EPSON
EXCEED YOUR VISION

WS-POS
アプリケーション開発ガイド

概要

WS-POS対応アプリケーションの開発環境について説明します。

Visual C# で開発

Visual C#環境での開発情報について説明します。

Visual Basic .NET で開発

Visual Basic .NET環境での開発情報について説明します。

JavaScript で開発

JavaScript環境での開発情報について説明します。

M00067100
Rev.A

ご注意

- 本書の内容の一部または全部を無断で転載、複写、複製、改ざんすることは固く お断りします。
- 本書の内容については、予告なしに変更することがあります。最新の情報はお問い合わせく ださい。
- 本書の内容については、万全を期して作成いたしましたが、万一ご不審な点や誤り、記載もれなど、お気づきの点がありましたらご連絡く ださい。
- 運用した結果の影響については、上項にかかわらず責任を負いかねますのでご了承く ださい。
- 本製品がお客様により不適切に使用されたり、本書の内容に従わずに取り扱われたり、またはエプソンおよびエプソン指定の者以外の第三者により修理・変更されたことなどに起因して生じた損害などにつきましては、責任を負いかねますのでご了承く ださい。
- エプソン純正品およびエプソン品質認定品以外のオプションまたは消耗品を装着してトラブルが発生した場合には、責任を負いかねますのでご了承く ださい。

商標について

EPSON® は、セイコーエプソン株式会社の登録商標です。

Microsoft®、Windows®、Internet Explorer®、Silverlight®、Visual Studio®、Visual Basic®、Visual C#® は、米国 Microsoft Corporation の米国およびその他の国における商標または登録商標です。

JavaScript® は、Oracle Corporation の米国およびその他の国における商標または登録商標です。

安全のために

記号の意味

本書では以下の記号が使われています。それぞれの記号の意味をよく理解してから製品を取り扱ってください。

	ご使用上、必ずお守りいただきたいことを記載しています。この表示を無視して誤った取り扱いをすると、製品の故障や動作不良の原因になる可能性があります。
	補足説明や知っておいていただきたいことを記載しています。

使用制限

本製品を航空機・列車・船舶・自動車などの運行に直接関わる装置・防災防犯装置・各種安全装置など機能・精度などにおいて高い信頼性・安全性が必要とされる用途に使用される場合は、これらのシステム全体の信頼性および安全維持のためにフェールセーフ設計や冗長設計の措置を講じるなど、システム全体の安全設計にご配慮いただいた上で当社製品をご使用いただくようお願いいたします。

本製品は、航空宇宙機器、幹線通信機器、原子力制御機器、医療機器など、きわめて高い信頼性・安全性が必要とされる用途への使用を意図しておりませんので、これらの用途には本製品の適合性をお客様において十分ご確認のうえ、ご判断ください。

本書について

本書の目的

本書は、WS-POS(Web Services for Point of Service) を利用したアプリケーションの開発、設計に必要な情報を開発技術者に提供することを、その目的としています。

本書の構成

本書は次のように構成されています。

もくじ

概要

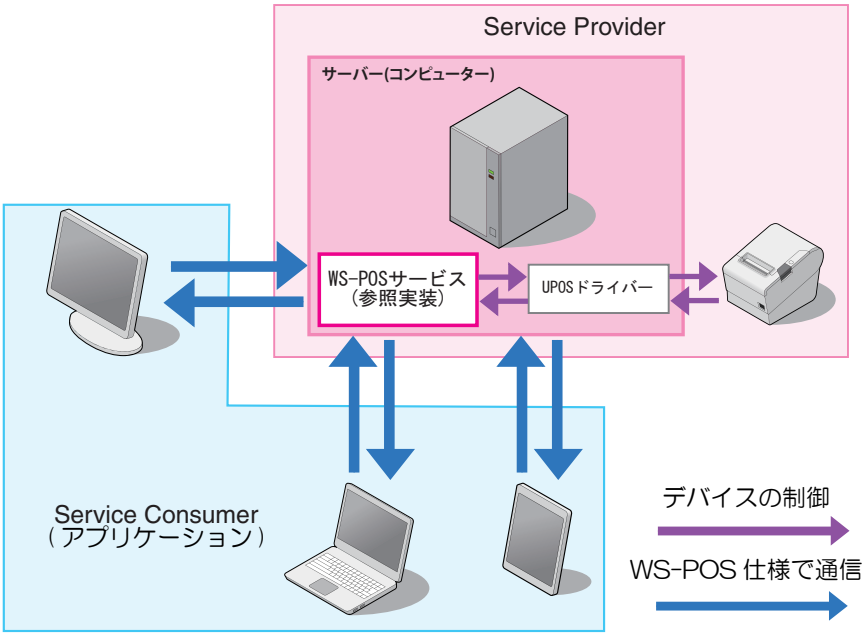
本章では、WS-POS 対応アプリケーションの開発環境について説明しています。

Web Services for Point of Service

Web Services for Point of Service(WS-POS) は、Web ベースの POS システムでデバイスを制御します。

WS-POS の仕様書、参照実装は、下記 URL を参照してください。
http://www.nrf-arts.org/content/unifiedpos

WS-POS の構成



- Service Provider (サービスプロバイダー) とは、WS-POS サービスを提供する側を指します。
- Service Consumer (サービスコンシューマー) とは、WS-POS サービスを利用する側を指します。本書は、Service Consumer で利用する WS-POS 対応アプリケーションの開発情報について説明しています。

開発環境

開発言語

本書では、以下の言語の WS-POS 対応アプリケーション開発方法について説明しています。

❏ Microsoft Visual Basic .NET

❏ Microsoft Visual C#

❏ JavaScript

WS-POS サービスを提供する環境によって、使用できる言語が異なります。

WS-POS サービス環境	使用できる開発言語
Windows サービス	• Microsoft Visual Basic .NET • Microsoft Visual C#
Internet Information Services	• JavaScript

WS-POS サービスの動作環境については、WS-POS 環境セットアップガイドを参照してください。

WS-POS サービス（参照実装）

❏ WS-POS 参照実装 Version 1.0

以下から入手できます。(2013 年 10 月 1 日時点）
http://www.nrf-arts.org/content/unifiedpos

提供物

マニュアル

❏ WS-POS アプリケーション開発ガイド（本書）

❏ WS-POS 環境セットアップガイド

ドライバー

❏ EPSON OPOS ADK for .NET

ダウンロード

提供物は、下記エプソン販売ホームページからダウンロードできます。

http://www.epson.jp/support/sd/

Visual C# で開発

本章では、Visual C#(Windows Communication Foundation) 環境での、アプリケーション開発方法について説明しています。

Visual C# での開発

本章では、以下について説明しています。

本章のアプリケーション環境

本章では、以下の環境を想定して、説明しています。

項目	環境設定
Service Provider の IP アドレス : ポート番号	192.168.1.100:8087
イベント取得用の IP アドレス : ポート番号	192.168.1.102:8001
デバイスカテゴリー	POSPrinter
サービスクラス	UnifiedPOS.POSPrinter.V1_2.POSPrinter
Namespace	http://www.nrf-arts.org/UnifiedPOS/POSPrinter/
POSPrinter サービスのベースアドレス	http://192.168.1.100:8087/POSPrinter.svc
POSPrinter サービスのエンドポイント	POSPrinter1
アプリケーション用プログラムファイル名	Application.cs
開発環境	Microsoft Visual Studio 2010

アプリケーションプロジェクトの設定

新規プロジェクトの作成

Visual Studio で新規プロジェクトを作成する際、C# の Windows アプリケーションで作成します。

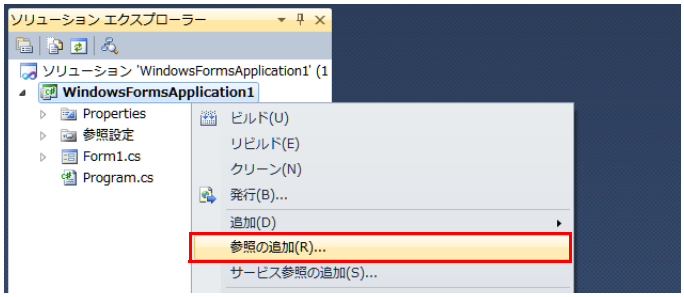
- 既存プロジェクトに追加する場合、新規プロジェクトを作成する必要はありません。
- コンソール版や Windows フォーム版でも作成できます。

WSPOSContract の追加

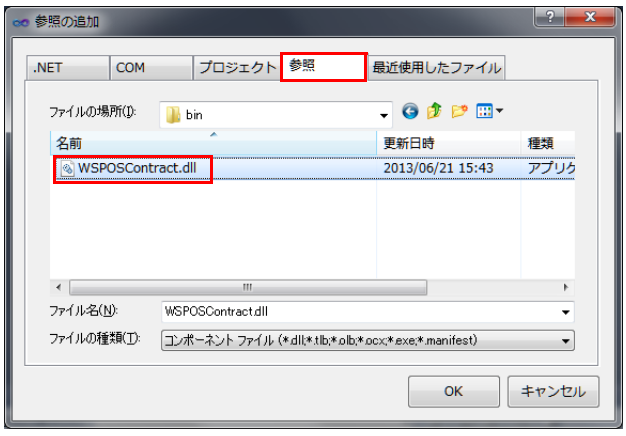
以下の手順で、WSPOSContract を追加します。

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ーに保存します。

2 Visual Studio で、ソリューションエクスプローラー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、手順 1 で保存した〝WSPOSContract.dll〟を選択し、[OK] をクリックします。

