EPSON
EXCEED YOUR VISION

WS-POS
Application Development Guide

Overview

Describes environments for developing WS-POS compatible applications.

Visual C#

Describes development information in a Visual C# environment.

Visual Basic .NET

Describes development information in a Visual Basic .NET environment.

JavaScript

Describes development information in a JavaScript environment.

M00067200
Rev.A

Cautions

- No part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Seiko Epson Corporation.
- The contents of this document are subject to change without notice. Please contact us for the latest information.
- While every precaution has taken in the preparation of this document, Seiko Epson Corporation assumes no responsibility for errors or omissions.
- Neither is any liability assumed for damages resulting from the use of the information contained herein.
- Neither Seiko Epson Corporation nor its affiliates shall be liable to the purchaser of this product or third parties for damages, losses, costs, or expenses incurred by the purchaser or third parties as a result of: accident, misuse, or abuse of this product or unauthorized modifications, repairs, or alterations to this product, or (excluding the U.S.) failure to strictly comply with Seiko Epson Corporation's operating and maintenance instructions.
- Seiko Epson Corporation shall not be liable against any damages or problems arising from the use of any options or any consumable products other than those designated as Original EPSON Products or EPSON Approved Products by Seiko Epson Corporation.

Trademarks

EPSON® is registered trademarks of Seiko Epson Corporation in the U.S. and other countries.

Microsoft®, Windows®, Visual Studio®, Visual Basic® and Visual C#® are either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries.

JavaScript® is either registered trademarks or trademarks of Oracle Corporation in the United States and other countries.

For Safety

Key to Symbols

The symbols in this manual are identified by their level of importance, as defined below. Read the following carefully before handling the product.

	Provides information that must be observed to avoid damage to your equipment or a malfunction.
	Provides important information and useful tips.

Restriction of Use

When this product is used for applications requiring high reliability/safety such as transportation devices related to aviation, rail, marine, automotive etc.; disaster prevention devices; various safety devices etc; or functional/precision devices etc, you should use this product only after giving consideration to including fail-safes and redundancies into your design to maintain safety and total system reliability. Because this product was not intended for use in applications requiring extremely high reliability/safety such as aerospace equipment, main communication equipment, nuclear power control equipment, or medical equipment related to direct medical care etc, please make your own judgment on this product's suitability after a full evaluation.

About this Manual

Aim of the Manual

The purpose of this document is to provide developers with information required for developing and designing applications using WS-POS (Web Services for Point of Service).

Manual Content

The manual is made up of the following sections:

Contents

Overview

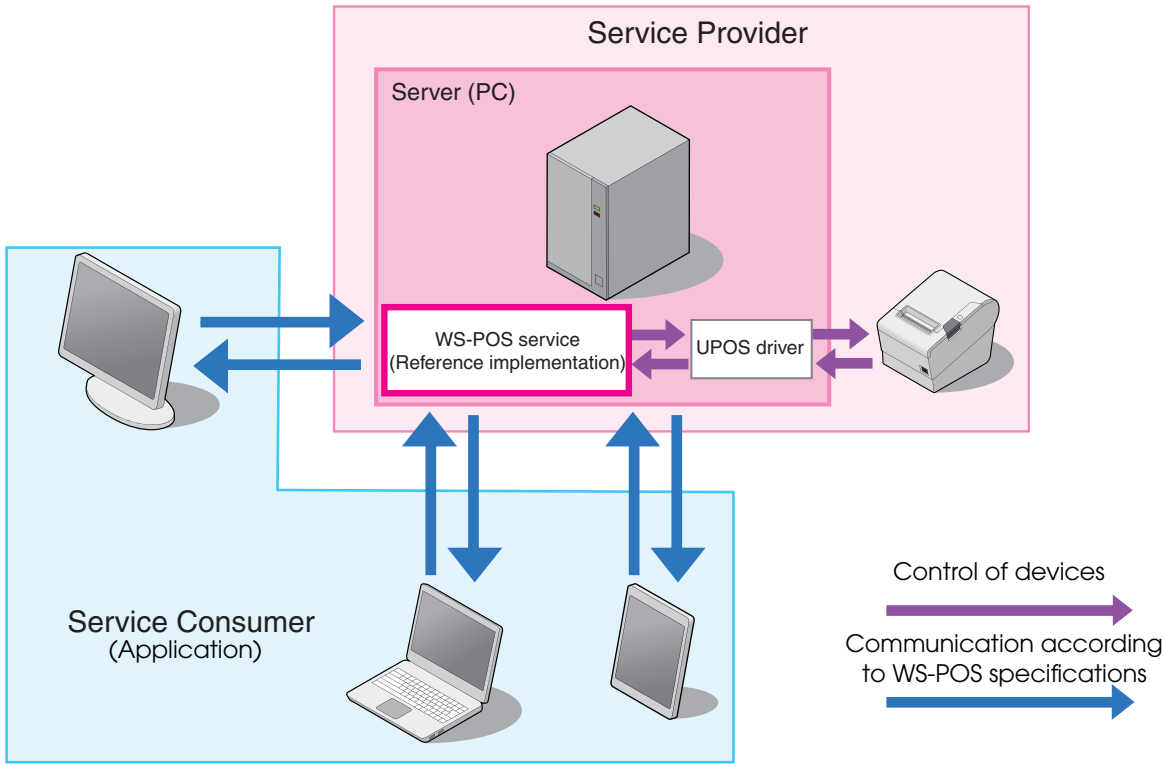
This chapter describes environments for developing WS-POS compatible applications.

Web Services for Point of Service

Web Services for Point of Service (WS-POS) control devices using a Web-based POS system.

For the WS-POS specifications and a reference implementation, refer to the following website:
http://www.nrf-arts.org/content/unifiedpos

WS-POS configuration



- Service Provider refers to the side providing WS-POS services.
- Service Consumer refers to the side using WS-POS services.
  This document sets forth information on developing applications compatible with WS-POS used by the Service Consumer.

Development Environment

Development language

This document describes methods for developing WS-POS compatible applications in the following languages:

- ❏ Microsoft Visual Basic .NET
- ❏ Microsoft Visual C#
- ❏ JavaScript

The language(s) that can be used depends on the environment where the WS-POS service is provided, as shown below.

WS-POS service environment	Development language that can be used
Windows Services	• Microsoft Visual Basic .NET • Microsoft Visual C#
Internet Information Services	• JavaScript

For the operating environments for WS-POS service, refer to the WS-POS Environment Setup Guide.

WS-POS Service (Reference Implementation)

- ❏ WS-POS reference implementation Version 1.0

Obtain from the following: (As of Oct. 1st, 2013)
http://www.nrf-arts.org/content/unifiedpos

Available Materials

Manual

- ❏ WS-POS Application Development Guide (this document)
- ❏ WS-POS Environment Setup Guide

Driver

- ❏ EPSON OPOS ADK for .NET

Download

Download the available materials from the following URL.

For customers in North America, go to the following web site:

http://www.epsonexpert.com/

For customers in other countries, go to the following web site:

https://download.epson-biz.com/?service=pos

Visual C#

This chapter describes how to develop applications in a Visual C# (Windows Communication Foundation) environment.

Application development using Visual C#

This chapter describes the following:

Application Environment in This Chapter

In this chapter, descriptions are made by assuming the following environment:

Item	Environment Setup
IP address : port number of Service Provider	192.168.1.100:8087
IP address : port number for event acquisition	192.168.1.102:8001
Device category	POSPrinter
Service class	UnifiedPOS.POSPrinter.V1_2.POSPrinter
Namespace	http://www.nrf-arts.org/UnifiedPOS/POSPrinter/
Base address of POSPrinter service	http://192.168.1.100:8087/POSPrinter.svc
Endpoint of POSPrinter service	POSPrinter1
Program file name for application	Application.cs
Development Environment	Microsoft Visual Studio 2010

Application Project Setup

Creating a new project

Select a C# Windows application when creating a new project with Visual Studio.

