168.1.120:8087/POSPrinter.svc/POSPrinterFP1100
name	この接続の名称を指定します。

※同じデバイス（例えば POSPrinter）を複数使用する場合は、<endpoint>の定義をコピーして複数のデバイス定義を記載します。

例）

```
<client>
  <endpoint address="http://192.168.1.120:8087/POSPrinter.svc/POSPrinter1"
          binding="basicHttpBinding"
          contract="UnifiedPOS.POSPrinter.V1_2.POSPrinter"
          name="POSPrinterPort1"/>
  <endpoint address="http:// 192.168.1.120:8087/POSPrinter.svc/POSPrinter2"
          binding="basicHttpBinding"
          contract="UnifiedPOS.POSPrinter.V1_2.POSPrinter"
          name="POSPrinterPort2"/>
</client>
```

2.8.2　サービスコンシューマ URL の変更

サービスコンシューマがサービスプロバイダからイベントを受信するため、サービスプロバイダへサービスコンシューマの URL を下記設定により通知します。(Open メソッドで通知されます)
以下の部位を変更します。

```
<host>
  <baseAddresses>
    <add baseAddress="http://localhost:8001/POSPrinterEvent"/>
  </baseAddresses>
</host>
```

Key	説明
add baseAddress	サービスコンシューマの URL を下記形式で指定します。 http://[コンシューマの IP アドレス:ポート番号]/POSPrinterEvent.svc 例) http://192.168.1.101:8001/POSPrinterEvent.svc

3. WS-POS アプリケーションの作成

3.1 開発環境

Visual Studio 2010 または、Visual Studio 2012 を用いて開発を行います。
ここでは、Visual Studio 2012 を例に説明します。

3.2 開発言語

WS-POS サービスを Windows サービスで提供する場合を説明します。
下記の開発言語が使用可能です。

・Microsoft Visual Basic .NET
・Microsoft Visual C#

3.3 作成手順

3.3.1 新規プロジェクトの作成

手順	Visual Basic .NET	Visual C#
1	Visual Basic の Windows アプリケーションで作成します。	C#の Windows アプリケーションで作成します。

※既存のプロジェクトでも作成可能です。
※コンソールアプリケーションや Windows フォームアプリケーションでも作成可能です。

3.3.2 WSPOSContract の追加

手順	Visual Basic .NET	Visual C#
1	参照実装に含まれている、 "WSPOSContract.dll を、プロジェクトフォルダにコピーします。	
2	ソリューションエクスプローラからプロジェクトを右クリックし、[参照の追加]をクリックします。	
3	[参照]タブを選択し、 "WSPOSContract.dll"を選択後、[OK]をクリックします。	
4	ソリューションエクスプローラに "WSPOSContract.dll"が追加されていることを確認してください。	

3.3.3 System.ServiceModel の追加

手順	Visual Basic .NET	Visual C#
1	ソリューションエクスプローラからプロジェクトを右クリックし、[参照の追加]をクリックします。	
2	[.NET]タブを選択し、 "System.ServiceModel"を選択後、[OK]をクリックします。	
3	ソリューションエクスプローラに "System.ServiceModel"が追加されていることを確認してください。	

3.3.4　アプリケーション構成ファイルの追加

手順	Visual Basic .NET	Visual C#
1	ソリューションエクスプローラの「すべてのファイルを表示」をクリックします。	ソリューションエクスプローラからプロジェクトを選択し、メニューバーの[プロジェクト]-[新しい項目の追加]を選択します。
2	ソリューションエクスプローラに”App.config”が追加されていることを確認してください。 ※”App.config”が追加されていない場合は、4 以降を実施してください。	[Visual C#アイテム]-[全般]から、[アプリケーション構成ファイル]を選択し、[追加]をクリックします。
3	”App.config”を右クリックし、[プロジェクトに含める]をクリックします。 ※4 以降の実施は不要です。	ソリューションエクスプローラに”App.config”が追加されていることを確認してください。
4	ソリューションエクスプローラからプロジェクトを選択し、メニューバーの[プロジェクト]-[新しい項目の追加]を選択します。	
5	[共通項目]-[全般]から、[アプリケーション構成ファイル]を選択し、[追加]をクリックします。	
6	ソリューションエクスプローラに”App.config”が追加されていることを確認してください。	