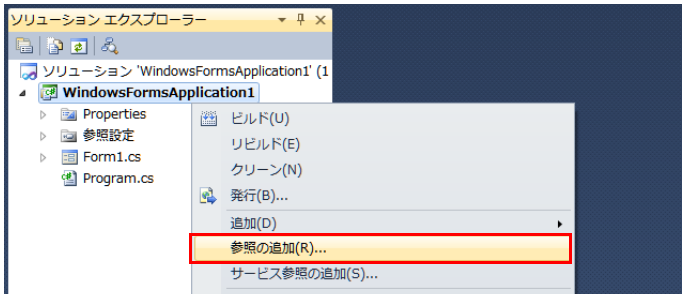


4 ソリューションエクスプローラーの参照設定に、〝WSPOSContract〟が追加されていることを確認します。

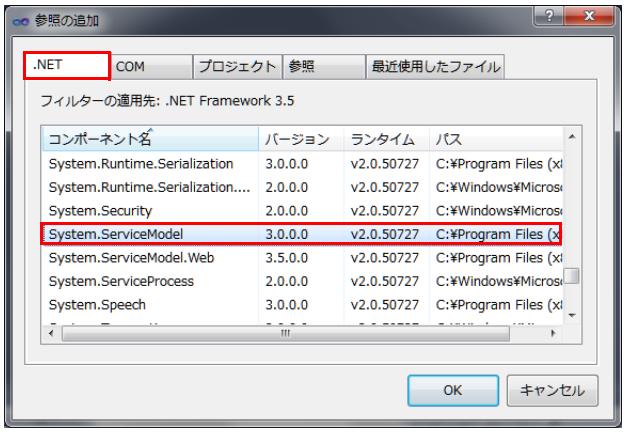
System.ServiceModel の追加

以下の手順で、System.ServiceModel を追加します。

1 Visual Studio で、ソリューションエクスプローラー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″ を選択し、[OK] をクリックします。



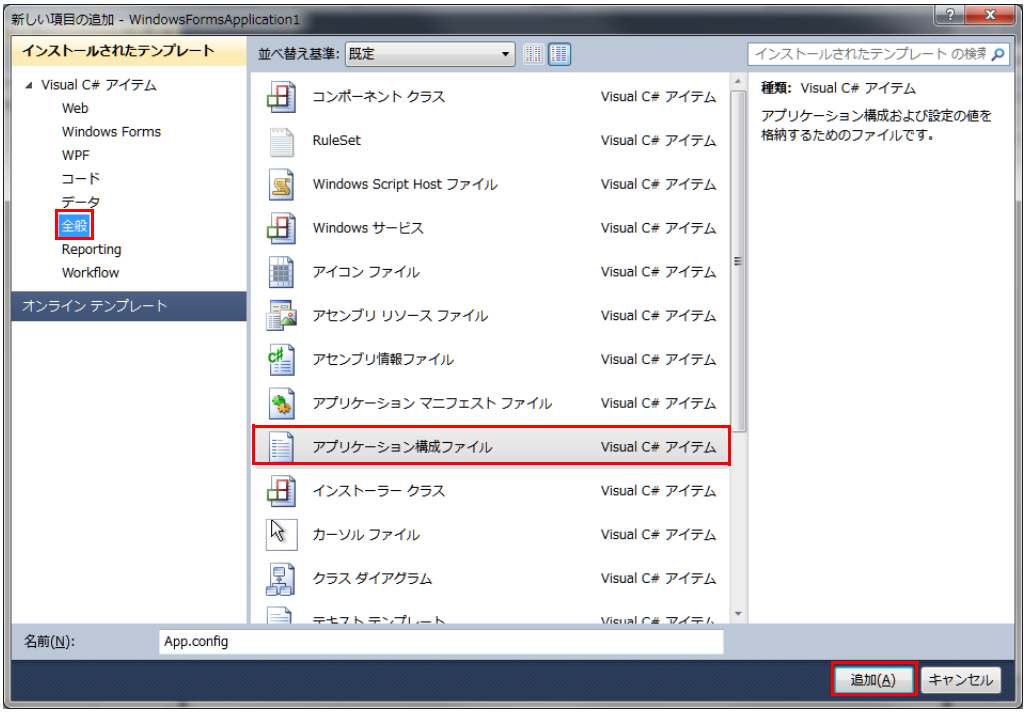
3 ソリューションエクスプローラーの参照設定に、″System.ServiceModel″ が追加されていることを確認します。

アプリケーション構成ファイルの追加

以下の手順で、アプリケーション構成ファイルを追加します。

1 Visual Studioで、ソリューションエクスプローラー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 [VisualC# 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 ソリューションエクスプローラーに〝App.config〟が追加されていることを確認します。

アプリケーション構成ファイルの追加

アプリケーション構成ファイル（App.config ファイル）に以下を記述してください。

〈App.config ファイル〉

```
<?xml version="1.0"?>
<configuration>
  ❏ Service Provider との接続情報を記述 (p.13)
  <system.serviceModel>
    <client>
      <endpoint address="http://192.168.1.100:8087/POSPrinter.svc/POSPrinter1"
                binding="basicHttpBinding"
                contract="UnifiedPOS.POSPrinter.V1_2.POSPrinter"
                name="POSPrinterPort1" />
    </client>
  </system.serviceModel>

  ❏ 通信ポートの最大同時オープン数を設定 (p.14)
  <system.net>
    <connectionManagement>
      <add address = "*" maxconnection = "48" />
    </connectionManagement>
  </system.net>

</configuration>
```

アプリケーション構成ファイルの詳細については、下記ウェブサイトを参照してください。
http://msdn.microsoft.com/ja-jp/library/ms733932.aspx
（2013 年 10 月 1 日時点）

Service Provider との接続情報を記述

Service Provider との接続情報を、〈endpoint〉に記述します。
属性の address、contract、binding に、Service Provider の情報と同じ情報を記述します。

〈endpoint〉の属性	説明
address	使用する Service Provider の URL を指定します。 Service Provider で設定した URL と同じアドレスを指定します。
binding	Service Provider で設定した binding を指定します。 本章では、"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名を指定します。
name	接続情報を識別するための名前を指定します。 Service Consumer が Service Provider との接続をする際に、アプリケーション構成ファイルの接続情報を取得するために使用します。

通信ポート の最大同時オープン数を設定

アプリケーションが通信ポートを開く際の、最大同時オープン数を設定します。
スレッドごとに通信ポートを開くため、スレッドの数に応じて、最大同時オープン数を指定します。
〈system.net〉タグの〈connectionManagement〉タグ内の maxconnection の値を、必要に応じて変更します。

- ここでは ″48″ で説明しています。お客様の環境に合わせて設定してください。
- サンプルアプリケーションでは、以下のスレッドを同時に使用します。通信のためのポート が、最低限 3 つ必要です。
  * メインスレッド
  * KeepAlive 用スレッド
  * Polling 用スレッド

Service Providerとの接続

Service Provider との接続に必要な処理を、アプリケーションファイル (Application.cs) に記述します。
以下の処理が必要です。

1. メッセージを送信するためのチャネルを生成 (15 ページ)
2. ConsumerID を取得 (15 ページ)
3. OpenSession (16 ページ)
4. CloseSession (16 ページ)

メッセージを送信するためのチャネルを生成

メッセージを送信するためのチャネルを生成します。

```
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、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();
```

ConsumerID を取得

ConsumerID を取得します。
ConsumerID は、デバイスを識別するための識別子として使用します。

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

上記では、"GUID" を生成して ID として使用しています。ID は、開発の際に、一意に識別できるものを使用してください。

OpenSession

Service Provider とのセッションを開始します。
OpenSession を呼び出した後に、各メソッドを呼び出すことができます。

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

CloseSession

Service Provider とのセッションを終了します。
各メソッドの処理が終了したら、このメソッドを呼び出し、Service Provider とのセッションを終了してください。

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

メソッドの呼び出し方法

ここでは、printNormal メソッドを使う場合の、一連の流れを説明しています。

メッセージを送信するためのチャネルを生成 (p.15) で生成したチャネル (device) に対して、メソッドを指定することで、各メソッドを呼び出せます。

WS-POS では、各メソッドを呼び出す際に、必ず consumerID が必要になります。
サービス側は、consumerID により、デバイスとコンシューマの関連付けをします。

〈Application.cs ファイル〉

❏ セッション開始
```
device.OpenSession(consumerID, null);
```
❏ デバイスの Open 処理
```
device.OpenDevice(consumerID);
```
❏ Claim 処理
```
device.Claim(consumerID, 10000);
```
❏ Enabled 処理
```
device.SetDeviceEnabled(consumerID, true);
```
❏ PrintNormal 処理
```
device.PrintNormal(consumerID, UnifiedPOS.POSPrinter.V1_2.PrinterStation.Receipt,
                   "ここに印刷するテキストを入力");
```
❏ Disabled 処理
```
device.SetDeviceEnabled(consumerID, false);
```
❏ Release 処理
```
device.Release(consumerID);
```
❏ デバイスの Close 処理
```
device.CloseDevice(consumerID);
```
❏ セッションの Close 処理
```
device.CloseSession(consumerID);
```