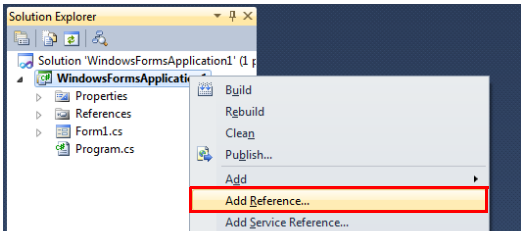
- If you add an application to an existing project, you don't have to create a new project.
- You can also create a new project with a console or Windows form application.

Adding WSPOSContract

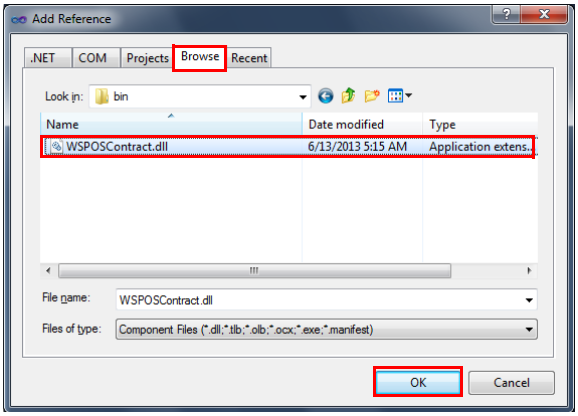
Add WSPOSContract as follows:

1 Save "WSPOSContract.dll", which is included in the reference implementation, into the project folder.

2 In Visual Studio, run Solution Explorer and right-click the project. Then click (Add Reference).



3 Open the (Browse) tab, select "WSPOSContract.dll" which you saved in step 1, and click (OK).

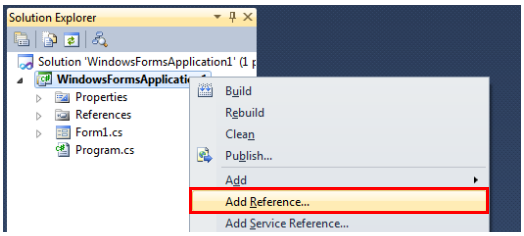


4 Confirm that "WSPOSContract" has been added to the References on Solution Explorer.

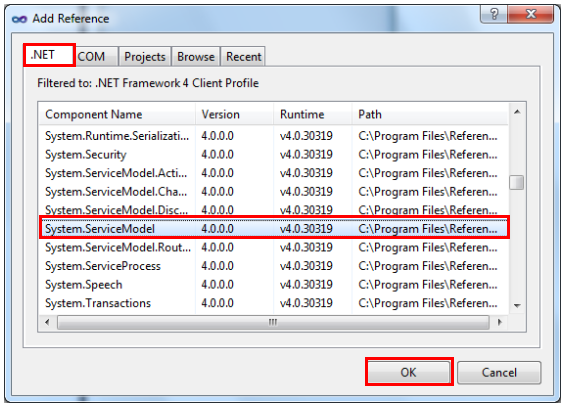
Adding System.ServiceModel

Add System.ServiceModel as follows:

1 In Visual Studio, run Solution Explorer and right-click the project. Then click (Add Reference).



2 Open the (NET) tab, select "System.ServiceModel" which you saved in step 1, and click (OK).



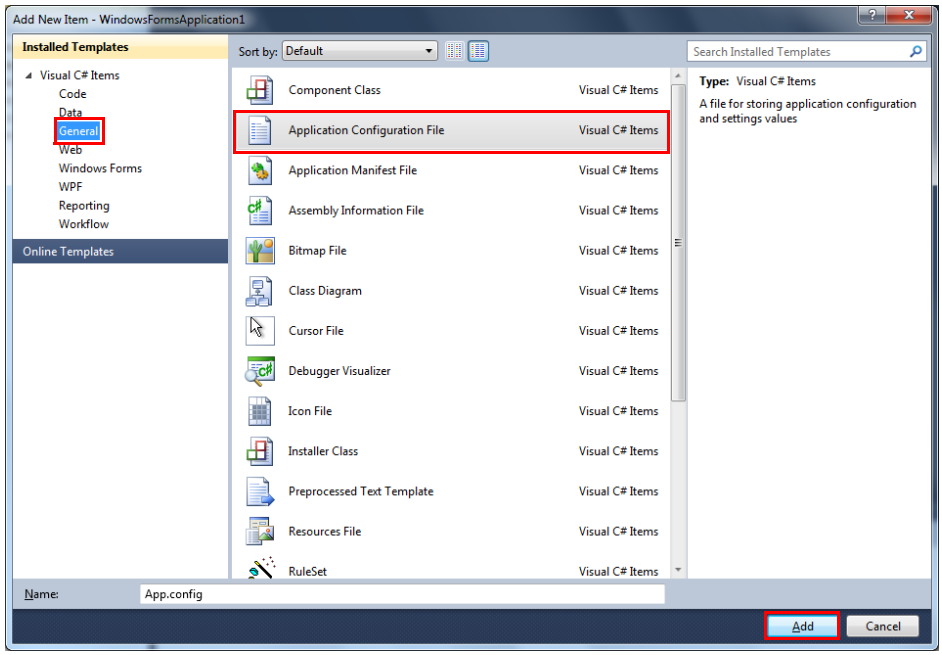
3 Confirm that "System.ServiceModel" has been added to the References on Solution Explorer.

Addition of the application configuration file

Add the application configuration file as follows:

1 In Visual Studio, select a project on Solution Explorer and select (Project)-(Add New Item) from the menu bar.

2 Select (General) from the (VisualC# Items) menu, then select (Application Configuration File) and click (Add).



3 Confirm that "App.config" has been added to Solution Explorer.

Addition of the application configuration file

Write the statements below in the application configuration file (App.config file).

<App.config file>

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

❏ Writing information about connection with Service Provider (p.17).

```
<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>
```

❏ Setting the maximum number of communication ports simultaneously opened (p.18).

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

```
</configuration>
```

For details on the application configuration file, refer to the following Website:
http://msdn.microsoft.com/en-us/library/ms733932.aspx
(As of Oct. 1st, 2013)

Writing information about connection with Service Provider

Write information about connection with Service Provider in <endpoint>.
For the attributes "address, contract, and binding", write the same pieces of information as those of Service Provider.

<endpoint> attribute	Description
address	Specify the URL of the Service Provider to be used. Specify the IP address correspoding to the URL set by the Service Provider.
binding	Specify the binding set by the Service Provider. In this chapter, specify "basicHttpBinding".
contract	Specify the contract name for the device category to be used.
name	Specify the name to identify connection information. This attribute is used for acquisition of connection information in the application configuration file when Service Consumer connects to Service Provider.