3.3.5　アプリケーション構成ファイルの編集

アプリケーション構成ファイルに下記の太字部分を記述します。
(Visual Basic .NET, Visual C#共通です)

```
<?xml version="1.0"?>
<configuration>

  <system.serviceModel>
    <client>
      <endpoint address="http://192.168.1.210:8087/POSPrinter.svc/POSPrinter1"
                binding="basicHttpBinding"
                contract="UnifiedPOS.POSPrinter.V1_2.POSPrinter"
                name="POSPrinterPort1"/>
    </client>
    <services>
  </system.serviceModel>

  <system.net>
    <connectionManagement>
      <add address = "*" maxconnection = "48" />
    </connectionManagement>
  </system.net>

</configuration>
```

Key	説明
endpoint address	使用するサービスプロバイダの URL を指定します。
binding	サービスプロバイダで設定している binding “basicHttpBinding”を指定します。
contract	使用するデバイスカテゴリーのコントラクトを指定します。
name	この接続情報を識別するための名称を指定します。 サービスプロバイダと接続する際に、接続情報を取得するために使用します。
maxconnection	サービスプロバイダとサービスコンシューマ間の通信で同時に使用する最大ポート数を指定します。 スレッド毎にポートを開くため、スレッドの数を考慮した値を指定してください。例えば、参照実装のサンプルでは、main スレッド、Polling 処理スレッド、KeepAlive 処理スレッドの3つが必要になります。

3.3.6　アプリケーションソースの編集

アプリケーションのソースに、サービスプロバイダとの接続処理を記述します。
サービスプロバイダとのセッション開始～終了の間に、メソッド・プロパティの処理を記述します。

<Visual Basic .NET>

◆記述ファイル

[アプリケーションで指定したファイル名].vb

◆ConsumerID を生成

ConsumerID は、デバイスを識別するために使用します。

```
Dim consumerID As String = Guid.NewGuid().ToString()
```

※上記例は、GUID を生成して ID として使用しています。

◆メッセージ送信用チャネル生成

```
Dim device As UnifiedPOS.POSPrinter.V1_2.POSPrinter
' アプリケーション構成ファイルで設定した endpoint の contract と name を指定する
Dim factory As New ChannelFactory
            (Of UnifiedPOS.POSPrinter.V1_2.POSPrinter)("POSPrinterPort1")
' チャネルを生成
device = factory.CreateChannel()
```

◆セッションの開始

サービスプロバイダとのセッションを開始します。

```
device.OpenSession(consumerID, Nothing)
```

※OpenSession 後、各メソッドの呼び出しが可能になります。

◆メソッド・プロパティの使用

printNormal メソッドを使用する場合、下記のように記述します。

```
device.OpenDevice(consumerID)
device.Claim(consumerID, 10000)
device.SetDeviceEnabled(consumerID, true)
device.PrintNormal(consumerID,
            UnifiedPOS.POSPrinter.V1_2.PrinterStation.Receipt, "印刷するテキスト")
‘ エスケープシーケンスを使用し Bold 印刷
device.PrintNormal(consumerID,
            UnifiedPOS.POSPrinter.V1_2.PrinterStation.Receipt, "\u001b|bC 太字印刷")
device.SetDeviceEnabled(consumerID, False)
device.Release(consumerID)
device.CloseDevice(consumerID)
```

◆セッションの終了

サービスプロバイダとのセッションを終了します。

```
device.CloseSession(consumerID)
```

<Visual C#>

◆記述ファイル:

[アプリケーションで指定したファイル名].cs

◆ConsumerID を生成

ConsumerID は、デバイスを識別するために使用します。

```
private string consumerID = Guid.NewGuid().ToString();
```

※上記例は、GUID を生成して ID として使用しています。

◆メッセージ送信用チャネル生成

```
private UnifiedPOS.POSPrinter.V1_2.POSPrinter device = null;
// アプリケーション構成ファイルで設定した endpoint の contract と name を指定
ChannelFactory< UnifiedPOS.POSPrinter.V1_2.POSPrinter > factory =
      new ChannelFactory< UnifiedPOS.POSPrinter.V1_2.POSPrinter >("POSPrinterPort1");
// チャネルを生成
device = factory.CreateChannel();
```

◆セッションの開始

サービスプロバイダとのセッションを開始します。

```
device.OpenSession(consumerID, null);
```

※OpenSession 後、各メソッドの呼び出しが可能になります。

◆メソッド・プロパティの使用

printNormal メソッドを使用する場合、下記のように記述します。

```
device.OpenDevice(consumerID);
device.Claim(consumerID, 10000);
device.SetDeviceEnabled(consumerID, true);
device.PrintNormal(consumerID,
            UnifiedPOS.POSPrinter.V1_2.PrinterStation.Receipt, " 印刷するテキスト");
// エスケープシーケンスを使用し Bold 印刷
device.PrintNormal(consumerID,
            UnifiedPOS.POSPrinter.V1_2.PrinterStation.Receipt, "\u001b|bC 太字印刷");
device.SetDeviceEnabled(consumerID, false);
device.Release(consumerID);
device.CloseDevice(consumerID);
```