エラー（例外）の取得方法

各メソッドを try,catch で囲み、例外をキャッチすることでエラーを取得します。

〈Application.cs ファイル〉

```
try {
// デバイスオープン
  device.OpenDevice(consumerID);
```

❑ Catch した例外から、UposException 型の詳細情報を取得します。

```
} catch ( FaultException<UnifiedPOS.POSPrinter.V1_2.UposException> ex ) {
```

❑ ErrorCode、ErrorCodeExtended を取得します。

```
  UnifiedPOS.POSPrinter.V1_2.UposException ue = ex.Detail;
  UnifiedPOS.POSPrinter.V1_2.ErrorCode code = ue.ErrorCode;
  int errorCodeEx = ue.ErrorCodeExtended;
}
```

イベント の取得方法

イベントの取得方法は、以下の方法があります。

- SelfHost 方式 (p.20)

  Service Consumer(アプリケーション) 側でイベントを受け取るためのサービスクラスを用意し、このサービスクラスに対して、Service Provider 側からイベントを通知してもらう方法です。

  ここでは、以下の環境を想定して設定します。

項目	設定
イベント取得用の IP アドレス : ポート番号	192.168.1.102:8001
イベント取得用のサービスクラス	POSPrinterEventService
イベント取得用サービスのベースアドレス	http://192.168.1.102:8001/POSPrinterEvent
サービスクラスの namespace	WSPOSEventService
デバイスカテゴリー	POSPrinter
サービス用プログラムファイル名	POSPrinterEventService.cs

- LongPolling 方式 (p.22)

  Service Consumer(アプリケーション) から Service Provider に対してイベント取得の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法です。

SelfHost 方式

以下の方法で、イベントを取得するプログラミングをします。

1 Visual Studio でイベント取得用サービスクラスを作成します。
プロジェクトを選択し、右クリックメニューから [追加]-[新しい項目] を選択し、C# のクラスを追加します。ファイル名は、″POSPrinterEventService.cs″ にします。

2 追加したファイル POSPrinterEventService.cs を編集します。
デバイスカテゴリーごとに必要なメソッドが異なるので、必要に応じて追加、削除します。
以下を参考にして、編集してください。

〈POSPrinterEventService.cs ファイル〉

```
namespace WSPOSEventService {
```

❑ イベント取得用のサービスを作成するための宣言

```
  [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 が発生した際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;
      }
```

❑ OutputCompleteEvent が発生した際に呼ばれるメソッド

```
    public virtual void OutputCompleteEvent(
            string ConsumerID, string Source, int EventID, DateTime TimeStamp, int
            OutputID)
      {
      }
```

❑ StatusUpdeteEvent が発生した際に呼ばれるメソッド

```
    public virtual void StatusUpdateEvent(
            string ConsumerID, string Source, int EventID, DateTime TimeStamp, int
            Status)
      {
      }
```

❑ DirectIOEvent が発生した際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 };
      }

  }
}
```

3 アプリケーション構成ファイル (App.config) を編集します。

以下を編集します。

〈App.config ファイル〉

```
<configuration>
  <system.serviceModel>
    <services>
```

❑ サービス名を指定する。

サービス名は、イベント取得用のサービスクラスの「namespace+ クラス名」で指定します。
今回の例では、″WSPOSEventService.POSPrinterEventService″ です。

```
      <service name=" WSPOSEventService.POSPrinterEventService ">
        <host>
          <baseAddresses>
```

❑ baseAddress にイベント取得用の URI を指定する。

```
            <add baseAddress = "http://192.168.1.102:8001/POSPrinterEvent" />
          </baseAddresses>
        </host>
```

❑ イベント取得用サービスの endpoint 情報を指定する。

```
        <endpoint address=""
                  binding="basicHttpBinding"
                  contract="UnifiedPOS.POSPrinterEvents.V1_2.POSPrinterEvent"
                  bindingNamespace="http://www.nrf-arts.org/UnifiedPOS/
                  POSPrinterEvents/" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

アプリケーション構成ファイルの詳細については、下記ウェブサイトを参照してください。
http://msdn.microsoft.com/ja-jp/library/ms733932.aspx
(2013 年 10 月 1 日時点)

4 アプリケーションでイベント取得用サービスを設定します。

アプリケーションファイル (Application.cs) 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。

〈Application.cs ファイル〉

```
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://192.168.1.102:8001/POSPrinterEvent");

////////////////////////////
// メソッドの呼び出しなどを行う //
////////////////////////////

// 終了時にはイベント取得用サービスも終了する
posPrinterEventServiceHost.Close();
```

LongPolling 方式

LongPolling 用のスレッドを用意し、Service Provider に対してイベント取得のリクエストを送信する処理を、Application.cs ファイルに記述します。

〈Application.cs ファイル〉

```
  private Thread longPollingThread;
```

❑ イベント取得用のサービスの開始

```
（第２引数には null を指定）
device.OpenSession(consumerID, null);
```

❑ ロングポーリング用にスレッドを作成して開始

```
longPollingThread = new Thread(new ThreadStart(POSPrinterEvent_LongPolling));
longPollingThread.IsBackground = true;
// メソッドの開始
longPollingThread.Start();
```

❑ Event 処理用のメソッド（メソッド名は任意）

```
private void POSPrinterEvent_LongPolling() {
  // イベント取得時の処理
  // 詳細は、POSPrinterEvent_LongPolling メソッド (p.23) を参照
}
```

POSPrinterEvent_LongPolling メソッド

スレッド用のメソッドは、以下を参考に記述します。

〈Application.cs ファイル〉

```
private void POSPrinterEvent_LongPolling(){
  while(consumerID != null){
    try{
```

❏ Service Provider に対して、イベント取得のリクエストを発行する。

```
      UnifiedPOS.POSPrinter.V1_2.POSPrinterEvent deviceEvent;
      // イベントポーリングを実行
      deviceEvent = device.PollForUPOSEvent(consumerID);

      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
```

❏ 各イベントが通知された際の処理を記述する。

（詳細は、受信したイベントごとの処理の実行方法 (p.24) を参照してください。）

```
      if (deviceEvent.ErrorEvent != null) {
        // ErrorEvent 取得時の処理
      } else if (deviceEvent.OutputCompleteEvent != null) {
        // OutputCompleteEvent 取得時の処理
      } else if (deviceEvent.StatusUpdateEvent != null) {
        // StatusUpdateEvent 取得時の処理
      } else if (deviceEvent.DirectIOEvent != null) {
        // DirectIOEvent 取得時の処理
      }

      // イベントへの応答（WS-POS Service Provider への応答）
      while (true) {
        try {
```

❏ Service Provider に対して、Response を返す。

イベント通知を受け取ったら、POSPrinterEventResponse に必要な情報をセットして、SetEventResponse() メソッドを呼び出します。

```
          device.SetEventResponse(consumerID, res);

          break;
```

❏ タイムアウトになった場合、再度イベント取得のリクエストを発行する。

```
        } catch (TimeoutException) {
          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;
          }
        }
      }
```

❏ タイムアウトになった場合、再度イベント取得のリクエストを発行する。

```
    } catch (TimeoutException) {
      continue;

    }
  }
}
```

受信したイベント ごとの処理の実行方法

イベントを受信したら、UnifiedPOS.POSPrinter.V1_2.POSPrinterEventResponse クラスに必要な情報を設定し、SetEventResponse() で Service Provider にレスポンスを返します。以下を参考にしてください。

〈Application.cs ファイル〉

```
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
```

❑ ErrorEvent 取得時の処理

```
if (deviceEvent.ErrorEvent != null) {
  UnifiedPOS.POSPrinter.V1_2.ErrorEvent ev = deviceEvent.ErrorEvent;
  res.ErrorEventResponse = new UnifiedPOS.POSPrinter.V1_2.ErrorEventResponse();
  res.ErrorEventResponse.EventID = ev.EventID;
```

❑ OutputCompleteEvent 取得時の処理

```
} else if (deviceEvent.OutputCompleteEvent != null) {
  UnifiedPOS.POSPrinter.V1_2.OutputCompleteEvent ev = deviceEvent.OutputCompleteEvent;
  res.OutputCompleteEventResponse = new
                  UnifiedPOS.POSPrinter.V1_2.OutputCompleteEventResponse();
  res.OutputCompleteEventResponse.EventID = ev.EventID;
```

❑ StatusUpdateEvent 取得時の処理

```
} else if (deviceEvent.StatusUpdateEvent != null) {
  UnifiedPOS.POSPrinter.V1_2.StatusUpdateEvent ev = deviceEvent.StatusUpdateEvent;
  res.StatusUpdateEventResponse = new UnifiedPOS.POSPrinter.V1_2.StatusUpdateEventResponse();
  res.StatusUpdateEventResponse.EventID = ev.EventID;
```

❑ DirectIOEvent 取得時の処理

```
} else if (deviceEvent.DirectIOEvent != null) {
  UnifiedPOS.POSPrinter.V1_2.DirectIOEvent ev = deviceEvent.DirectIOEvent;
  res.DirectIOEventResponse = new UnifiedPOS.POSPrinter.V1_2.DirectIOEventResponse();
  res.DirectIOEventResponse.EventID = ev.EventID;
}
```

❑ SetEventResponse() を呼び出し、Service Provider にレスポンス処理

```
device.SetEventResponse(consumerID, res);
```