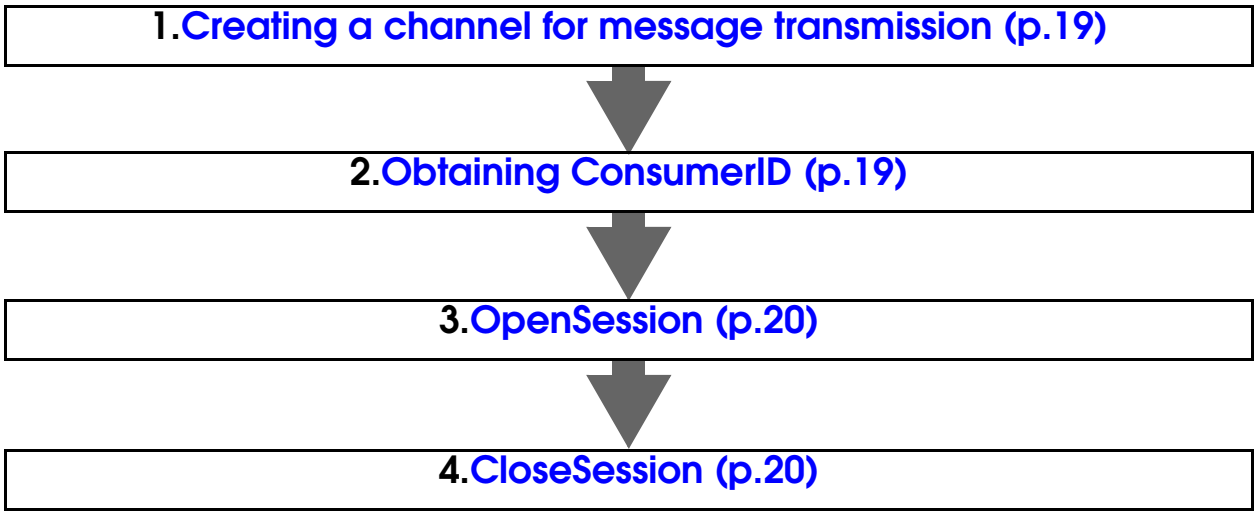
Setting the maximum number of communication ports simultaneously opened

Set the maximum number of communication ports that can be simultaneously opened by an application. Since communication ports are opened on a thread basis, specify the maximum number of communication ports simultaneously opened, based on the number of threads.
Change as needed the value for maxconnection in the <connectionManagement> tag of the <system.net> tag.

- In this section, the maximum value is set as "48" for explanation. Set the value that suits the customer's environment.
- The threads below are used simultaneously in the sample application. Therefore, at least three ports for communication are necessary.
  * Main thread
  * Thread for KeepAlive
  * Thread for Polling

Connection to Service Provider

Write a process necessary for connection to Service Provider in the application file (Application.cs).
The following process is required:



Creating a channel for message transmission

Create a channel for message transmission.

```
private UnifiedPOS.POSPrinter.V1_2.POSPrinter device = null;

// Specify the contract and name for the endpoint set by the application configuration
    file.
ChannelFactory< UnifiedPOS.POSPrinter.V1_2.POSPrinter > factory =
   new ChannelFactory< UnifiedPOS.POSPrinter.V1_2.POSPrinter >("POSPrinterPort1");

// Create a channel.
device = factory.CreateChannel();
```

Obtaining ConsumerID

Obtain ConsumerID.
ConsumerID is used as an identifier for device identification.

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

In the above example, "GUID" is created and used as ID. Use ID that can be uniquely identified during application development.

OpenSession

Start a session with Service Provider.
After calling OpenSession, you can call each method.

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

CloseSession

End the session with Service Provider.
When each method has been completely processed, call this method to terminate the session with Service Provider.

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

Method Call Procedure

This section describes a series of processes when using the printNormal method.
You can call each method by specifying it for the channel created in Creating a channel for message transmission (p.19).

> ⚠ For WS-POS, when calling each method, consumerID is always required.
> On the device side, the device and Service Consumer are associated with one another based on consumerID.

<Application.cs file>

❑ **Session starts.**

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

❑ **Device Open process.**

```
device.OpenDevice(consumerID);
```

❑ **Claim process.**

```
device.Claim(consumerID, 10000);
```

❑ **Enabled process.**

```
device.SetDeviceEnabled(consumerID, true);
```

❑ **PrintNormal process.**

```
device.PrintNormal(consumerID, UnifiedPOS.POSPrinter.V1_2.PrinterStation.Receipt,
                   "Enter here the text to be printed.");
```

❑ **Disabled process.**

```
device.SetDeviceEnabled(consumerID, false);
```

❑ **Release process.**

```
device.Release(consumerID);
```

❑ **Device Close process.**

```
device.CloseDevice(consumerID);
```

❑ **Session Close process.**

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

Error (Exception) Acquisition Procedure

Enclose a method with try and catch and catch an exception to obtain an error.

<Application.cs file>

```
try {

// Device open.
  device.OpenDevice(consumerID);
```

❑ Obtain UposException-type detailed information from the caught exception.

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

❑ Obtain ErrorCode and ErrorCodeExtended.

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

Event Acquisition Method

The following event acquisition methods are available:

- SelfHost method (p.24)

  In this method, a service class for receiving an event is provided on the ServiceConsumer side (application), and event notification is made from the Service Provider side to this service class.

  In this section, settings are made by assuming the following environment:

Item	Settings
IP address : port number for event acquisition	192.168.1.102:8001
Service class for event acquisition	POSPrinterEventService
Base address of service for event acquisition	http://192.168.1.102:8001/POSPrinterEvent
Namespace of service class	WSPOSEventService
Device category	POSPrinter
File name of program for service	POSPrinterEventService.cs

- LongPolling method (p.26)

  In this method, an event acquisition request is sent from ServiceConsumer (application) to Service Provider, and event notification is made as its response.

SelfHost method

Create a program for event acquisition as follows:

1 Create a service class for event acquisition using Visual Studio.

Select a project, right click on it and select (Project)-(Add New Item) to add a C# class. Specify the file name as "POSPrinterEventService.cs".

2 Edit the added file "POSPrinterEventService.cs".

Add or delete a method as needed because the required method varies depending on the device category.

Edit by referring to the following:

<POSPrinterEventService.cs file>

```
namespace WSPOSEventService {
```

❑ **Declaration for creating a service for event acquisition.**

```
  [ServiceBehavior (Namespace = "http://www.nrf-arts.org/UnifiedPOS/
    POSPrinterEvents/",InstanceContextMode = InstanceContextMode.Single)]

  // Take over the event class of the category to which an event you want to obtain
       belongs.
  public class POSPrinterEventService: UnifiedPOS.POSPrinter.V1_2.POSPrinterEvent
  {
```

❑ **Method to be called when ErrorEvent occurs.**

```
    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;
      }
```

❑ **Method to be called when OutputCompleteEvent occurs.**

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

❑ **Method to be called when StatusUpdeteEvent occurs.**

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

❑ **Method to be called when DirectIOEvent occurs.**

```
    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 Edit the application configuration file (App.config).

Edit the following:

<App.config file>

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

❑ **Specify the service name.**

Specify the service name as the "namespace+ class name" of the service class for event acquisition. "WSPOSEventService.POSPrinterEventService" is specified in this example.

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

❑ **Specify the URL for event acquisition for baseAddress.**

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

❑ **Specify the endpoint information of the service for event acquisition.**

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

For details on the application configuration file, refer to the following Website:
http://msdn.microsoft.com/en-us/library/ms733932.aspx
(As of Oct. 1st, 2013)

4 Set a service for event acquisition in the application.

Write a process for creating a service host for event acquisition in the application file (Application.cs).

<Application.cs file>

```
private ServiceHost posPrinterEventServiceHost;

posPrinterEventServiceHost =
 new ServiceHost(typeof(WSPOSEventService.POSPrinterEventService));

// Start a service for event acquisition.
posPrinterEventServiceHost.Open();

// Pass Endpoint using an OpenSession argument.
device.OpenSession(consumerID, "http://192.168.1.102:8001/POSPrinterEvent");

//////////////////////////
// Call a method, etc. //
//////////////////////////

// When terminating, terminate also the service for event acquisition.
posPrinterEventServiceHost.Close();
```

LongPolling method

Prepare a thread for LongPolling and write a process for sending an event acquisition request to Service Provider in the Application.cs file.

<Application.cs file>

```
private Thread longPollingThread;
```

❑ **Start a service for event acquisition.**

(Specify null for the second argument.)

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

❑ **Create a thread for LongPolling and start.**

```
longPollingThread = new Thread(new ThreadStart(POSPrinterEvent_LongPolling));
longPollingThread.IsBackground = true;
// The method starts.
longPollingThread.Start();
```

❑ **Method for event processing (the method can be arbitrarily named).**

```
private void POSPrinterEvent_LongPolling() {
  // Processing for event acquisition.
  // For details, refer to the POSPrinterEvent_LongPolling method (p.27).
}
```

POSPrinterEvent_LongPolling method

Write a method for threads by referring to the following:

<Application.cs file>

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

❏ Issue an event acquisition request to Service Provider.