◆セッションの終了

サービスプロバイダとのセッションを終了します。

```
device.CloseSession(consumerID);
```

3.3.7　例外処理

Catch した例外から UposException 型の詳細情報を取得することで、エラーの原因を判断できます。

<Visual Basic .NET>

```
try
    ' デバイスオープン
    device.OpenDevice(consumerID)
Catch ex As FaultException(Of UnifiedPOS.POSPrinter.V1_2.UposException)
    ' Catch した例外から UposException 型の詳細情報を取得
    Dim ue As UnifiedPOS.POSPrinter.V1_2.UposException = ex.Detail
    ' ErrorCode を取得
    Dim code As UnifiedPOS.POSPrinter.V1_2.ErrorCode = ue.ErrorCode
    ' ErrorCodeExtended を取得
    Dim errorCodeEx As Integer = ue.ErrorCodeExtended
End Try
```

<Visual C#>

```
try {
    // デバイスオープン
    device.OpenDevice(consumerID);
} catch ( FaultException<UnifiedPOS.POSPrinter.V1_2.UposException> ex ) {
    // Catch した例外から UposException 型の詳細情報を取得
    UnifiedPOS.POSPrinter.V1_2.UposException ue = ex.Detail;
    // ErrorCode を取得
    UnifiedPOS.POSPrinter.V1_2.ErrorCode code = ue.ErrorCode;
    // ErrorCodeExtended を取得
    int errorCodeEx = ue.ErrorCodeExtended;
}
```

3.3.8　イベント処理-SelfHost 方式

サービスコンシューマ（アプリケーション）に用意したイベント受信サービスクラスに、サービスプロバイダからイベントを通知してもらう方式です。

<Visual Basic .NET>

手順	イベント処理-SelfHost 方式　Visual Basic.NET	
1	Visual Studio でイベント取得用サービスクラスを作成	ソリューションエクスプローラからプロジェクトを右クリックし、[追加]-[新しい項目]をクリックします。
2	VB のクラスを追加	ファイル名"POSPrinterEventService.vb"でクラスを追加します。

3	追加したファイル"POSPrinterEventService.vb"を編集します。以下を参考にしてください。
	
	◆POSPrinterEventService.vb
	Namespace WSPOSEventService
	  <ServiceBehavior(Namespace:="http://www.nrf-arts.org/UnifiedPOS/POSPrinterEvents/ ",
	  InstanceContextMode:=InstanceContextMode.Single)> _
	  public class POSPrinterEventService
	  ' イベントを取得したいカテゴリのイベントクラスを継承する
	  Implements UnifiedPOS.POSPrinterEvents.V1_2.POSPrinterEvent
	
	  ' ErrorEvent 発生時に呼ばれるメソッド
	  Public Function ErrorEvent(
	  ConsumerID As String, Source As String, EventID As Integer, TimeStamp As Date,
	  ErrorCode As UnifiedPOS.POSPrinterEvents.V1_2.ErrorCode,
	  ErrorCodeExtended As Integer,
	  ErrorLocus As UnifiedPOS.POSPrinterEvents.V1_2.ErrorLocus,
	  ErrorResponse As UnifiedPOS.POSPrinterEvents.V1_2.ErrorResponse) _
	  As UnifiedPOS.POSPrinterEvents.V1_2.ErrorResponse Implements
	  UnifiedPOS.POSPrinterEvents.V1_2.POSPrinterEvent.ErrorEvent
	  Return UnifiedPOS.POSPrinterEvents.V1_2.ErrorResponse.Clear
	  End Function
	
	  ' OutputComplateEvent 発生時に呼ばれるメソッド
	  Public Sub OutputCompleteEvent(
	  ConsumerID As String, Source As String, EventID As Integer,
	  TimeStamp As Date, OutputID As Integer) _
	  Implements
	  UnifiedPOS.POSPrinterEvents.V1_2.POSPrinterEvent.OutputCompleteEvent
	  End Sub
	
	  ' StatusUpdateEvent 発生時に呼ばれるメソッド
	  Public Sub StatusUpdateEvent(
	  ConsumerID As String, Source As String, EventID As Integer, TimeStamp As Date,
	  Status As Integer) _
	  Implements UnifiedPOS.POSPrinterEvents.V1_2.POSPrinterEvent.StatusUpdateEvent
	  End Sub
	