KeepAlive

デバイスとアプリケーションの接続が確立されてから（OpenSession()）、一定時間（タイムアウト時間は Service Provider 側で設定）アプリケーションとデバイス間でアクションがない場合、デバイスとアプリケーション間のコネクションが切断されます。そのため、一定時間を過ぎても、デバイスとアプリケーション間の接続が切断されないよう、KeepAlive を行う仕組みが提供されています。

タイムアウト時間は Service Provider 側で設定されます。

〈Application.cs ファイル〉

```
private System.Windows.Forms.Timer timer;
```

❑ KeepAlive の処理をタイマーに設定

```
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
////（本書では、Provider から取得したタイムアウト時間の半分を KeepAlive の時間に使用します。）
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 {
```

❑ KeepAlive の呼び出し処理

```
      device.KeepAlive(consumerID);
    } catch (TimeoutException) {
```

❑ タイムアウトになった場合の処理

- Service Provider との接続がオフラインになっているため、チャネル情報、consumerID を null にして、再度 Open し直すようにする。
- LongPolling でイベントを取得している場合、LongPolling 用のスレッドも停止する。

```
      timer.Stop();

      // KeepAlive がタイムアウトになった場合、再度 Open からやりなおす。
      consumerID = null;
      device = null;
    }
  }
}
```

エスケープシーケンスの指定方法

ここでは、POSPrinter の PrintNormal メソッドで、エスケープシーケンスを使用する場合を例に説明します。

以下は、″Hello WS-POS!″ を太字にする場合の例です。他のエスケープシーケンスについても、同じ方法で指定してください。

```
// PrintNormal 処理
device.PrintNormal( consumerID,
                    UnifiedPOS.POSPrinter.V1_2.PrinterStation.Receipt,
                    "¥u001b|bCHello WS-POS!\n");
```

❑ 太字にする場合 (ESC|bC コマンド)

エスケープシーケンス： **¥u001b|bC**

Visual Basic .NET で開発

本章では、Visual Basic .NET(Windows Communication Foundation) 環境での、アプリケーション開発方法について説明しています。

Visula Basic .NET での開発

本章では、以下について説明しています。

本章のアプリケーション環境

本章では、以下の環境を想定して、説明しています。

項目	環境設定
Service Provider の IP アドレス : ポート番号	192.168.1.100:8087
イベント取得用の IP アドレス : ポート番号	192.168.1.102:8001
デバイスカテゴリー	POSPrinter
サービスクラス	UnifiedPOS.POSPrinter.V1_2.POSPrinter
Namespace	http://www.nrf-arts.org/UnifiedPOS/POSPrinter/
POSPrinter サービスのベースアドレス	http://192.168.1.100:8087/POSPrinter.svc
POSPrinter サービスのエンドポイント	POSPrinter1
アプリケーション用プログラムファイル名	Application.vb
開発環境	Microsoft Visual Studio 2010

アプリケーションプロジェクトの設定

新規プロジェクトの作成

Visual Studio で新規プロジェクトを作成する際、Visual Basic の Windows アプリケーションで作成します。

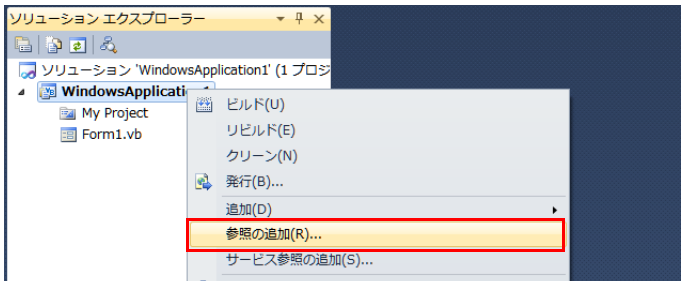
- 既存プロジェクトに追加する場合、新規プロジェクトを作成する必要はありません。
- コンソール版や Windows フォーム版でも作成できます。

WSPOSContract の追加

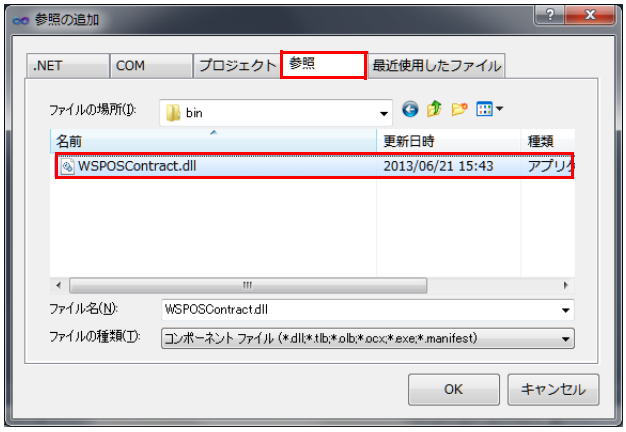
以下の手順で、WSPOSContract を追加します。

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ーに保存します。

2 Visual Studio で、ソリューションエクスプローラー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、手順 1 で保存した〝WSPOSContract.dll〟を選択し、[OK]をクリックします。

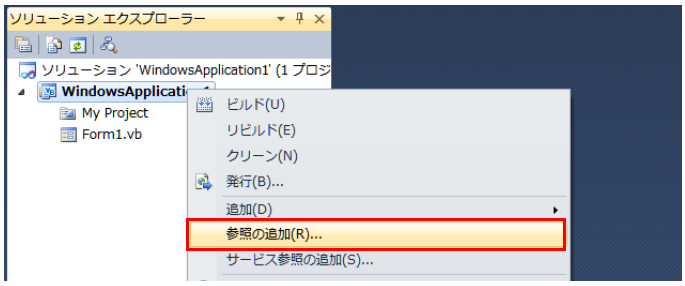


4 ソリューションエクスプローラーの参照設定に、〝WSPOSContract〟が追加されていることを確認します。

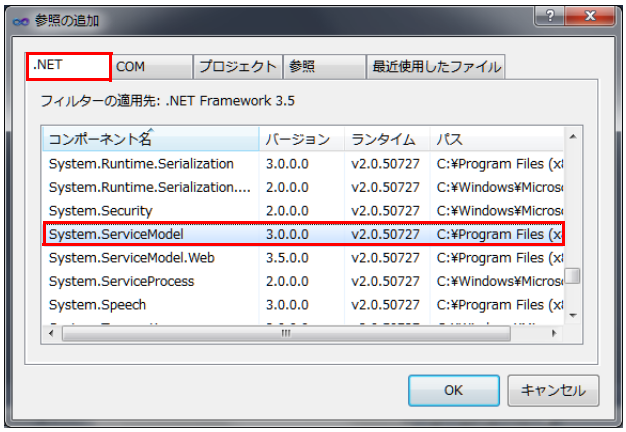
System.ServiceModel の追加

以下の手順で、System.ServiceModel を追加します。

1 Visual Studio で、ソリューションエクスプローラー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″ を選択し、[OK] をクリックします。



3 ソリューションエクスプローラーの参照設定に、″System.ServiceModel″ が追加されていることを確認します。

プロジェクトにアプリケーション構成ファイルを追加

Visual Studio のプロジェクトを生成すると、自動的にアプリケーション構成ファイル (App.config) が作成されます。以下の手順で、プロジェクトに追加します。

1 Visual Studio で、ソリューションエクスプローラーの「すべてのファイルを表示」をクリックします。

2 ソリューションエクスプローラーに〝App.config〟が追加されていることを確認します。

> アプリケーション構成ファイル(App.config) が追加されていない場合、アプリケーション構成ファイルを追加 (p.30) を参照し、アプリケーション構成ファイルを追加してください。

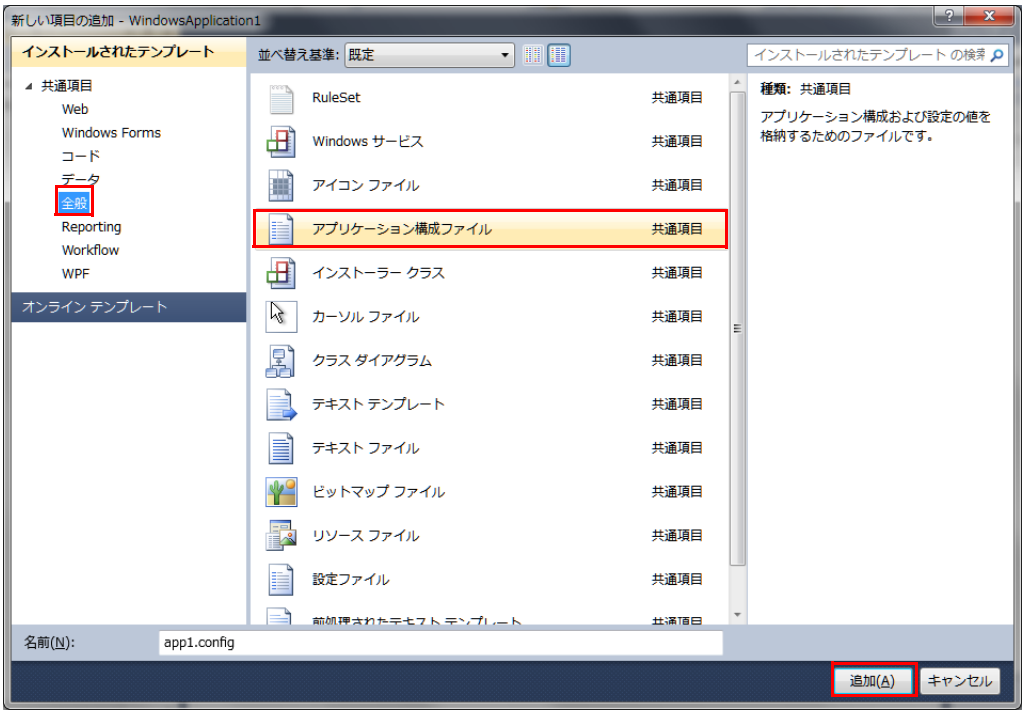
3 〝App.config〟ファイルを右クリックし、[プロジェクトに含める] を選択します。