```
      UnifiedPOS.POSPrinter.V1_2.POSPrinterEvent deviceEvent;
      // Execute event polling.
      deviceEvent = device.PollForUPOSEvent(consumerID);

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

      // Execute the process for each of the received events.
```

❏ Write the process when notification of each event is made.

(For details, refer to Procedure for Procedure for Execution of Process for Each of Received Events (p.28).)

```
      if (deviceEvent.ErrorEvent != null) {
        // Process when ErrorEvent is obtained.
      } else if (deviceEvent.OutputCompleteEvent != null) {
        // Process when OutputCompleteEvent is obtained.
      } else if (deviceEvent.StatusUpdateEvent != null) {
        // Process when StatusUpdateEvent is obtained.
      } else if (deviceEvent.DirectIOEvent != null) {
        // Process when DirectIOEvent is obtained.
      }

      // Response to event (response to WS-POS Service Provider).
      while (true) {
        try {
```

❏ Return Response to Service Provider.

When an event notification is received, set necessary information in POSPrinterEventRespon call the SetEventResponse() method.

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

          break;
```

❏ When a timeout occurs, issue an event acquisition request again.

```
        } catch (TimeoutException) {
          continue;

        } catch (FaultException<UnifiedPOS.POSPrinter.V1_2.UposException> ex) {
          if (ex.Detail.ErrorCode == UnifiedPOS.POSPrinter.V1_2.ErrorCode.Illegal) {
            // When ILLEGAL is returned, since SetEventResponse has already been received,
            // go back to PollForUPOSEvent again.
            break;
          } else {
            throw;
          }
        }
      }
```

❏ When a timeout occurs, issue an event acquisition request again.

```
    } catch (TimeoutException) {
      continue;

    }
  }
}
```

Procedure for Execution of Process for Each of Received Events

When an event is received, set necessary information in the UnifiedPOS.POSPrinter.V1_2.POSPrinterEventResponse class and use SetEventResponse() to return a response to Service Provider. Refer to the following:

<Application.cs file>

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

// Execute the process for each of the received events.
```

❑ **Process when ErrorEvent is obtained.**

```
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;
```

❑ **Process when OutputCompleteEvent is obtained.**

```
} 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;
```

❑ **Process when StatusUpdateEvent is obtained.**

```
} 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;
```

❑ **Process when DirectIOEvent is obtained.**

```
} 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;
}
```

❑ **Call SetEventResponse() and process a response to Service Provider.**

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

KeepAlive

After connection is established between the device and the application (OpenSession()), if there is no action between them for a certain period of time (a timeout duration is set by the Service Provider side), the connection between the device and the application will be cut off.
For that reason, a mechanism is provided that performs KeepAlive to prevent the connection between the device and the application from being cut off even after a certain period of time.

A timeout duration is set by the Service Provider side.

<Application.cs file>

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

❑ **Set the KeepAlive process to the timer.**

```
timer.Tick += new System.EventHandler(timer_Tick);

// Start the timer for KeepAlive when the application starts.
int providerSessionTimeout = device.GetProviderSessionTimeout(consumerID);
// Obtain the timeout duration set by the Service Provider.
// (In this document, a half of the timeout duration obtained from Provider is used for
    KeepAlive time.)
timer.Interval = providerSessionTimeout / 2 * 1000;
// Start the timer.
timer.Start();


private void timer_Tick(object sender, EventArgs e) {
  // WS-POS KeepAlive
  if (consumerID != null) {
    try {
```

❑ **Process to call KeepAlive.**

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

❑ **Process to be performed when a timeout occurs.**

- Since connection to Service Provider is offline, set channel information and consumerID to null to perform Open again.
- If an event is obtained through LongPolling, also stop the thread for LongPolling.

```
      timer.Stop();

