

the professional
labeling software

www.nicelabel.com, info@nicelabel.com



Pocket NiceLabel Programming Guide

Version 20080925

© 2008 Euro Plus d.o.o.

All rights reserved.

www.nicelabel.com



Head Office

Euro Plus d.o.o.
Poslovna cona A2
4208 Šenčur
Slovenia
Tel.: +386 4 280 50 00
Fax: +386 4 233 11 48

www.europlus.si
info@europlus.si

Table of Contents

- 1 Introduction4**
 - 1.1 Who is this Guide for?.....4**
 - 1.2 Sample Code4**
 - 1.3 Sample Applications.....4**

- 2 Getting Started5**
 - 2.1 Using Pocket NiceLabel Engine5**
 - 2.1.1 Installation.....5
 - 2.1.2 Developing your Application5
 - 2.1.3 Deploying your Application6

- 3 PocketNiceEngine Class Library7**
 - 3.1 PocketNiceEngine Namespace7**
 - 3.1.1 EngineClassFactory Class7
 - 3.1.2 IEngine Interface.....9
 - 3.1.3 ILabel Interface12
 - 3.1.4 IOutput Interface14
 - 3.1.5 IVariable Interface.....21
 - 3.1.6 IPromptVariable Interface24
 - 3.1.7 IDateTimeVariable Interface25
 - 3.1.8 ILabelVariables Interface25
 - 3.1.9 IFormat Interface26
 - 3.1.10DataKindType Enum27
 - 3.1.11FormatKind Enum27
 - 3.1.12InputType Enum28
 - 3.1.13OutputKindType Enum28
 - 3.1.14ProductionKind28
 - 3.1.15PromptType Enum28
 - 3.1.16StepKind Enum29

- 4 ErrorHandler Class Library30**
 - 4.1 ErrorHandler Namespace.....30**
 - 4.1.1 ErrorException Class30
 - 4.1.2 ErrorID Enum31

- 5 Appendix34**
 - Euro Plus d.o.o. and Niceware International, LLC34
 - NiceLabel Product Overview.....34
 - NiceLabel Standard Series34
 - NiceLabel Enterprise Series35
 - NiceLabel Developer Series35

- 6 Online Support36**

- 7 Contact Information37**

Disclaimer

Euro Plus d.o.o. & Niceware® International, LLC reserve the right to make changes in specifications and other information contained in this document without prior notice, and the reader should in all cases consult Euro Plus d.o.o. & Niceware® International, LLC to determine whether any such changes have been made. The information in this publication does not represent a commitment on the part of Euro Plus d.o.o. or Niceware® International, LLC. Euro Plus d.o.o. & Niceware® International, LLC shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material. This document contains proprietary information that is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced, or translated into another language without the prior written consent of Euro Plus d.o.o. & Niceware® International, LLC.

Web Addresses: www.europlus.si, www.nicewareintl.com

Trademarks

NiceLabel®, NiceLabel Pro®, NiceLabel PocketSDK®, NiceLabel WebSDK®, NiceLabel SDK®, and NiceDriver® are trademarks or registered trademarks of Euro Plus d.o.o in the U.S.A. and other countries. Niceware® is a registered trademark of Niceware International, LLC.

Microsoft, Visual Studio, Visual C#, Visual Basic, Windows, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation.

Other brands and their products are trademarks or registered trademarks of their respective holders and should be noted as such.

1 Introduction

Welcome to the *Pocket NiceLabel Programming Guide*. Pocket NiceLabel Engine provides all of the tools necessary to develop applications for printing from mobile devices. These tools include class libraries, sample applications, and associated documentation. Pocket NiceLabel Engine allows developers to programmatically access Pocket NiceLabel on mobile devices.

Pocket NiceLabel Engine Programming Guide is designed to assist with the development of custom mobile applications that rely on the mobile printing power of Pocket NiceLabel Engine.