	  ' DirectIOEvent 発生時に呼ばれるメソッド
	  Public Function DirectIOEvent(
	  ConsumerID As String, Source As String, EventID As Integer, TimeStamp As Date,
	  EventNumber As Integer, Data As Integer, Obj As Object) _
	  As UnifiedPOS.POSPrinterEvents.V1_2.DirectIOData Implements
	  UnifiedPOS.POSPrinterEvents.V1_2.POSPrinterEvent.DirectIOEvent
	  Return New UnifiedPOS.POSPrinterEvents.V1_2.DirectIOData()
	  End Function
	
	  End Class
	End Namespace	

手順	イベント処理-SelfHost 方式　Visual Basic.NET

4	アプリケーション構成ファイル(App.config)を編集します。
	
	◆App.config
	`<configuration>`
	`  <system.serviceModel>`
	`  <services>`
	`  <!-- サービス名を指定（サービス名は[サービスクラスのnamespace]+[クラス名]） -->`
	`  <service name=" WSPOSEventService.POSPrinterEventService`
	`  behaviorConfiguration="WSPOSClient.POSPrinterEventServiceBehavior">`
	`  <host>`
	`  <baseAddresses>`
	`  <!—baseAddressにイベント取得用URLを指定 -->`
	`  <add baseAddress = "http://192.168.1.101:8001/POSPrinterEvent" />`
	`  </baseAddresses>`
	`  </host>`
	`  <!—イベント取得用サービスのendpoint情報を指定 -->`
	`  <endpoint address=""`
	`  binding="basicHttpBinding"`
	`  contract="UnifiedPOS.POSPrinterEvents.V1_2.POSPrinterEvent"`
	`  bindingNamespace=http://www.nrf-arts.org/UnifiedPOS/POSPrinterEvents/`
	`  />`
	`  </service>`
	`  </services>`
	`  </system.serviceModel>`
	`</configuration>`

5	アプリケーションソースにイベント取得用のサービスホストを生成する処理を記述します。
	
	`Dim posPrinterEventServiceHost As ServiceHost`
	`' イベント取得用のサービスの開始`
	`posPrinterEventServiceHost =`
	`  New ServiceHost(GetType(WSPOSEventService.POSPrinterEventService))`
	
	`‘ OpenSession の引数で Endpoint を渡す`
	`device.OpenSession(consumerID, "http://192.168.1.101:8001/POSPrinterEvent")`
	
	`' メソッドの呼び出しなどの処理`
	
	`'イベント取得用サービス終了`
	`posPrinterEventServiceHost.Close()`

<Visual C#>

手順	イベント処理-SelfHost 方式　Visual C#	
1	Visual Studio でイベント取得用サービスクラスを作成	ソリューションエクスプローラからプロジェクトを右クリックし、[追加]-[新しい項目]をクリックします。
2	C#のクラスを追加	ファイル名”POSPrinterEventService.cs”でクラスを追加します。
3	追加したファイル”POSPrinterEventService.cs”を編集します。以下を参考にしてください。 **◆POSPrinterEventService.cs** (コード参照)	

```
namespace WSPOSEventService {
  [ServiceBehavior (Namespace = "http://www.nrf-arts.org/UnifiedPOS/
    POSPrinterEvents/",InstanceContextMode = InstanceContextMode.Single)]
    // イベントを取得したいカテゴリのイベントクラスを継承する
    public class POSPrinterEventService: UnifiedPOS.POSPrinter.V1_2.POSPrinterEvent
    {
      // ErrorEvent 発生時に呼ばれるメソッド
      public virtual UnifiedPOS.POSPrinterEvents.V1_2.ErrorResponse ErrorEvent(
              string ConsumerID, string Source, int EventID, DateTime TimeStamp,
              UnifiedPOS.POSPrinterEvents.V1_2.ErrorCode ErrorCode,
              int ErrorCodeExtended,
              UnifiedPOS.POSPrinterEvents.V1_2.ErrorLocus ErrorLocus,
              UnifiedPOS.POSPrinterEvents.V1_2.ErrorResponse ErrorResponse)
      {
         return UnifiedPOS.POSPrinterEvents.V1_2.ErrorResponse.Clear;
      }

      // OutputCompeteEvent 発生時に呼ばれるメソッド
      public virtual void OutputCompleteEvent(
              string ConsumerID, string Source, int EventID, DateTime TimeStamp, int OutputID)
      {
      }

      // StatusUpdateEvent 発生時に呼ばれるメソッド
      public virtual void StatusUpdateEvent(
              string ConsumerID, string Source, int EventID, DateTime TimeStamp, int Status)
      {
      }

      // DirectIOEvent 発生時に呼ばれるメソッド
      public virtual UnifiedPOS.POSPrinterEvents.V1_2.DirectIOData DirectIOEvent(
              string ConsumerID, string Source, int EventID, DateTime TimeStamp, int
               EventNumber, int Data, object Obj)
      {
        return new UnifiedPOS.POSPrinterEvents.V1_2.DirectIOData()
                    { Data = Data, Obj = Obj };
      }
    }
 }
```