アプリケーション構成ファイルを追加

以下の手順で、アプリケーション構成ファイルを追加します。

1 Visual Studioで、ソリューションエクスプローラー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 [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。



3 ソリューションエクスプローラーに〝App.config〟が追加されていることを確認します。

アプリケーション構成ファイルの追加

アプリケーション構成ファイル（App.config ファイル）に以下を記述してください。

〈App.config ファイル〉

```
<?xml version="1.0"?>
<configuration>
  ❏ Service Provider との接続情報を記述 (p.31)
  <system.serviceModel>
    <client>
      <endpoint address="http://192.168.1.100:8087/POSPrinter.svc/POSPrinter1"
                binding="basicHttpBinding"
                contract="UnifiedPOS.POSPrinter.V1_2.POSPrinter"
                name="POSPrinterPort1" />
    </client>
  </system.serviceModel>
  ❏ 通信ポートの最大同時オープン数を設定 (p.32)
  <system.net>
    <connectionManagement>
      <add address = "*" maxconnection = "48" />
    </connectionManagement>
  </system.net>
</configuration>
```


Service Provider との接続情報を記述

Service Provider との接続情報を、〈endpoint〉に記述します。
属性の address、contract、binding に、Service Provider の情報と同じ情報を記述します。

〈endpoint〉の属性	説明
address	使用する Service Provider の URL を指定します。 Service Provider で設定した URL と同じアドレスを指定します。
contract	使用するデバイスカテゴリーのコントラクト名を指定します。
binding	Service Provider で設定した binding を指定します。 本章では、"basicHttpBinding" を指定します。
name	接続情報を識別するための名前を指定します。 Service Consumer が Service Provider との接続をする際に、アプリケーション構成ファイルの接続情報を取得するために使用します。

アプリケーション構成ファイルの詳細については、下記ウェブサイトを参照してください。
http://msdn.microsoft.com/ja-jp/library/ms733932.aspx
（2013 年 10 月 1 日時点）

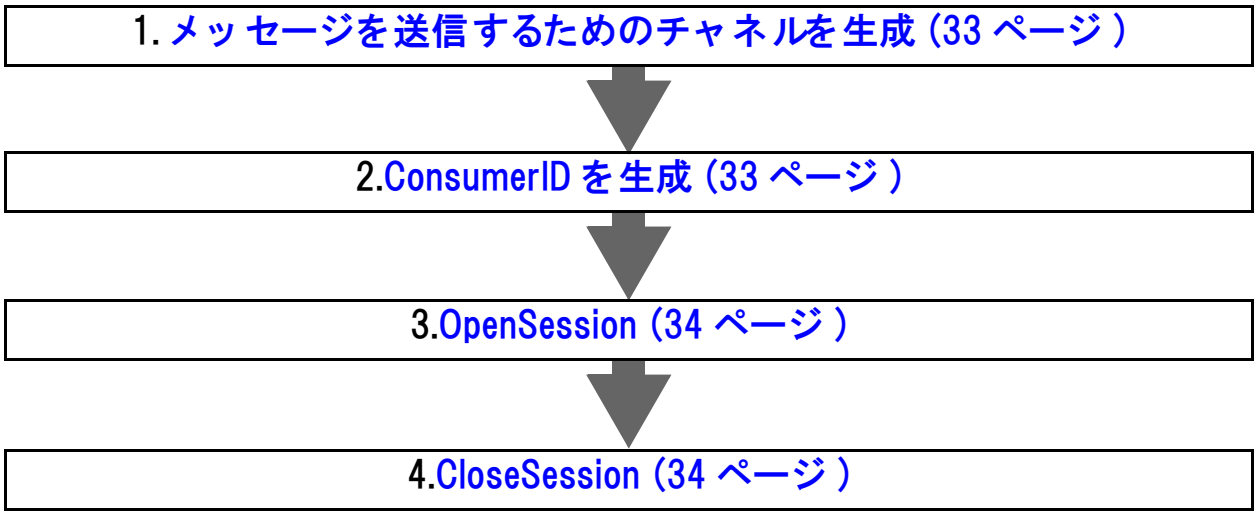
通信ポート の最大同時オープン数を設定

アプリケーションが通信ポートを開く際の、最大同時オープン数を設定します。
スレッドごとに通信ポートを開くため、スレッドの数に応じて、最大同時オープン数を指定します。
〈system.net〉タグの〈connectionManagement〉タグ内の maxconnection の値を、必要に応じて変更します。

- ここでは ″48″ で説明しています。お客様の環境に合わせて設定してください。
- サンプルアプリケーションでは、以下のスレッドを同時に使用します。通信のためのポート が、最低限 3 つ必要です。
  * メインスレッド
  * KeepAlive 用スレッド
  * Polling 用スレッド

Service Providerとの接続

Service Provider との接続に必要な処理を、アプリケーションファイル (Application.vb) に記述します。
以下の処理が必要です。



メッセージを送信するためのチャネルを生成

メッセージを送信するためのチャネルを生成します。

```
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、name を指定する
Dim factory As New ChannelFactory
                        (Of UnifiedPOS.POSPrinter.V1_2.POSPrinter)("POSPrinterPort")

' チャネルを生成する
device = factory.CreateChannel()
```

ConsumerID を生成

ConsumerID を生成します。
ConsumerID は、デバイスを識別するための識別子として使用します。

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

上記では、"GUID" を生成して ID として使用しています。ID は、開発の際に、一意に識別できるものを使用してください。

OpenSession

Service Provider とのセッションを開始します。
OpenSession を呼び出した後に、各メソッドを呼び出すことができます。

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

CloseSession

Service Provider とのセッションを終了します。
各メソッドの処理が終了したら、このメソッドを呼び出し、Service Provider とのセッションを終了してください。

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

メソッドの呼び出し方法

ここでは、printNormal メソッドを使う場合の、一連の流れを説明しています。

メッセージを送信するためのチャネルを生成 (p.33) で生成したチャネル (device) に対して、メソッドを指定することで、各メソッドを呼び出せます。

WS-POS では、各メソッドを呼び出す際に、必ず consumerID が必要になります。
サービス側は、consumerID により、デバイスとコンシューマの関連付けをします。

〈Application.vb ファイル〉

❑ セッション開始
```
device.OpenSession(consumerID, Nothing)
```
❑ デバイスの Open 処理
```
device.OpenDevice(consumerID)
```
❑ Claim 処理
```
device.Claim(consumerID, 10000)
```
❑ Enabled 処理
```
device.SetDeviceEnabled(consumerID, true)
```
❑ PrintNormal 処理
```
device.PrintNormal(consumerID, UnifiedPOS.POSPrinter.V1_2.PrinterStation.Receipt,
                   "ここに印刷するテキストを入力")
```
❑ Disabled 処理
```
device.SetDeviceEnabled(consumerID, False)
```
❑ Release 処理
```
device.Release(consumerID)
```
❑ デバイスの Close 処理
```
device.CloseDevice(consumerID)
```
❑ セッションの Close 処理
```
device.CloseSession(consumerID)
```

エラー（例外）の取得方法

各メソッドを try,catch で囲み、例外をキャッチすることでエラーを取得します。

〈Application.vb ファイル〉

```
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します。

```
    Dim ue As UnifiedPOS.POSPrinter.V1_2.UposException = ex.Detail
```

❑ ErrorCode、ErrorCodeExtended を取得します。

```
    Dim code As UnifiedPOS.POSPrinter.V1_2.ErrorCode = ue.ErrorCode
    Dim errorCodeEx As Integer = ue.ErrorCodeExtended
End Try
```

イベント の取得方法

イベントの取得方法は、以下の方法があります。

- SelfHost 方式 (p.38)

  Service Consumer(アプリケーション) 側でイベントを受け取るためのサービスクラスを用意し、このサービスクラスに対して、Service Provider 側からイベントを通知してもらう方法です。

  ここでは、以下の環境を想定して設定します。

項目	設定
イベント取得用の IP アドレス : ポート番号	192.168.1.102:8001
イベント取得用のサービスクラス	POSPrinterEventService
イベント取得用サービスのベースアドレス	http://192.168.1.102:8001/POSPrinterEvent
サービスクラスの namespace	WSPOSEventService
デバイスカテゴリー	POSPrinter
サービス用プログラムファイル名	POSPrinterEventService.vb

- LongPolling 方式 (p.40)

  Service Consumer(アプリケーション) から Service Provider に対してイベント取得のリクエストを送り、そのレスポンスとしてイベントを通知してもらう方法です。

SelfHost 方式

以下の方法で、イベントを取得するプログラムを記述します。

1 Visual Studio でイベント取得用サービスクラスを作成します。
プロジェクトを選択し、右クリックメニューから [追加]-[新しい項目] を選択し、VB のクラスを追加します。ファイル名は、"POSPrinterEventService.vb" にします。

2 追加したファイル POSPrinterEventService.vb を編集します。
デバイスカテゴリーごとに必要なメソッドが異なるので、必要に応じて追加、削除します。
以下を参考にして、編集してください。

〈POSPrinterEventService.vb ファイル〉