      // When KeepAlive times out, start again from Open.
      consumerID = null;
      device = null;
    }
  }
}
```

Escape Sequence Specification Procedure

This section describes how to specify an escape sequence by using an example of the PrintNormal method of POSPrinter using an escape sequence.
The example below is a case of formatting "Hello WS-POS!" in bold. For other escape sequences, use the same specification procedure.

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

❑ To format in bold (ESC | bC command)

Escape sequence: **\u001b|bC**

Visual Basic .NET

This document describes how to develop applications in a Visual Basic .NET (Windows Communication Foundation) environment.

Application development using Visual Basic .NET

This chapter describes the following:

Application Environment in This Chapter

In this chapter, descriptions are made by assuming the following environment:

Item	Environment Setup
IP address : port number of Service Provider	192.168.1.100:8087
IP address : port number for event acquisition	192.168.1.102:8001
Device category	POSPrinter
Service class	UnifiedPOS.POSPrinter.V1_2.POSPrinter
Namespace	http://www.nrf-arts.org/UnifiedPOS/POSPrinter/
Base address of POSPrinter service	http://192.168.1.100:8087/POSPrinter.svc
Endpoint of POSPrinter service	POSPrinter1
Program file name for application	Application.vb
Development Environment	Microsoft Visual Studio 2010

Application Project Setup

Creating a new project

Select a Visual Basic Windows application when creating a new project in Visual Studio.

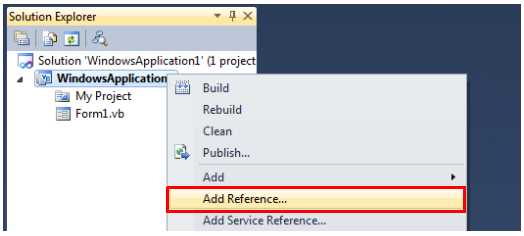
- If you add an application to an existing project, you don't have to create a new project.
- You can also create a new project with a console or Windows form application.

Adding WSPOSContract

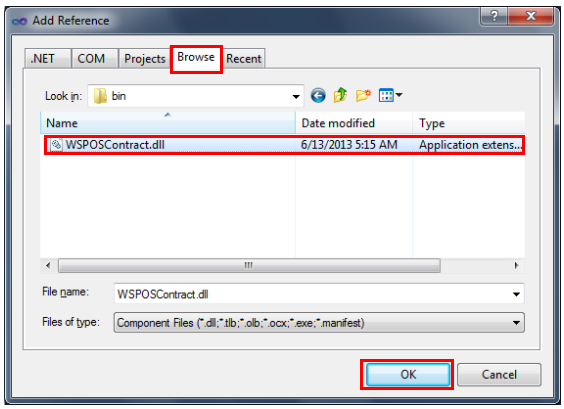
Add WSPOSContract as follows:

1 Save "WSPOSContract.dll", which is included in the reference implementation, into the project folder.

2 In Visual Studio, run Solution Explorer and right-click the project.
Then click (Add Reference).



3 Open the (Browse) tab, select "WSPOSContract.dll" which you saved in step 1, and click (OK).

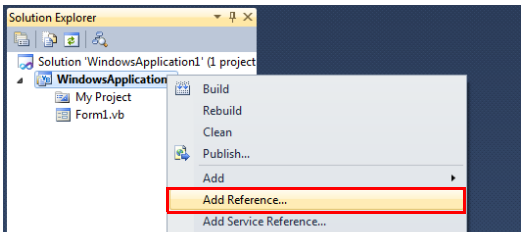


4 Confirm that "WSPOSContract" has been added to the References on Solution Explorer.

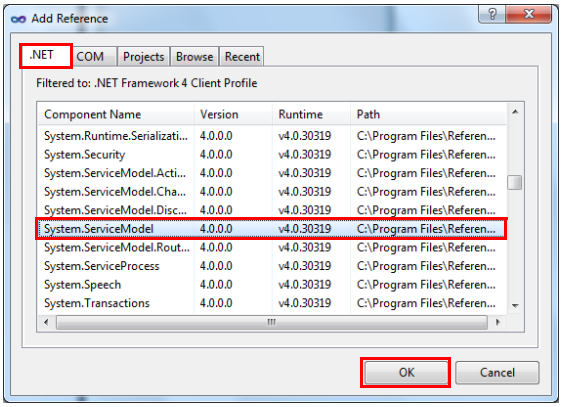
Adding System.ServiceModel

Add System.ServiceModel as follows:

1 In Visual Studio, run Solution Explorer and right-click the project. Then click (Add Reference).



2 Open the (NET) tab, select "System.ServiceModel" which you saved in step 1, and click (OK).



3 Confirm that "System.ServiceModel" has been added to the References on Solution Explorer.

Adding the application configuration file to the project

When a Visual Studio project is created, its application configuration file (App.config) is automatically created.
Add that file to the project as follows:

1 In Visual Studio, click "Show All Files" on Solution Explorer.

2 Confirm that "App.config" has been added to Solution Explorer.

> If the application configuration file (App.config) has not been added, refer to Adding the application configuration file (p.34) and add the application configuration file.

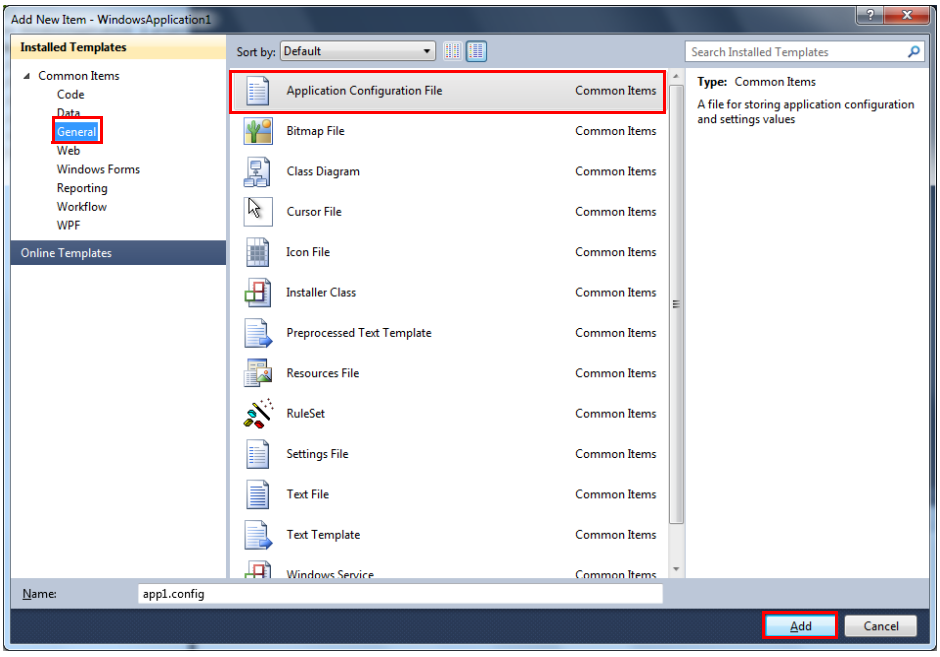
3 Right-click the file "App.config" and select (Include in Project).

Adding the application configuration file

Add the application configuration file as follows:

1 In Visual Studio, select a project on Solution Explorer and select (Project)-(Add New Item) from the menu bar.

2 Select (Common Items)-(General), then (Application Configuration File).



3 Confirm that "App.config" has been added to Solution Explorer.

Adding the application configuration file

Write the statements below in the application configuration file (App.config file).

<App.config file>

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

❏ Writing information about connection with Service Provider (p.35).

```
<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>
```

❏ Setting the maximum number of communication ports simultaneously opened (p.36).

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

```
</configuration>
```


Writing information about connection with Service Provider

Write information about connection with Service Provider in <endpoint>.
For the attributes "address, contract, and binding", write the same pieces of information as those of Service Provider.

<endpoint> attribute	Description
address	Specify the URL of the Service Provider to be used. Specify the IP address correspoding to the URL set by the Service Provider.
contract	Specify the contract name for the device category to be used.
binding	Specify the binding set by the Service Provider. In this chapter, specify "basicHttpBinding".
name	Specify the name to identify connection information. This attribute is used for acquisition of connection information in the application configuration file when Service Consumer connects to Service Provider.

For details on the application configuration file, refer to the following Website:
http://msdn.microsoft.com/en-us/library/ms733932.aspx
(As of Oct. 1st, 2013)

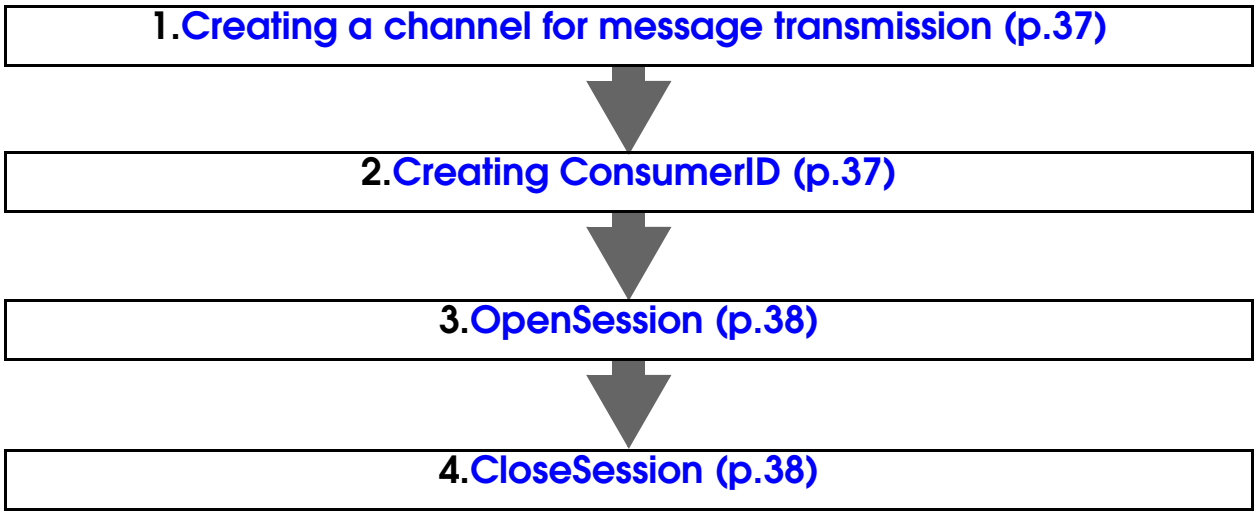
Setting the maximum number of communication ports simultaneously opened

Set the maximum number of communication ports that can be simultaneously opened by an application. Since communication ports are opened on a thread basis, specify the maximum number of communication ports simultaneously opened, based on the number of threads.
Change as needed the value for maxconnection in the <connectionManagement> tag of the <system.net> tag.

- In this section, the maximum value is set as "48" for explanation. Set the value that suits the customer's environment.
- The threads below are used simultaneously in the sample application. Therefore, at least three ports for communication are necessary.
  * Main thread
  * Thread for KeepAlive
  * Thread for Polling

Connection to Service Provider

Write a process necessary for connection to Service Provider in the application file (Application.vb).
The following process is required:



Creating a channel for message transmission

Create a channel for message transmission.