手順	イベント処理-SelfHost 方式　Visual C#

4	アプリケーション構成ファイル(App.config)を編集します。
	
	◆App.config
	`<configuration>`
	`  <system.serviceModel>`
	`  <services>`
	`  <!-- サービス名を指定（サービス名は[サービスクラスのnamespace]+[クラス名]） -->`
	`  <service name=" WSPOSEventService.POSPrinterEventService`
	`  behaviorConfiguration="WSPOSClient.POSPrinterEventServiceBehavior">`
	`  <host>`
	`  <baseAddresses>`
	`  <!—baseAddressにイベント取得用URLを指定 -->`
	`  <add baseAddress = "http://192.168.1.101:8001/POSPrinterEvent" />`
	`  </baseAddresses>`
	`  </host>`
	`  <!—イベント取得用サービスのendpoint情報を指定 -->`
	`  <endpoint address=""`
	`  binding="basicHttpBinding"`
	`  contract="UnifiedPOS.POSPrinterEvents.V1_2.POSPrinterEvent"`
	`  bindingNamespace=http://www.nrf-arts.org/UnifiedPOS/POSPrinterEvents/`
	`  />`
	`  </service>`
	`  </services>`
	`  </system.serviceModel>`
	`</configuration>`

5	アプリケーションソースにイベント取得用のサービスホストを生成する処理を記述します。
	
	`private ServiceHost posPrinterEventServiceHost;`
	`posPrinterEventServiceHost =`
	`  new ServiceHost(typeof(WSPOSEventService.POSPrinterEventService));`
	
	`// イベント取得用のサービスの開始`
	`posPrinterEventServiceHost.Open();`
	
	`// OpenSession の引数で Endpoint を渡す`
	`device.OpenSession(consumerID, "http://192.168.1.101:8001/POSPrinterEvent");`
	
	`// メソッドの呼び出しなどの処理`
	
	`//イベント取得用サービス終了`
	`posPrinterEventServiceHost.Close();`

3.3.9　イベント処理-LongPolling 方式

サービスコンシューマ（アプリケーション）からサービスプロバイダに、イベント取得リクエストを送信し、そのレスポンスとしてイベントを通知してもらう方式です。

<Visual Basic .NET>

手順	イベント処理-LongPolling 方式　Visual Basic.NET

1	LongPolling用のスレッドを動作させ、サービスプロバイダにイベント取得リクエストを送る処理を追加します。
	
	`Dim longPollingThread As Threading.Thread`
	`'セッションオープン`
	`device.OpenSession(consumerID, Nothing)`
	`'LongPolling用スレッド作成・開始`
	`longPollingThread = New Threading.Thread(AddressOf POSPrinterEvent_LongPolling)`
	`longPollingThread.IsBackground = True`
	`longPollingThread.Start()`

2	POSPrinterEvent_LongPollingメソッドにイベント取得時の処理を記載します。以下を参考にしてください。
	
	`Private Sub POSPrinterEvent_LongPolling()`
	`  While Not consumerID Is Nothing`
	`    Try`
	`      Dim deviceEvent As UnifiedPOS.POSPrinter.V1_2.POSPrinterEvent`
	`      ' イベントポーリングを実行（イベント取得リクエスト発行）`
	`      deviceEvent = device.PollForUPOSEvent(consumerID)`
	`      If deviceEvent Is Nothing Then`
	`        Continue While`
	`      End If`
	
	`      Dim res As New UnifiedPOS.POSPrinter.V1_2.POSPrinterEventResponse`
	
	`      ' 受信したイベントごとの処理を実行`
	`      If Not deviceEvent.ErrorEvent Is Nothing Then`
	`        ' ErrorEvent 取得時の処理`
	`        Dim ev As UnifiedPOS.POSPrinter.V1_2.ErrorEvent = deviceEvent.ErrorEvent`
	`        res.ErrorEventResponse = new UnifiedPOS.POSPrinter.V1_2.ErrorEventResponse()`
	`        res.ErrorEventResponse.EventID = ev.EventID`
	