```
Namespace WSPOSEventService
```

❑ イベント取得用のサービスを作成するための宣言をする

```
  <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 が発生した際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
```

❑ OutputCompleteEvent が発生した際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
```

❑ StatusUpdeteEvent が発生した際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 が発生した際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
```

3 アプリケーション構成ファイル (App.config) を編集します。

以下を編集します。

〈App.config ファイル〉

```
<configuration>
  <system.serviceModel>
    <services>
```

❑ サービス名を指定する。

サービス名は、イベント取得用のサービスクラスの「namespace+ クラス名」で指定します。
今回の例では、″WSPOSEventService.POSPrinterEventService″ です。

```
      <service name=" WSPOSEventService.POSPrinterEventService ">
        <host>
          <baseAddresses>
```

❑ baseAddress にイベント取得用の URI を指定する。

```
            <add baseAddress = "http://192.168.1.102:8001/POSPrinterEvent" />
          </baseAddresses>
        </host>
```

❑ イベント取得用サービスの endpoint 情報を指定する。

```
        <endpoint address=""
                  binding="basicHttpBinding"
                  contract="UnifiedPOS.POSPrinterEvents.V1_2.POSPrinterEvent"
                  bindingNamespace="http://www.nrf-arts.org/UnifiedPOS/
                  POSPrinterEvents/" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

アプリケーション構成ファイルの詳細については、下記ウェブサイトを参照してください。
http://msdn.microsoft.com/ja-jp/library/ms733932.aspx
(2013 年 10 月 1 日時点)

4 アプリケーションでイベント取得用サービスを設定します。

アプリケーションファイル (Application.vb) 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。

〈Application.vb ファイル〉

```
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://192.168.1.102:8001/POSPrinterEvent")

''''''''''''''''''''''''''''
' メソッドの呼び出しなどを行う '
''''''''''''''''''''''''''''

' 終了時はイベント取得用サービスも終了する
posPrinterEventServiceHost.Close()
```

LongPolling 方式

LongPolling 用のスレッドを用意し、Service Provider に対してイベント取得のリクエストを送る処理を、Application.vb ファイルに記述します。

〈Application.vb ファイル〉

```
  Dim longPollingThread As Threading.Thread
```

❑ セッションオープン

（第２引数には null を指定）

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

❑ ロングポーリング用にスレッドを作成して開始

```
longPollingThread = New Threading.Thread(AddressOf POSPrinterEvent_LongPolling)
longPollingThread.IsBackground = True
longPollingThread.Start()
```

❑ Event 処理用のメソッド（メソッド名は任意）

```
Private Sub POSPrinterEvent_LongPolling()
   ' この中にイベント取得時の処理を記述。
   ' 詳細は、POSPrinterEvent_LongPolling メソッド (p.41) を参照
End Sub
```

POSPrinterEvent_LongPolling メソッド

スレッド用のメソッドは、以下を参考に記述します。

〈Application.vb ファイル〉

```
Private Sub POSPrinterEvent_LongPolling()
  While Not consumerID Is Nothing
    Try
```

❏ Service Provider に対して、イベント取得のリクエストを発行する。

```
      Dim deviceEvent As UnifiedPOS.POSPrinter.V1_2.POSPrinterEvent
      ' イベントポーリングを実行
      deviceEvent = device.PollForUPOSEvent(consumerID)

      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
```

❏ 各イベントが通知された際の処理を記述する。

（詳細は、受信したイベントごとの処理の実行方法 (p.42) を参照してください。）

```
      If Not deviceEvent.ErrorEvent Is Nothing Then
        ' ErrorEvent 取得時の処理
      ElseIf Not deviceEvent.OutputCompleteEvent Is Nothing Then
        ' OutputCompleteEvent 取得時の処理
      ElseIf Not deviceEvent.StatusUpdateEvent Is Nothing Then
        ' StatusUpdateEvent 取得時の処理
      ElseIf Not deviceEvent.DirectIOEvent Is Nothing Then
        ' DirectIOEvent 取得時の処理
      End If

      ' イベントへの応答（WS-POS Service Provider への応答）
      While (True)

        Try
```

❏ Service Provider に対して、Response を返す。

イベント通知を受け取ったら、POSPrinterEventResponse に必要な情報をセットして、SetEventResponse() メソッドを呼び出します。

```
          device.SetEventResponse(consumerID, res)

          Exit Try
```

❏ タイムアウトになった場合、再度イベント取得のリクエストを発行する。

```
        Catch ex As TimeoutException
          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
```

❏ タイムアウトになった場合、再度イベント取得のリクエストを発行する。

```
      Catch ex As TimeoutException
        Continue While
      End Try

  End While
End Sub
```

受信したイベント ごとの処理の実行方法

イベントを受信したら、UnifiedPOS.POSPrinter.V1_2.POSPrinterEventResponse クラスに必要な情報を設定し、SetEventResponse() で Service Provider にレスポンスを返します。以下を参考にしてください。

〈Application.vb ファイル〉

```
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
```

❏ ErrorEvent 取得時の処理

```
If Not deviceEvent.ErrorEvent Is Nothing Then
  Dim ev As UnifiedPOS.POSPrinter.V1_2.ErrorEvent = deviceEvent.ErrorEvent
  res.ErrorEventResponse = new UnifiedPOS.POSPrinter.V1_2.ErrorEventResponse()
  res.ErrorEventResponse.EventID = ev.EventID
```

❏ OutputCompleteEvent 取得時の処理

```
ElseIf Not deviceEvent.OutputCompleteEvent Is Nothing Then
  Dim ev As UnifiedPOS.POSPrinter.V1_2.OutputCompleteEvent = deviceEvent.OutputCompleteEvent
  res.OutputCompleteEventResponse = new
                UnifiedPOS.POSPrinter.V1_2.OutputCompleteEventResponse()
  res.OutputCompleteEventResponse.EventID = ev.EventID
```

❏ StatusUpdateEvent 取得時の処理

```
ElseIf Not deviceEvent.StatusUpdateEvent Is Nothing Then
  Dim ev As UnifiedPOS.POSPrinter.V1_2.StatusUpdateEvent = deviceEvent.StatusUpdateEvent
  res.StatusUpdateEventResponse = new UnifiedPOS.POSPrinter.V1_2.StatusUpdateEventResponse()
  res.StatusUpdateEventResponse.EventID = ev.EventID
```

❏ DirectIOEvent 取得時の処理

```
ElseIf Not deviceEvent.DirectIOEvent Is Nothing Then
  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
```

❏ SetEventResponse() を呼び出し、Service Provider にレスポンス処理

```
device.SetEventResponse(consumerID, res)
```

KeepAlive

デバイスとアプリケーションの接続が確立されてから（OpenSession()）、一定時間（ タイムアウト時間は Service Provider 側で設定 ）アプリケーションとデバイス間でアクションがない場合、デバイスとアプリケーション間のコネクションが切断されます。そのため、一定時間を過ぎても、デバイスとアプリケーション間の接続が切断されないよう、KeepAlive を行う仕組みが提供されています。

タイムアウト時間は Service Provider 側で設定されます。

〈Application.vb ファイル〉

```
Dim timer As System.Windows.Forms.Timer
timer = New Timer()
```

❑ KeepAlive の処理をタイマーに設定

```
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する
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する
'' （本書では、Provider から取得したタイムアウト時間の半分を KeepAlive の時間に使用します。）
timer.Interval = ((providerSessionTimeout / 2) * 1000)
'' タイマーを開始
timer.Start()


Private Sub timer_Tick(sender As Object, e As EventArgs)
  ' WS-POS KeepAlive
  If Not consumerID Is Nothing Then
    Try
```

❑ KeepAlive の呼び出し処理

```
      device.KeepAlive(consumerID)
    Catch ex As TimeoutException
```

❑ タイムアウトになった場合の処理

- Service Provider との接続がオフラインになっているため、チャネル情報、consumerID を null にして、再度 Open し直すようにする。
- LongPolling でイベントを取得している場合、LongPolling 用のスレッドも停止する。

```
      timer.Stop()

      ' KeepAlive がタイムアウトになった場合、再度 Open からやりなおす。
      consumerID = Nothing
      device = Nothing
    End Try
  End If
End Sub
```

エスケープシーケンスの指定方法

ここでは、POSPrinter の PrintNormal メソッドで、エスケープシーケンスを使用する場合を例に説明します。

以下は、″Hello WS-POS!″ を太字にする場合の例です。他のエスケープシーケンスについても、同じ方法で指定してください。

```
' PrintNormal 処理
device.PrintNormal( consumerID,
                    UnifiedPOS.POSPrinter.V1_2.PrinterStation.Receipt,
                    "¥u001b|bCHello WS-POS!\n")
```

❑ 太字にする場合 (ESC|bC コマンド)

エスケープシーケンス： **¥u001b|bC**

JavaScript で開発

本章では、JavaScript 環境での、アプリケーション開発方法について説明しています。

JavaScript での開発

WS-POS のサービスは、SOAP メッセージを送信して、Service Provider と Service Consumer 間の通信を行います。

本書では、以下について説明しています。

> ⚠ WS-POS サービス環境が Windows サービスの場合、JavaScript で開発したアプリケーションは使用できません。WS-POS 参照実装にクロスドメインアクセスの制限があるため、「WS-POS 環境セットアップガイド」では、回避する一例として、Internet Information Services 環境の構築方法について説明しています。本章では、その環境に合わせて、JavaScript を使った開発方法を説明しています。