```
Dim device As UnifiedPOS.POSPrinter.V1_2.POSPrinter

' Specify the contract and name for the endpoint set by the application configuration
  file.
Dim factory As New ChannelFactory
                        (Of UnifiedPOS.POSPrinter.V1_2.POSPrinter)("POSPrinterPort")

' Create a channel.
device = factory.CreateChannel()
```

Creating ConsumerID

Create ConsumerID.
ConsumerID is used as an identifier for device identification.

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

In the above example, "GUID" is created and used as ID. Use ID that can be uniquely identified during application development.

OpenSession

Start a session with Service Provider.
After calling OpenSession, you can call each method.

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

CloseSession

Terminate the session with Service Provider.
When each method has been completely processed, call this method to terminate the session with Service Provider.

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

Method Call Procedure

This section describes a series of processes when using the printNormal method.
You can call each method by specifying it for the channel created in Creating a channel for message transmission (p.37).

> ⚠ For WS-POS, when calling each method, consumerID is always required.
> On the device side, the device and Service Consumer are associated with one another based on consumerID.

<Application.vb file>

❑ **Session starts.**

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

❑ **Device Open process.**

```
device.OpenDevice(consumerID)
```

❑ **Claim process.**

```
device.Claim(consumerID, 10000)
```

❑ **Enabled process.**

```
device.SetDeviceEnabled(consumerID, true)
```

❑ **PrintNormal process.**

```
device.PrintNormal(consumerID, UnifiedPOS.POSPrinter.V1_2.PrinterStation.Receipt,
                   "Enter here the text to be printed.")
```

❑ **Disabled process.**

```
device.SetDeviceEnabled(consumerID, False)
```

❑ **Release process.**

```
device.Release(consumerID)
```

❑ **Device Close process.**

```
device.CloseDevice(consumerID)
```

❑ **Session Close process.**

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

Error (Exception) Acquisition Procedure

Enclose a method with try and catch and catch an exception to obtain an error.

<Application.vb file>

```
try
   ' Device open.
   device.OpenDevice(consumerID)

Catch ex As FaultException(Of UnifiedPOS.POSPrinter.V1_2.UposException)
```

❑ Obtain UposException-type detailed information from the caught exception.

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

❑ Obtain ErrorCode and ErrorCodeExtended.

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

Event Acquisition Method

The following event acquisition methods are available:

- SelfHost method (p.41)

  In this method, a service class for receiving an event is provided on the ServiceConsumer side (application), and event notification is made from the Service Provider side to this service class.

  In this section, settings are made by assuming the following environment:

Item	Settings
IP address : port number for event acquisition	192.168.1.102:8001
Service class for event acquisition	POSPrinterEventService
Base address of service for event acquisition	http://192.168.1.102:8001/POSPrinterEvent
Namespace of service class	WSPOSEventService
Device category	POSPrinter
File name of program for service	POSPrinterEventService.vb

- LongPolling method (p.44)

  In this method, an event acquisition request is sent from ServiceConsumer (application) to Service Provider, and event notification is made as its response.

SelfHost method

Create a program for event acquisition as follows:

1 Create a service class for event acquisition using Visual Studio.

Select a project, right click on it and select (Project)-(Add New Item) to add a VB class. Specify the file name as "POSPrinterEventService.vb".

2 Edit the added file "POSPrinterEventService.vb".
Add or delete a method as needed because the required method varies depending on the device category.
Edit by referring to the following:

<POSPrinterEventService.vb file>

```
Namespace WSPOSEventService
```

❑ **Declare items for creating a service for event acquisition.**

```
  <ServiceBehavior(Namespace:="http://www.nrf-arts.org/UnifiedPOS/POSPrinterEvents/
    ", InstanceContextMode:=InstanceContextMode.Single)> _

  public class POSPrinterEventService
    ' Take over the event class of the category to which an event you want to obtain
       belongs.
    Implements UnifiedPOS.POSPrinterEvents.V1_2.POSPrinterEvent
```

❑ **Method to be called when ErrorEvent occurs.**

```
    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
```

❑ **Method to be called when OutputCompleteEvent occurs.**

```
    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
```

❑ **Method to be called when StatusUpdeteEvent occurs.**

```
    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
```

❑ **Method to be called when DirectIOEvent occurs.**

```
    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 Edit the application configuration file (App.config).

Edit the following:

<App.config file>

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

❑ Specify the service name.

Specify the service name as the "namespace+ class name" of the service class for event acquisition. "WSPOSEventService.POSPrinterEventService" is specified in this example.

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

❑ Specify the URL for event acquisition for baseAddress.

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

❑ Specify the endpoint information of the service for event acquisition.

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

For details on the application configuration file, refer to the following Website:
http://msdn.microsoft.com/en-us/library/ms733932.aspx
(As of Oct. 1st, 2013)

4 Set a service for event acquisition in the application.

Write a process for creating a service host for event acquisition in the application file (Application.vb).

<Application.vb file>

```
Dim posPrinterEventServiceHost As ServiceHost

' Start a service for event acquisition.
posPrinterEventServiceHost =
 New ServiceHost(GetType(WSPOSEventService.POSPrinterEventService))

' Pass Endpoint using an OpenSession argument.
device.OpenSession(consumerID, "http://192.168.1.102:8001/POSPrinterEvent")

''''''''''''''''''''''''
' Call a method, etc. '
''''''''''''''''''''''''

' When terminating, terminate also the service for event acquisition.
posPrinterEventServiceHost.Close()
```

LongPolling method

Prepare a thread for LongPolling and write a process for sending an event acquisition request to Service Provider in the Application.vb file.

<Application.vb file>

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

❑ Open a session.

(Specify null for the second argument.)

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

❑ Create a thread for LongPolling and start.

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

❑ Method for event processing (the method can be arbitrarily named).

```
Private Sub POSPrinterEvent_LongPolling()
  ' Here, write the process when an event is obtained.
  ' For details, refer to the POSPrinterEvent_LongPolling method (p.45).
End Sub
```

POSPrinterEvent_LongPolling method

Write a method for threads by referring to the following:

<Application.vb file>

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

❑ **Issue an event acquisition request to Service Provider.**

```
      Dim deviceEvent As UnifiedPOS.POSPrinterEvents.V1_2.POSPrinterEvent
      ' Execute event polling.
      deviceEvent = device.PollForUPOSEvent(consumerID)

      Dim res As New UnifiedPOS.POSPrinterEvents.V1_2.POSPrinterEventResponse

      ' Execute the process for each of the received events.
```

❑ Write the process when notification of each event is made.

(For details, refer to Procedure for Execution of Process for Each of Received Events (p.46).)

```
      If Not deviceEvent.ErrorEvent Is Nothing Then
        ' Process when ErrorEvent is obtained.
      ElseIf Not deviceEvent.OutputCompleteEvent Is Nothing Then
        ' Process when OutputCompleteEvent is obtained.
      ElseIf Not deviceEvent.StatusUpdateEvent Is Nothing Then
        ' Process when StatusUpdateEvent is obtained.
      ElseIf Not deviceEvent.DirectIOEvent Is Nothing Then
        ' Process when DirectIOEvent is obtained.
      End If

      ' Response to event (response to WS-POS Service Provider).
      While (True)

        Try
```

❑ **Return Response to Service Provider.**

When an event notification is received, set necessary information in POSPrinterEventResponse and call the SetEventResponse() method.

```
          device.SetEventResponse(consumerID, res)

          Exit Try
```

❑ When a timeout occurs, issue an event acquisition request again.

```
        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
          ' When ILLEGAL is returned, since SetEventResponse has already been received,
          ' go back to PollForUPOSEvent again.
            Exit While
          Else
            Throw
          End If
        End Try
    End While
```

❑ **When a timeout occurs, issue an event acquisition request again.**

```
    Catch ex As TimeoutException
      Continue While
    End Try

  End While
End Sub
```

Procedure for Execution of Process for Each of Received Events

When an event is received, set necessary information in the UnifiedPOS.POSPrinter.V1_2.POSPrinterEventResponse class and use SetEventResponse() to return a response to Service Provider.
Refer to the following:

<Application.vb file>

```
Dim res As New UnifiedPOS.POSPrinter.V1_2.POSPrinterEventResponse

' Execute the process for each of the received events.
```

❑ **Process when ErrorEvent is obtained.**

```
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
```

❑ **Process when OutputCompleteEvent is obtained.**

```
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
```

❑ **Process when StatusUpdateEvent is obtained.**

```
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
```

❑ **Process when DirectIOEvent is obtained.**

```
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
```

❑ **Call SetEventResponse() and process a response to Service Provider.**

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

KeepAlive

After connection is established between the device and the application (OpenSession()), if there is no action between them for a certain period of time (a timeout duration is set by the Service Provider side), the connection between the device and the application will be cut off.
For that reason, a mechanism is provided that performs KeepAlive to prevent the connection between the device and the application from being cut off even after a certain period of time.