	`      ElseIf Not deviceEvent.OutputCompleteEvent Is Nothing Then`
	`        ' OutputCompleteEvent 取得時の処理`
	`        res.OutputCompleteEventResponse = new`
	`                        UnifiedPOS.POSPrinter.V1_2.OutputCompleteEventResponse()`
	`        res.OutputCompleteEventResponse.EventID = ev.EventID`
	
	`      ElseIf Not deviceEvent.StatusUpdateEvent Is Nothing Then`
	`        ' StatusUpdateEvent 取得時の処理`
	`        Dim ev As UnifiedPOS.POSPrinter.V1_2.StatusUpdateEvent =`
	`                                        deviceEvent.StatusUpdateEvent`
	`        res.StatusUpdateEventResponse = new`
	`                        UnifiedPOS.POSPrinter.V1_2.StatusUpdateEventResponse()`
	`        res.StatusUpdateEventResponse.EventID = ev.EventID`
	
	`      ElseIf Not deviceEvent.DirectIOEvent Is Nothing Then`
	`        ' DirectIOEvent 取得時の処理`
	`        Dim ev As UnifiedPOS.POSPrinter.V1_2.DirectIOEvent = deviceEvent.DirectIOEvent`
	`        res.DirectIOEventResponse = new UnifiedPOS.POSPrinter.V1_2.DirectIOEventResponse()`
	`        res.DirectIOEventResponse.EventID = ev.EventID`
	
	`      End If`

手順	イベント処理-LongPolling 方式　Visual Basic.NET

2
続き
	``
	        ' イベントへの応答 (WS-POSサービスプロバイダ への応答)
	        While (True)
	          Try
	            device.SetEventResponse(consumerID, res)
	            ' 応答が正常に完了したら、次のイベント取得のためPollForUPOSEventに戻る
	            Exit While
	          Catch ex As TimeoutException
	            ' イベントへの応答がオフラインまたはタイムアウトの場合、イベント応答を再試行
	            Continue While
	
	          Catch ex As FaultException(Of UnifiedPOS.POSPrinter.V1_2.UposException)
	            If ex.Detail.ErrorCode = UnifiedPOS.POSPower.V1_2.ErrorCode.Illegal Then
	              ' ILLEGAL が帰ってきた場合は、SetEventResponse は既に受信されていたので、
	              ' 再度 PollForUPOSEvent に戻る
	              Exit While
	            Else
	              Throw
	            End If
	          End Try
	        End While
	
	      Catch ex As TimeoutException
	        ' イベントポーリングがオフラインまたはタイムアウトの場合、PollForUPOSEventを再試行
	        Continue While
	      End Try
	    End While
	End Sub
	``

<Visual C#>

手順	イベント処理-LongPolling 方式　Visual C#
1	LongPolling用のスレッドを動作させ、サービスプロバイダにイベント取得リクエストを送る処理を追加します。
2	POSPrinterEvent_LongPollingメソッドにイベント取得時の処理を記載します。以下を参考にしてください。

```
private Thread longPollingThread;
// セッションオープン
device.OpenSession(consumerID, null);
// LongPolling用スレッド作成・開始
longPollingThread = new Thread(new ThreadStart(POSPrinterEvent_LongPolling));
longPollingThread.IsBackground = true;
longPollingThread.Start();
```

```
void POSPrinterEvent_LongPolling()
{
   while (consumerID != null)
   {
      try
      {
         UnifiedPOS.POSPrinter.V1_2.POSPrinterEvent deviceEvent;
         // イベントポーリングを実行
         deviceEvent = device.PollForUPOSEvent(consumerID);

         if (deviceEvent == null)
         {
            continue; // イベントがなかった場合、再度ポーリング
         }

         UnifiedPOS.POSPrinter.V1_2.POSPrinterEventResponse res =
                        new UnifiedPOS.POSPrinter.V1_2.POSPrinterEventResponse();

         // 受信したイベントごとの処理を実行
         if (deviceEvent.ErrorEvent != null)
         {
            // ErrorEvent 取得時の処理
            UnifiedPOS.POSPrinter.V1_2.ErrorEvent ev = deviceEvent.ErrorEvent;
            res.ErrorEventResponse = new UnifiedPOS.POSPrinter.V1_2.ErrorEventResponse();
            res.ErrorEventResponse.EventID = ev.EventID;
         }
         else if (deviceEvent.OutputCompleteEvent != null)
         {
            // OutputCompleteEvent 取得時の処理
            UnifiedPOS.POSPrinter.V1_2.OutputCompleteEvent ev =
                                                   deviceEvent.OutputCompleteEvent;
            res.OutputCompleteEventResponse = new
                              UnifiedPOS.POSPrinter.V1_2.OutputCompleteEventResponse();
            res.OutputCompleteEventResponse.EventID = ev.EventID;
         }
         else if (deviceEvent.StatusUpdateEvent != null)
         {
            // StatusUpdateEvent 取得時の処理
            UnifiedPOS.POSPrinter.V1_2.StatusUpdateEvent ev =
                                                   deviceEvent.StatusUpdateEvent;
            res.StatusUpdateEventResponse = new
                                UnifiedPOS.POSPrinter.V1_2.StatusUpdateEventResponse();
            res.StatusUpdateEventResponse.EventID = ev.EventID;
         }
         else if (deviceEvent.DirectIOEvent != null)
         {
            // DirectIOEvent 取得時の処理
            UnifiedPOS.POSPrinter.V1_2.DirectIOEvent ev = deviceEvent.DirectIOEvent;
            res.DirectIOEventResponse = new
                                   UnifiedPOS.POSPrinter.V1_2.DirectIOEventResponse();
            res.DirectIOEventResponse.EventID = ev.EventID;
         }
```