本章のアプリケーション環境

本章では、以下の環境を想定して、説明しています。

項目	環境設定
Service Provider の IP アドレス : ポート番号	192.168.1.100:8087
デバイスカテゴリー	POSPrinter
Namespace	http://www.nrf-arts.org/UnifiedPOS/POSPrinter/
POSPrinter サービスのベースアドレス	http://192.168.1.100:8087/POSPrinter.svc
POSPrinter サービスのエンドポイント	POSPrinter1
JavaScript ファイル名	POSPrinterControl.js

SOAP メッセージの送信方法

SOAP メッセージの送信には、以下の処理を行います。参考にしてください。

〈POSPrinterControl.js ファイル〉

```
var ID = "0123456789";
```

❑ XMLHttpRequest オブジェクト生成

```
// XMLHttpRequest オブジェクト生成メソッドの呼び出し
var xmlhttp = createHttpRequest();
```

❑ SOAP メッセージの作成方法 (p.49)

```
// 送信する SOAP メッセージの作成
var message = createWSPOSMessage(ID);
```

❑ Service Provider にリクエストを送信する準備 (p.47)

```
// 要求メッセージの送信準備
xmlhttp.open("POST", "http://192.168.1.100:8087/POSPrinter.svc/POSPrinter1", false);
xmlhttp.setRequestHeader("Content-Type", "text/xml; charset=UTF-8");
xmlhttp.setRequestHeader("SOAPAction", "\"http://www.nrf-arts.org/UnifiedPOS/
                                        POSPrinter/PrintNormal\"");
xmlhttp.onreadystatechange = function(){callback(xmlhttp)};
```

❑ 要求メッセージの送信

```
xmlhttp.send(message);
```

```
// 各ブラウザ用の XMLHttpRequest オブジェクトを生成
function createHttpRequest(){
    // Internet Explorer 用
    if(window.ActiveXObject){
        try {
            // MSXML2 以降用
            return new ActiveXObject("Msxml2.XMLHTTP");
        } catch (e) {
            try {
                // 旧 MSXML 用
                return new ActiveXObject("Microsoft.XMLHTTP");
            } catch (e2) {
                return null;
            }
         }
    } else if(window.XMLHttpRequest){
        // Internet Explorer 以外の XMLHttpRequest オブジェクト実装ブラウザ用
        return new XMLHttpRequest();
    } else {
        return null;
    }
}
```

上記は、PrintNormal メソッドを呼び出す処理を行っています。

Service Provider にリクエスト を送信する準備

以下の処理を行います。

1 open メソッドで、以下を指定します。

```
xmlhttp.open("POST", "http://192.168.1.100:8087/POSPrinter.svc/POSPrinter1", false);
```

パラメーター	値
リクエストタイプ	SOAP では、POST メソッドを利用してメッセージを送るため、リクエストタイプは "POST" を指定
サービスの URL	利用する Service Provider の URL を指定
同期通信	false を指定 (環境に合わせて指定してください。)

2 setRequestHeader() メソッドで HTTP リクエスト時のヘッダーを設定します。

```
xmlhttp.setRequestHeader("Content-Type", "text/xml; charset=UTF-8");
xmlhttp.setRequestHeader("SOAPAction", "\"http://www.nrf-arts.org/UnifiedPOS/
                          POSPrinter/PrintNormal\"");
```

3 onreadystatechange イベントが発生した場合の、コールバック関数を指定します。

```
xmlhttp.onreadystatechange = function(){callback(xmlhttp)};
```

〈指定するコールバック関数〉

```
function callback (xmlhttp){
  // リクエスト処理常態が完了の場合
  if (xmlhttp.readyState == 4) {
    if (xmlhttp.status == 200) {
      // http ステータスが正常 (200) な場合の処理
    } else {
      // http ステータスが異常あり (200 以外 ) の場合の処理
    }
  }
}
```

メソッドの呼び出し方法

ここでは、printNormal メソッドを使う場合の、各メソッドを呼び出す順番を説明しています。
メソッドの呼び出し方は、SOAP メッセージの送信方法 (p.46) を参考にしてください。

❏ セッション開始
`OpenSession`
❏ デバイスの Open 処理
`OpenDevice`
❏ Claim 処理
`Claim`
❏ Enabled 処理
`SetDeviceEnabled`
❏ PrintNormal 処理
`PrintNormal`
❏ Disabled 処理
`SetDeviceEnabled`
❏ Release 処理
`Release`
❏ デバイスの Close 処理
`CloseDevice`
❏ セッションの Close 処理
`CloseSession`

SOAP メッセージの作成方法

ここでは、printNormal メソッドのリクエストを送信する場合のメッセージの例を記述します。
メソッドによって記述する内容が異なります。以下を参考にしてコーディングしてください。

送信する SOAP メッセージの内容

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>

    ❑ printNormal メソッド呼び出し
    <PrintNormal xmlns="http://www.nrf-arts.org/UnifiedPOS/POSPrinter/">

      ❑ printNormal メソッドのパラメーターの指定
        メソッドのパラメーターの詳細は、パラメーターの定義 (p.49) 参照してください。
        <ConsumerID>1234567890</ConsumerID>
        <Station>Receipt</Station>
        <Data> ここに印刷する文字を記述します。</Data>
    </PrintNormal>

  </s:Body>
</s:Envelope>
```

ConsumerID は、開発の際に、一意に識別できるものを使用してください。

パラメーターの定義

メソッドごとのパラメーターは、WS-POS の仕様書に同梱されている〝POSPrinterV1.13.2.xsd〟ファイルに定義されています。
メソッド名の <element> タグにある、<sequence> タグ内の各 <element> タグがパラメーターになります。
以下は、printNormal メソッドのパラメーター定義です。

❑ パラメーターを定義するメソッド

```
<xs:element name="PrintNormal">
  <xs:complexType>
    <xs:sequence>

      ❑ パラメーターの定義
        <xs:element minOccurs="0" name="ConsumerID" nillable="true" type="xs:string" />
        <xs:element minOccurs="0" name="Station" type="tns:PrinterStation" />
        <xs:element minOccurs="0" name="Data" nillable="true" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

エラー（例外）の取得方法

送信した SOAP メッセージの受信が失敗した場合、例外が発生します。

例外が発生した場合、コールバック関数内に以下の処理を記述することで、UposException の例外内容を取得できます。

〈POSPrinterControl.js ファイル〉

```
// コールバック関数の処理に追加する。
var responseXML = xmlhttp.responseXML;

var fault = responseXML.getElementsByTagName("s:Fault");
if( fault.length > 0 ){
```

❑ エラーメッセージは、faultstring タグの中に記述されます。

```
  var strMsg =
        fault[0].getElementsByTagName("faultstring").item(0).firstChild.nodeValue;

  var uposException = fault[0].getElementsByTagName("UposException");
  if( uposException.length > 0 ){
```

❑ UposException の例外は、UposException タグの中に記述されます。

```
    var errorCode =
        fault[0].getElementsByTagName("ErrorCode").item(0).firstChild.nodeValue;
    var errorCodeExtended =
        fault[0].getElementsByTagName("ErrorCodeExtended").item(0).firstChild.nodeValue;
  }
}
```

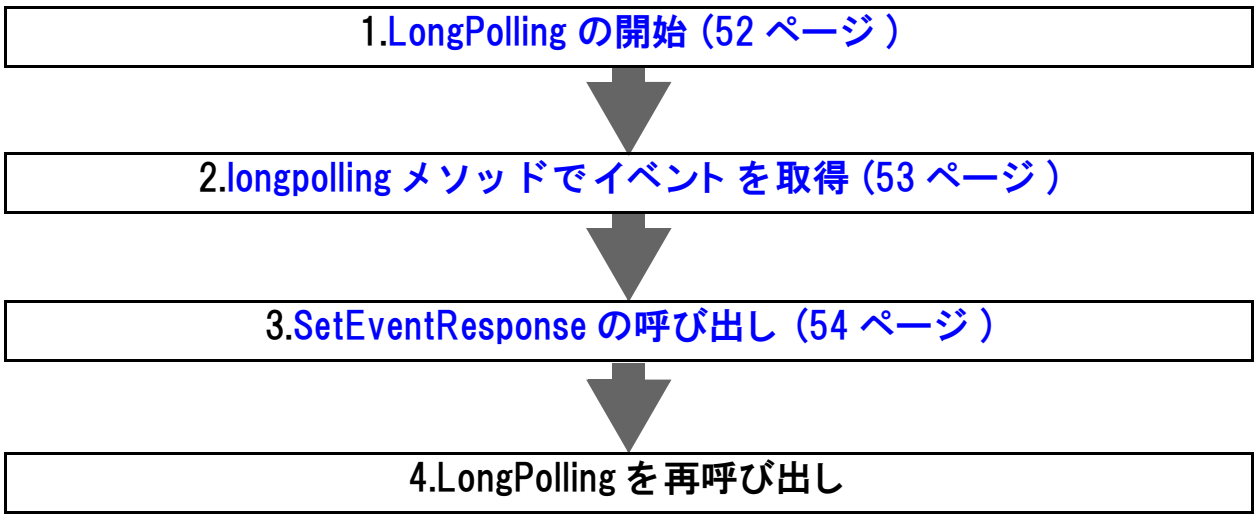
UposException の仕様の詳細は、"POSPrinterV1.13.2.xsd" ファイルを参照してください。