A timeout duration is set by the Service Provider side.

<Application.vb file>

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

❑ **Set the KeepAlive process to the timer.**

```
AddHandler timer.Tick, New EventHandler(AddressOf timer_Tick)

' Start the timer for KeepAlive when the application starts.
Dim providerSessionTimeout As Integer = device.GetProviderSessionTimeout(consumerID)
' Obtain the timeout duration set by the Service Provider.
'' (In this document, a half of the timeout duration obtained from Provider is used for
    KeepAlive time.)
timer.Interval = ((providerSessionTimeout / 2) * 1000)
'' Start the timer.
timer.Start()


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

❑ **Process to call KeepAlive.**

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

❑ **Process to be performed when a timeout occurs.**

- Since connection to Service Provider is offline, set channel information and consumerID to null to perform Open again.
- If an event is obtained through LongPolling, also stop the thread for LongPolling.

```
      timer.Stop()

      ' When KeepAlive times out, start again from Open.
      consumerID = Nothing
      device = Nothing
    End Try
  End If
End Sub
```

Escape Sequence Specification Procedure

This section describes how to specify an escape sequence by using an example of the PrintNormal method of POSPrinter using an escape sequence.
The example below is a case of formatting "Hello WS-POS!" in bold. For other escape sequences, use the same specification procedure.

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

❏ To format in bold (ESC | bC command)

Escape sequence: **\u001b|bC**

JavaScript

This chapter describes how to develop applications in a JavaScript environment.

Application Development Using JavaScript

WS-POS services perform communication between Service Provider and Service Consumer through SOAP message transmission.
This document describes the following:

- SOAP Message Transmission Procedure (p.50)
- SOAP Message Creation Procedure (p.53)
- Error (Exception) Acquisition Procedure (p.54)
- Event Acquisition Method (p.55)
- KeepAlive (p.60)
- Escape Sequence Specification Procedure (p.61)

> When Windows Service is selected for the WS-POS service environment, applications developed with JavaScript cannot be used. Since WS-POS reference implementation has cross-domain access restrictions, the configuration method of Internet Information Services environment is described in "WS-POS Environment Setup Guide" as one of examples to avoid them. This chapter describes how to develop applications using JavaScript based on that environment.

Application Environment in This Chapter

In this chapter, descriptions are made by assuming the following environment:

Item	Environment Setup
IP address : port number of Service Provider	192.168.1.100:8087
Device category	POSPrinter
Namespace	http://www.nrf-arts.org/UnifiedPOS/POSPrinter/
Base address of POSPrinter service	http://192.168.1.100:8087/POSPrinter.svc
Endpoint of POSPrinter service	POSPrinter1
JavaScript file name	POSPrinterControl.js

SOAP Message Transmission Procedure

For SOAP message transmission, the processes below should be performed. Refer to the following:

<POSPrinterControl.js file>

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

❑ **Create an XMLHttpRequest object.**

```
// Call the method for creating an XMLHttpRequest object.
var xmlhttp = createHttpRequest();
```

❑ **SOAP Message Creation Procedure (p.53).**

```
// Create a SOAP message to send.
var message = createWSPOSMessage(ID);
```

❑ **Preparing transmission of a request to Service Provider (p.51).**

```
// Prepare transmission of a request message.
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)};
```

❑ **Send a request message.**

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

```
// Create the XMLHttpRequest object for each browser.
function createHttpRequest(){
    // For Internet Explorer
    if(window.ActiveXObject){
        try {
            // For MSXML2 or later
            return new ActiveXObject("Msxml2.XMLHTTP");
        } catch (e) {
            try {
                // For previous MSXML
                return new ActiveXObject("Microsoft.XMLHTTP");
            } catch (e2) {
                return null;
            }
         }
    // For any XMLHttpRequest-object-implemented browser except Internet Explorer
    } else if(window.XMLHttpRequest){
        return new XMLHttpRequest();
    } else {
        return null;
    }
}
```

In the above exmple, a process for calling the PrintNormal method is performed.

Preparing transmission of a request to Service Provider

Execute the following processes:

1 Specify as follows by using the open method:

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

Parameter	Value
Request type	Specify "POST" for request type because SOAP sends messages using the POST method.
Service's URL	Specify the URL of the Service Provider to be used.
Synchronous communication	Specify false (specify according to the environment).

2 Set a header for HTTP request by using the setRequestHeader() method.

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

3 Specify a callback function when an onreadystatechange occurs.

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

<Callback function to be specified>

```
function callback (xmlhttp){
  // When the request processing state is ready.
  if (xmlhttp.readyState == 4) {
    if (xmlhttp.status == 200) {
      // Process when the http status is normal (200).
    } else {
      // Process when the http status is abnormal (not 200).
    }
  }
}
```

Method Call Procedure

This section describes the order of methods to be called when using the printNormal method.
For details on how to call methods, refer to SOAP Message Transmission Procedure (p.50).

❏ **Session starts.**
`OpenSession`
❏ **Device Open process.**
`OpenDevice`
❏ **Claim process.**
`Claim`
❏ **Enabled process.**
`SetDeviceEnabled`
❏ **PrintNormal process.**
`PrintNormal`
❏ **Disabled process.**
`SetDeviceEnabled`
❏ **Release process.**
`Release`
❏ **Device Close process.**
`CloseDevice`
❏ **Session Close process.**
`CloseSession`

SOAP Message Creation Procedure

This section describes a message example when sending a request for the printNormal method.
The contents to be written vary depending on the method. Write codes by referring to the following:

Contents of a SOAP message to be sent

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

    ❑ Call the printNormal method.
    <PrintNormal xmlns="http://www.nrf-arts.org/UnifiedPOS/POSPrinter/">

      ❑ Specify the printNormal method parameters.
        For details on method parameters, refer to Parameter Definition (p.53).
        <ConsumerID>1234567890</ConsumerID>
        <Station>Receipt</Station>
        <Data>Write here the text to be printed.</Data>

    </PrintNormal>

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

Use the ConsumerID that can be uniquely identified during application development.

Parameter Definition

The parameters for each method are defined in the "POSPrinterV1.13.2.xsd" file which is enclosed with the WS-POS specifications.
Each <element> tag in the <sequence> tag, which exists in the <element> tag of the method name, is a parameter.
The following shows how the printNormal method parameters are defined:

❑ **Method for defining parameters.**

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

      ❑ Parameter definition.
      <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>
```

Error (Exception) Acquisition Procedure

If a sent SOAP message fails to be received, an exception occurs.
By writing the following process within the callback function, if an exception occurs, you can obtain the exception details of UposException.