手順	イベント処理-LongPolling 方式　Visual C#

2
続き
	// イベントへの応答(WS-POSサービスプロバイダへの応答)
	while (true)
	{
	try
	{
	device.SetEventResponse(consumerID, res);
	// 応答が正常に完了したら、次のイベントを取得するためPollForUPOSEventに戻る
	break;
	}
	catch (TimeoutException)
	{
	// イベントの返信がオフラインで帰ってきた場合、PollEventではなく、再度SetEventResponseを行う
	continue;
	}
	catch (FaultException<UnifiedPOS.POSPrinter.V1_2.UposException> ex)
	{
	if (ex.Detail.ErrorCode == UnifiedPOS.POSPrinter.V1_2.ErrorCode.Illegal)
	{
	// ILLEGALが帰ってきた場合は、SetEventResponseは既に受信されていたので、
	// 再度PollForUPOSEvent に戻る
	break;
	}
	else
	{
	throw;
	}
	}
	}
	}
	catch (TimeoutException)
	{
	// イベントポーリングがオフラインまたはタイムアウトの場合、PollForUPOSEventを再試行
	continue;
	}
	}
	}

3.3.10 KeepAlive 処理

OpenSession()後、アプリケーションとデバイス間で通信が無い時間が、サービスプロバイダ側で設定されているタイムアウト時間を超えると、デバイスとのコネクションが切断されます。KeepAlive はこれを防止するための仕組みです。

<Visual Basic .NET>

手順	KeepAlive 処理　Visual Basic.NET

1	タイマーを使用し、KeepAlive処理を行う処理を記載します。以下を参考にしてください。
	
	Dim timer As System.Windows.Forms.Timer
	timer = New Timer()
	
	AddHandler timer.Tick, New EventHandler(AddressOf timer_Tick)
	
	' アプリケーション開始時に KeepAlive 用のタイマーを開始
	Dim providerSessionTimeout As Integer = device.GetProviderSessionTimeout(consumerID)
	' Service Provider で設定されているタイムアウト時間を取得
	'ここでは、サービスプロバイダから取得したタイムアウト時間の半分を KeepAlive の時間とします
	timer.Interval = ((providerSessionTimeout / 2) * 1000)
	' タイマー開始
	timer.Start()
	
	Private Sub timer_Tick(sender As Object, e As EventArgs)
	' WS-POS KeepAlive
	  If Not consumerID Is Nothing Then
	  Try
	  device.KeepAlive(consumerID)
	  Catch ex As TimeoutException
	  ' タイムアウト処理
	  timer.Stop()
	  ' KeepAlive がタイムアウトになった場合（オフライン）
	  ' チャネル情報、ConsumerIDをnullにします
	  consumerID = Nothing
	  device = Nothing
	  ' LongPollingでイベント取得時は、LongPollingスレッドを停止させます
	  ' この後、再度Openし直してください
	  End Try
	  End If
	End Sub

<Visual C#>

手順	KeepAlive 処理　Visual C#

1	タイマーを使用し、KeepAlive処理を行う処理を記載します。以下を参考にしてください。
	
	private System.Windows.Forms.Timer timer;
	
	timer.Tick += new System.EventHandler(timer_Tick);
	