イベント の取得方法

イベントは、LongPolling 方式で取得します。

Service Consumer(アプリケーション) から Service Provider に対してイベント取得のリクエストを送り、そのレスポンスとしてイベントを通知してもらう方法です。

以下の処理が必要です。



LongPolling 用の SOAP メッセージの送信方法は、SOAP メッセージの送信方法 (p.46) を参照してください。

LongPolling の開始

イベントを取得する LongPolling のコールバック指定して SOAP メッセージを送信します。
以下を参考にしてコーディングしてください。

<POSPrinterControl.js ファイル>

```
var xmlhttp_longpolling;
```

❏ XMLHttpRequest オブジェクトの生成

```
xmlhttp_longpolling = createHttpRequest();
```

❏ 送信する SOAP メッセージの作成

詳細は、送信する SOAP メッセージ (p.52) を参照してください。

```
var message = createWSPOSMessage_longpolling(ID);
```

❏ SOAP メッセージの送信準備

```
// open メソッドで、非同期通信を指定
xmlhttp_longpolling.open("POST", "http://192.168.1.100:8087/POSPrinter.svc/POSPrinter1",
true);
// setRequestHeader メソッドで HTTP リクエスト時のヘッダーを設定
xmlhttp_longpolling.setRequestHeader("Content-Type", "text/xml; charset=UTF-8");
xmlhttp_longpolling.setRequestHeader("SOAPAction","\"http://www.nrf-arts.org/UnifiedPOS/
POSPrinter/PollForUPOSEvent\"");
// イベント発生時に呼ばれる LongPolling のコールバック関数を指定 (p.53)
xmlhttp_longpolling.onreadystatechange = function(){longpolling(xmlhttp_longpolling)};
```

❏ SOAP メッセージの送信

```
xmlhttp_longpolling.send(message);
```

送信する SOAP メッセージ

送信する SOAP メッセージの内容は、以下のとおりです。

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>
    <PollForUPOSEvent xmlns="http://www.nrf-arts.org/UnifiedPOS/POSPrinter/">
      <ConsumerID>1234567890</ConsumerID>
    </PollForUPOSEvent>
  </s:Body>
</s:Envelope>
```

longpolling メソッドで イベント を取得

イベント発生時に呼ばれる、コールバック関数 (longpolling メソッド) を記述します。
以下を参考にしてコーディングしてください。

<POSPrinterControl.js ファイル>

```
function longpolling(xmlhttp) {

  if (xmlhttp.readyState == 4) {
    if (xmlhttp.status == 200) {
```

❏ **http ステータスが正常時の処理 (http ステータス :200)**

イベントを解析し、Service Provider に SetEventResponse を返します。
SetEventResponse の詳細は、SetEventResponse の呼び出し (p.54) を参照してください。

```
      longPollingResponse(xmlhttp);
    } else {
```

❏ **http ステータスが異常時の処理 (http ステータス :200 以外)**

```
    }
  }
}
```

SetEventResponse の呼び出し

イベントを解析し、Service Provider に対して SetEventResponse を返します。

- ここでは、デバイスカテゴリーの POSPrinter を例にしています。使用するデバイスカテゴリーごとに、対応するイベント が異なります。必要に応じて、イベント をコーディングしてください。
- SetEventResponse は、イベント ごとに必要なパラメーターが変わります。定義を確認してください。

<POSPrinterControl.js ファイル>

```
function longPollingResponse(xmlhttp){

  var responseXML = xmlhttp.responseXML;

  // ErrorEvent
  ❑ XML データから ErrorEvent タグの情報を取得
  var errorEvent = responseXML.getElementsByTagName("ErrorEvent");
  if( errorEvent.length > 0) {
    ❑ ErrorEvent 取得時の処理（情報が存在する場合）
      詳細は、イベント取得時の処理 (p.55) を参照してください。
  }

  // OutputCompleteEvent
  //   XML データから OutputCompleteEvent タグの情報を取得
  var outputCompEvent = responseXML.getElementsByTagName("OutputCompleteEvent");
  if( outputCompEvent.length > 0) {
    // OutputCompleteEvent 取得時の処理（情報が存在する場合）
  }

  // StatusUpdateEvent
  //   XML データから StatusUpdateEvent タグの情報を取得
  var statusUpdateEvent = responseXML.getElementsByTagName("StatusUpdateEvent");
  if( statusUpdateEvent.length > 0) {
    // StatusUpdateEvent 取得時の処理（情報が存在する場合）
  }

  // DirectIOEvent
  //   XML データから DirectIOEvent タグの情報を取得
  var directIO = responseXML.getElementsByTagName("DirectIOEvent");
  if( directIO.length > 0) {
    // DirectIOEvent 取得時の処理（情報が存在する場合）
  }
}
```

イベント 取得時の処理

取得したイベントのデータを、必要に応じて解析し、処理の完了を errorEvent_setEventResponse メソッドで Service Provider に送信します。正しく送信されると、Service Provider から SetEventResponse がレスポンスされます。レスポンスを確認したら、再度、LongPolling を呼び出します。

ここでは、ErrorEvent を例にしています。他のイベントも以下の例を参考に解析し、SetEventResponse を呼び出してください。

〈POSPrinterControl.js ファイル〉

```
var num = errorEvent[0].attributes.length;
if(num > 0) {
  if(errorEvent[0].attributes.item(0).text == "true") {

  }
}else{
  if(errorEvent[0].childNodes.length > 0) {
```

❑ イベントのデータを取得・解析

```
    var oEventID = errorEvent[0].getElementsByTagName("EventID").item(0);
    var evID = oEventID.firstChild.nodeValue;
    var oErrorCode = errorEvent[0].getElementsByTagName("ErrorCode").item(0);
    var evErrorCode = oErrorCode.firstChild.nodeValue;
    var oErrorCodeExtended =
              errorEvent[0].getElementsByTagName("ErrorCodeExtended").item(0);
    var evErrorCodeExtended = oErrorCodeExtended.firstChild.nodeValue;
    var oErrorLocus = errorEvent[0].getElementsByTagName("ErrorLocus").item(0);
    var evErrorLocus = oErrorLocus.firstChild.nodeValue;
    var oErrorResponse = errorEvent[0].getElementsByTagName("ErrorResponse").item(0);
    var evErrorResponse = oErrorResponse.firstChild.nodeValue;
```

❑ errorEvent_setEventResponse メソッドの呼び出し

```
    errorEvent_setEventResponse(evID, "Clear");
    return;
  }
}
```

KeepAlive

デバイスとアプリケーションの接続が確立されてから（OpenSession()）、一定時間アプリケーションとデバイス間でアクションがない場合、デバイスとアプリケーション間のコネクションが切断されます。

そのため、一定時間を過ぎても、デバイスとアプリケーション間の接続が切断されないよう、KeepAlive の仕組みが提供されています。

- タイムアウト時間は Service Provider 側で設定されます。
- KeepAlive 関数の呼び出し方は、SOAP メッセージの送信方法 (p.46) を参照してください。

以下は、JavaScript のタイマー機能を使い、定期的に KeepAlive 関数を呼び出す方法です。

<POSPrinterControl.js ファイル>

- **KeepAlive 開始（タイマースタート）**
  定期的に KeepAlive 関数を呼び出します。

```
var timerID = setInterval("KeepAlive(GUID)", 60 * 1000);

function KeepAlive (GUID) {
```

- **KeeAlive メソッドを呼び出すためのメッセージを作成、送信**

```
// Http リクエストメッセージ作成
var xmlhttp = createHttpRequest();
// KeepAlive メソッドメッセージ作成
var keepAliveMessage = createWSPOSMessage_keepAlive(GUID);

// 要求メッセージ送信準備と送信
xmlhttp.open("POST", "http://192.168.1.100:8087/POSPrinter.svc/POSPrinter1", false);
xmlhttp.setRequestHeader("Content-Type", "text/xml; charset=UTF-8");
xmlhttp.setRequestHeader("SOAPAction"
        , "\"http://www.nrf-arts.org/UnifiedPOS/POSPrinter/KeepAlive\"");
xmlhttp.onreadystatechange = function(){ehLoaded(xmlhttp)};
xmlhttp.send(keepAliveMessage);
}
```

- **KeepAlive を止める（タイマーを止める）**

```
clearInterval(timerID);
```

エスケープシーケンスの指定方法

ここでは、POSPrinter の PrintNormal メソッドで、エスケープシーケンスを使用する場合を例に説明します。

以下は、″Hello WS-POS!″ を太字にし、改行する場合の例です。他のエスケープシーケンスについても、同じ方法で指定してください。

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>
    <PrintNormal xmlns="http://www.nrf-arts.org/UnifiedPOS/POSPrinter/">
      <ConsumerID>1234567890</ConsumerID>
      <Station>Receipt</Station>
      <Data>&#x1b;|bCHello WS-POS!&#x0a;</Data>
    </PrintNormal>
  </s:Body>
</s:Envelope>
```

❏ 太字にする場合 (ESC|bC コマンド)

エスケープシーケンス：　**&#x1b;|bC**

❏ 改行を指定する場合 (¥n)

エスケープシーケンス：　**
**