1.1 Who is this Guide for?

This guide is for programmers who want to leverage the full potential of integrating Pocket NiceLabel into mobile applications. Because Pocket NiceLabel Engine can be referenced from a wide range of modern programming environments, this guide will not attempt to act as a programming tutorial. It does contain a concise explanation of the PocketNiceEngine and ErrorHandler namespaces along with associated sample code.

1.2 Sample Code

Visual Basic .NET and Visual C# .NET sample code is provided for most of the commonly used members throughout this documentation. Code is shown in the **Sample Use** subsection of each member.

1.3 Sample Applications

Complete sample .NET Compact Framework 2.0 (Visual Studio 2005) applications can be found installed at <C:\Program Files\Pocket NiceLabel\Sample Applications\VS2005>. Each project has source code for both VB and C#.

SimpleSample Project

SimpleSample demonstrates the basics of handling a label, a variable, the output port, and printing. With the exception of prompting the user for a variable's value, all settings are hard-coded for the sake of simplicity and clarity.

UserSelect Project

This project provides a more user friendly interface and demonstrates dynamically working with unknown labels and variable collections as well as letting the user select the printing port.

2 Getting Started

Pocket NiceLabel Engine is a set of Microsoft.NET Compact Framework Assemblies that expose programmatic interfaces to Pocket NiceLabel. These interfaces expose the same mobile label printing components that Pocket NiceLabel utilizes.

Pocket NiceLabel Engine gives custom mobile applications the ability to control label printing functions. Your mobile application can print labels using Pocket NiceLabel Engine as the print server. Pocket NiceLabel Engine will handle the opening of LVX (Pocket NiceLabel) files and the printing of labels to printers on the following ports: TCP/IP (Wi-Fi), Bluetooth and COM (serial). Pocket NiceLabel Engine becomes the perfect bolt-on label printing engine for mobile applications. Pocket NiceLabel Engine also has the ability to create a JOB file instead of printing directly to a port with a printer specific output file. The JOB file can be sent over a TCP/IP socket connection or be saved on a remote hard disk or network drive where the NiceWatch print server software will execute the NiceCommands contained within the JOB file.

2.1 Using Pocket NiceLabel Engine

2.1.1 Installation

The NiceLabel 5 Standard Series Installer will deploy Pocket NiceLabel to a docked mobile device as well as copy Pocket NiceLabel Engine resource files to the PC.

Development PC

The following development files will be installed at <C:\Program Files\Pocket NiceLabel\>.

- PocketNiceEngine.dll – Class library for the PocketNiceEngine namespace.
- ErrorHandler.dll – Class library for the ErrorHandler namespace.
- Sample Applications directory – Root directory for integration sample applications.

Mobile Device

The installer will install the following on any CE.NET mobile device docked to the PC

- Pocket NiceLabel – Contains the runtime resources for Pocket NiceLabel Engine.
- Microsoft .NET Compact Framework – Necessary for Pocket NiceLabel and any custom applications utilizing Pocket NiceLabel Engine.

2.1.2 Developing your Application

Microsoft Visual Studio

PocketNiceEngine and ErrorHandler namespaces are automatically registered in the PC's Global Assembly Cache (GAC) at installation time. To add a reference to these in your project use the Project drop-down and select "Add Reference..." Then select PocketNiceEngine and/or ErrorHandler from the list on the .NET tab. Click OK and the class libraries are now available to your program.

Please refer to Section 3 - PocketNiceEngine Class Library and Section 4 – ErrorHandler Class Library for details on usage.

Other Development Environments

Please follow your specific development environment's documentation for interfacing with .NET managed assemblies.

2.1.3 Deploying your Application

Visual Studio will automatically deploy all necessary reference and resource files with your application. No additional steps are necessary.

3 PocketNiceEngine Class Library

3.1 PocketNiceEngine Namespace

Classes from this namespace are not directly (word internal) accessible from external applications. External applications must use **EngineClassFactory** when they are using Pocket Engine functionality.