	// アプリケーション開始時に KeepAlive 用のタイマーを開始する
	int providerSessionTimeout = device.GetProviderSessionTimeout(consumerID);
	// Service Provider で設定されているタイムアウト時間を取得する
	// ここでは、サービスプロバイダから取得したタイムアウト時間の半分を KeepAlive の時間とします
	timer.Interval = providerSessionTimeout / 2 * 1000;
	// タイマーを開始
	timer.Start();
	
	private void timer_Tick(object sender, EventArgs e) {
	  // WS-POS KeepAlive
	  if (consumerID != null) {
	  try {
	  device.KeepAlive(consumerID);
	  } catch (TimeoutException) {
	  // タイムアウト処理
	  timer.Stop();
	  // KeepAlive がタイムアウトになった場合（オフライン）
	  // チャネル情報、ConsumerIDをnullにします
	  consumerID = null;
	  device = null;
	  // LongPollingでイベント取得時は、LongPollingスレッドを停止させます
	  // この後、再度Openし直してください
	  }
	  }
	}

3.4　OPOS のメソッド・プロパティについて

生成したチャネルのメンバーとして、OPOS と同様のメソッド・プロパティが使用できます。
メソッド・プロパティについては、OPOS の APG(アプリケーションプログラマーズガイド)を参照してください。

*FP-1100 OPOS の場合は下記の APG を参照してください。
「FP-1100 POSPrinter, CashDrawer シリアル・USB・LAN インターフェース対応
OPOS-OCX ドライバ　アプリケーションプログラマーズガイド」
(FP1100_OPOS_APG_ja.pdf)

3.5　WS-POS で使用できないメソッド・プロパティ

下記の OPOS のプロパティ・メソッドは WS-POS で使用できません。
コンシューマおよびプロバイダが停止する場合がありますので、使用しないでください。

◆メソッド
- ・CompareFirmwareVersion
- ・ResetStatistics
- ・PrintMemoryBitmap
- ・DrawRuledLine

◆プロパティ
- ・BinaryConversion
- ・OpenResult
- ・ResultCode
- ・ResultCodeExtended
- ・CapRecRuledLine
- ・CapSlpRuledLine

4. 注意事項

- PC を休止またはスリープ状態にする場合
  コンシューマ実行中に PC を休止またはスリープ状態にすることは推奨しません。
  通信がない時間が続いた場合、タイムアウトとなり、プロバイダとのセッションが切断する可能性があります。その場合、再度アプリケーションを起動してください。復旧しない場合はサーバー側でサービスプロバイダの再起動を行ってください。

- SendTimeout 値の変更について
  WCF サービスのバインディングプロパティの SendTimeout の既定値は”00:01:00”です。コンシューマ側でバッファサイズが大きいデータを送信し、1分間処理が返ってこない場合、タイムアウトとなり、アプリケーション内で例外が発生する可能性があります。その場合、サービスコンシューマ側で以下のように SendTimeout 値を変更してください。

  サービスコンシューマ設定ファイルに以下の太字部分を追加します。

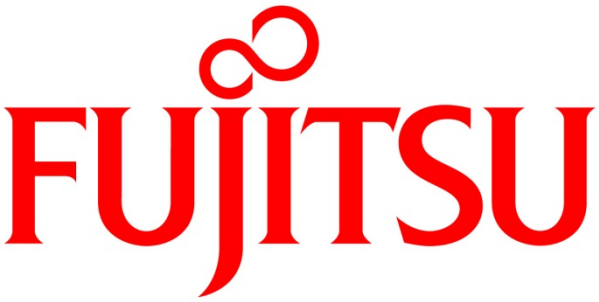
```
<client>
  <endpoint address="http://192.168.1.120:8087/POSPrinter.svc/POSPrinter1"
            binding="basicHttpBinding"
            bindingConfiguration="longTimeBinding"
            contract="UnifiedPOS.POSPrinter.V1_2.POSPrinter"
            name="POSPrinterPort1"/>
</client>
<bindings>
  <basicHttpBinding>
     <binding name="longTimeoutBinding" SendTimeout="00:01:00"/>    ・・・※
  </basicHttpBinding >
</bindings >
```

  ※ご使用の環境により、適切なタイムアウト値を設定してください。

- WS-POS 参照実装の 64bit 環境での使用について
  現在提供されている WS-POS 参照実装のプロバイダは 64bit 環境で動作しません。
  ″WcfServiceHostWindowsService35.exe″が「Any CPU」でビルドされているため、サービスとして動作させると 64bit 動作となり、32bit である CCO にアクセスできないためです。64bit 環境で動作させるには 32bit(x86)でビルドする必要があります。

5. 改訂履歴

Revesion	更新日
1.0.0.0	初版(2014.09.12)


FUJITSU