<POSPrinterControl.js file>

```
// Add to the callback function process.
var responseXML = xmlhttp.responseXML;

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

❑ **An error message is written in the faultstring tag.**

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

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

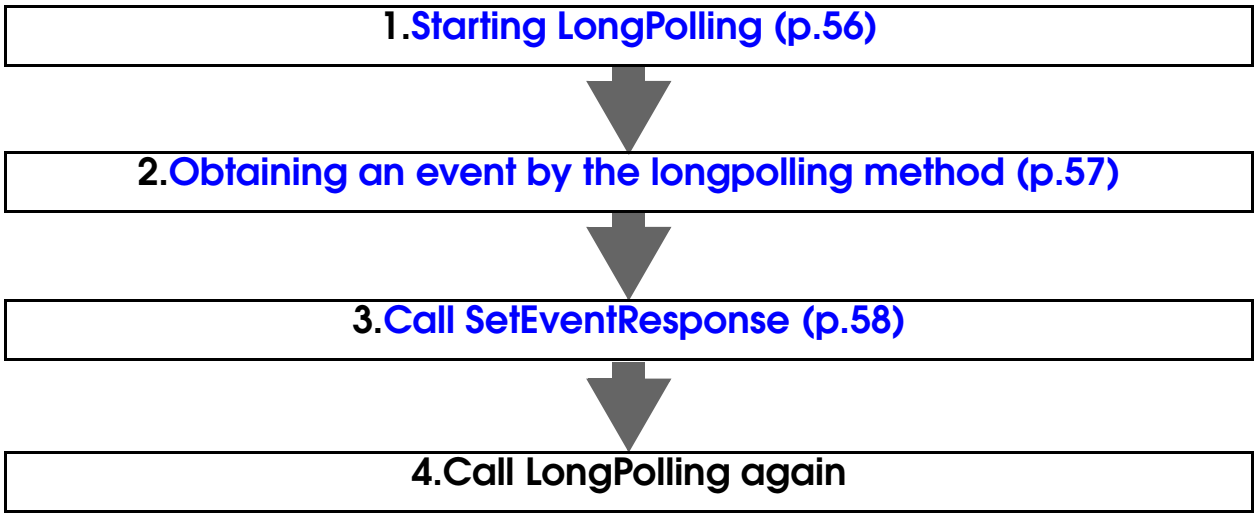
❑ **An UposException exception is written in the UposException tag.**

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

For details on the specifications for UposException, refer to the file "POSPrinterV1.13.2.xsd".

Event Acquisition Method

Obtain an event using the LongPolling method.
In this method, an event acquisition request is sent from ServiceConsumer (application) to Service Provider, and event notification is made as its response.
The following process is required:



For details on how to send a SOAP message for LongPolling, refer to SOAP Message Transmission Procedure (p.50).

Starting LongPolling

Specify a LongPolling callback function to obtain an event to send a SOAP message.
Write codes by referring to the following:

<POSPrinterControl.js file>

```
var xmlhttp_longpolling;
```

❏ **Create an XMLHttpRequest object.**

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

❏ **Create a SOAP message to send.**

For details, refer to SOAP Message to Send (p.56).

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

❏ **Prepare transmission of a SOAP message.**

```
// Specify asynchronous communication by using the open method.
xmlhttp_longpolling.open("POST", "http://192.168.1.100:8087/POSPrinter.svc/POSPrinter1",
true);
// Set a header for HTTP request by using the setRequestHeader method.
xmlhttp_longpolling.setRequestHeader("Content-Type", "text/xml; charset=UTF-8");
xmlhttp_longpolling.setRequestHeader("SOAPAction","\"http://www.nrf-arts.org/UnifiedPOS/
POSPrinter/PollForUPOSEvent\"");
// Specify a LongPolling callback function to be called when an event occurs (p. 57).
xmlhttp_longpolling.onreadystatechange = function(){longpolling(xmlhttp_longpolling)};
```

❏ **Send a SOAP message.**

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

SOAP Message to Send

The contents of a SOAP message to send are as follows:

```
<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>
```

Obtaining an event by the longpolling method

Write a callback function (longpolling method) to be called when an event occurs.
Write codes by referring to the following:

<POSPrinterControl.js file>

```
function longpolling(xmlhttp) {

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

❑ **Process when the http status is normal (http status: 200).**

Analyze an event and return SetEventResponse to Service Provider. For details on SetEventResponse, refer to Call SetEventResponse (p.58).

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

❑ **Process when the http status is abnormal (http status: not 200).**

```
    }
  }
}
```

Call SetEventResponse

Analyze an event and return SetEventResponse to Service Provider.

- In this section, the device category, POSPrinter, is used as an example. Depending on the device category to be used, the corresponding event varies. Write codes for an event as needed.
- For SetEventResponse, necessary parameters change on an event basis. Check the definition.

<POSPrinterControl.js file>

```
function longPollingResponse(xmlhttp){

  var responseXML = xmlhttp.responseXML;

  // ErrorEvent
  ❑ Obtain ErrorEvent tag information from XML data.
  var errorEvent = responseXML.getElementsByTagName("ErrorEvent");
  if( errorEvent.length > 0) {
    ❑ The process when ErrorEvent is obtained (if information exists).
      For details, refer to Processing for event acquisition (p.59).
  }

  // OutputCompleteEvent
  //  Obtain OutputCompleteEvent tag information from XML data.
  var outputCompEvent = responseXML.getElementsByTagName("OutputCompleteEvent");
  if( outputCompEvent.length > 0) {
    // The process when OutputCompleteEvent is obtained (if information exists).
  }

  // StatusUpdateEvent
  //  Obtain StatusUpdateEvent tag information from XML data.
  var statusUpdateEvent = responseXML.getElementsByTagName("StatusUpdateEvent");
  if( statusUpdateEvent.length > 0) {
    // The process when StatusUpdateEvent is obtained (if information exists).
  }

  // DirectIOEvent
  //  Obtain DirectIOEvent tag information from XML data.
  var directIO = responseXML.getElementsByTagName("DirectIOEvent");
  if( directIO.length > 0) {
    // The process when DirectIOEvent is obtained (if information exists).
  }
}
```

Processing for event acquisition

Analyze the obtained event data as needed and send a notification of process completion to Service Provider by using the errorEvent_setEventResponse method. When the above notification is correctly sent, Service Provider responds with SetEventResponse. When that response is confirmed, call LongPolling again.

In this section, ErrorEvent is used as an example. Also analyze any other event by referring to the example below and call SetEventResponse.

<POSPrinterControl.js file>

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

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

❑ **Obtain and analyze event data.**

```
    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;
```

❑ **Call the errorEvent_setEventResponse method.**

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

KeepAlive

After connection is established between the device and the application (OpenSession()), if there is no action between them for a certain period of time, the connection between the device and the application will be cut off.
For that reason, a mechanism is provided that performs KeepAlive to prevent the connection between the device and the application from being cut off even after a certain period of time.

- A timeout duration is set by the Service Provider side.
- For details on how to call the KeepAlive function, refer to SOAP Message Transmission Procedure (p.50).

The following shows a method for periodically calling the KeepAlive function by using the timer function of JavaScript.

<POSPrinterControl.js file>

❑ **Start KeepAlive (start the timer).**
Periodically call the KeepAlive function.

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

function KeepAlive (GUID) {
```

❑ **Create and send a message for calling the KeepAlive method.**

```
    // Create a Http request message.
    var xmlhttp = createHttpRequest();
    // Create a KeepAlive method message.
    var keepAliveMessage = createWSPOSMessage_keepAlive(GUID);

    // Prepare and send a request message.
    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);
}
```

❑ **Stop KeepAlive (stop the timer).**

```
clearInterval(timerID);
```

Escape Sequence Specification Procedure

This section describes how to specify an escape sequence by using an example of the PrintNormal method of POSPrinter using an escape sequence.
The example below is a case of formatting "Hello WS-POS!" in bold and start a new line. For other escape sequences, use the same specification procedure.

```
<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>
```

- To format in bold (ESC | bC command)

  Escape sequence: **&#x1b;|bC**

- To specify line feed (\n)

  Escape sequence: **
**