PocketNiceEngine assembly is communicating with external applications thru its public interfaces:

- **IEngine** – Provides functionality for opening and closing the labels
- **ILabel** – Label interface, provides functionality for actual printing
- **IOutput** – Used by ILabel for access to print device
- **IVariable** – Variable interface
- **IPromptVariable** – Prompt variable interface (**IVariable** extension)
- **IDateTimeVariable** – Date time variable interface (**IVariable** extension)
- **IFormat** – Data format
- **ILabelVariables** – Label variables collection

3.1.1 EngineClassFactory Class

This is the only class visible outside PocketNiceEngine assembly capable of creating new instances. Its purpose is to act as a class factory for other parts of Print Assembly.

Methods:

CreateEngine

Description:

Method returns new instance of IEngine interface

Signature:

Public Shared Function **CreateEngine()** As **PocketNiceEngine.IEngine**

Sample Use:

```
[Visual Basic]
Dim pneEngine As PocketNiceEngine.IEngine
pneEngine = PocketNiceEngine.EngineClassFactory.CreateEngine
```

```
[C#]
PocketNiceEngine.IEngine pneEngine;
pneEngine = PocketNiceEngine.EngineClassFactory.CreateEngine();
```

CreateFormat

Description:

Method returns a new instance of IFormat interface.

Signature:

Public Shared Function **CreateFormat**(ByVal *kind* As **PocketNiceEngine.FormatKind**)
As **PocketNiceEngine.IFormat**

Parameters:

Name	Data Type	Description
kind	FormatKind	Data type (Numeric, Date, Hex, etc) of Format

CreateLabelVariables**Description:**

Method returns a new instance of ILabelVariables interface.

Signature:

Public Shared Function **CreateLabelVariables**() As **PocketNiceEngine.ILabelVariables**

Sample Use:

[Visual Basic]

```
Dim pneVariables As PocketNiceEngine.ILabelVariables
pneVariables = pneLabel.LabelVariables
```

[C#]

```
PocketNiceEngine.IVariable pneVariable;
pneVariables = pneLabel.LabelVariables;
```

CreateOutput**Description:**

Method returns a new instance of IOutput interface.

Signature:

Public Shared Function **CreateOutput**() As **PocketNiceEngine.IOutput**

Sample Use:

[Visual Basic]

```
Dim pneOutput As PocketNiceEngine.IOutput
pneOutput = PocketNiceEngine.EngineClassFactory.CreateOutput
```

[C#]

```
PocketNiceEngine.IOutput pneOutput;
pneOutput = PocketNiceEngine.EngineClassFactory.CreateOutput();
```

CreateVariable (+1 overloads)**Description:**

Method returns a new instance of IVariable interface.

Signature:

Public Shared Function **CreateVariable**(ByVal *kind* As **PocketNiceEngine.InputType**)
As **PocketNiceEngine.IVariable**

Public Shared Function **CreateVariable**(ByVal *kind* As **PocketNiceEngine.InputType**,
ByVal *format* As **PocketNiceEngine.IFormat**) As **PocketNiceEngine.IVariable**

Parameters:

Name	Data Type	Description
kind	InputType	Input type (Prompt, Database, Date, etc...) of variable
format	IFormat	Data format type (Numeric, Date, Hex, etc...)

Note: Usually only Prompt variables are used outside of the assembly.

3.1.2 IEngine Interface

Methods:

Init

Description:

Initializes Pocket NiceLabel Engine Assembly

Signature:

```
Sub Init()
```

Sample Use:

```
[Visual Basic]
pneEngine.Init()
```

```
[C#]
pneEngine.Init();
```

OpenLabel

Description:

New label is opened by calling **OpenLabel** and providing the full path to the *.LVX file, if the file is valid and the label data is successfully loaded new ILabel instance created. ILabel reference is returned if operation is successful null otherwise.

If label is already loaded only reference to the existing label is returned.

Signature:

Function **OpenLabel**(ByVal *path* As String) As **PocketNiceEngine.ILabel**

Parameters:

Name	Data Type	Description
path	String	Full path to .lvx label file.

Sample Use:

```
[Visual Basic]
pneLabel = pneEngine.OpenLabel(LabelPath)
```

```
[C#]
pneLabel = pneEngine.OpenLabel(LabelPath);
```

CloseLabel

Description:

CloseLabel removes ILabel instance from the label list.

Signature:

Sub **CloseLabel**(ByVal *lbl* As **PocketNiceEngine.ILabel**)

Parameters:

Name	Data Type	Description
lbl	ILabel	Label object to close

Sample Use:

[Visual Basic]
 pneEngine.CloseLabel(pneLabel)

[C#]
 pneEngine.CloseLabel(pneLabel);

Registration

Description:

Register (enter registration code)

Signature:

Function **Registration**(ByVal *code* As String, ByVal *user* As String, ByVal *company* As String) As Boolean

Parameters:

Name	Data Type	Description
code	String	Your Registration Code
user	String	Your User Name
company	String	Your Company Name

Sample Use:

[Visual Basic]
 pneEngine.Registration("My Reg Code", "John", "XYZ, Inc.")

[C#]
 pneEngine.Registration("My Reg Code", "John", "XYZ, Inc.");

Properties:

DefaultOutput

Description:

Sets or returns default Output object.

Signature:

Property **DefaultOutput**() As PocketNiceEngine.IOutput

Sample Use:

```
[Visual Basic]
pneLabel.Output = pneEngine.DefaultOutput

[C#]
pneLabel.Output = pneEngine.DefaultOutput;
```

LabelList

Description:

Returns a collection of ILabel objects.

Signature:

ReadOnly Property **LabelList**() As System.Collections.Generic.IList(Of PocketNiceEngine.ILabel)

Sample Use:

```
[Visual Basic]
Dim pneLabels As PocketNiceEngine.ILabel()
pneLabels = CType(pneEngine.LabelList, PocketNiceEngine.ILabel())

[C#]
PocketNiceEngine.ILabel[] pneLabels;
pneLabels = ((PocketNiceEngine.ILabel[])(pneEngine.LabelList));
```

NumberOfLicenses

Description:

Returns number of Pocket NiceLabel licenses that registration provides.

Signature:

ReadOnly Property **NumberOfLicenses**() As Integer

Sample Use:

```
[Visual Basic]
Dim NumOfUsers As Int32 = pneEngine.NumberOfLicenses

[C#]
Int32 NumOfUsers = pneEngine.NumberOfLicenses;
```

UserName

Description:

Returns User name provided at registration time.

Signature:

ReadOnly Property **UserName**() As String

Sample Use:

```
[Visual Basic]
Dim RegUser As String = pneEngine.UserName

[C#]
string RegUser = pneEngine.UserName;
```

Company

Description:

Returns Company name provided at registration time.

Signature:

ReadOnly Property **Company**() As String

Sample Use:

```
[Visual Basic]
Dim RegCompany As String = pneEngine.Company

[C#]
string RegCompany = pneEngine.Company;
```

Code

Description:

Returns Registration Code

Signature:

ReadOnly Property **Code**() As String

Sample Use:

```
[Visual Basic]
Dim RegCode As String = pneEngine.Code

[C#]
string RegCode = pneEngine.Code;
```

3.1.3 ILabel Interface

Currently the only supported label file type is *.LVX (XML). Label data is loaded by calling the **Load** method. Label members (like variables, functions, etc...) are created at this time and their properties set in accordance to data in LVX file

IOutput class is responsible for providing communication between Engine and the external device (printer, PC, etc...). IOutput must not be null when print command is called.

Currently two production types (JOB, direct printing) are supported (ProductionType property)

Methods:

Print (+1 overloads)

Description:

Print current label

Signatures:

```
Sub Print(ByVal param As String)
Sub Print(ByVal param As String, ByVal kind As PocketNiceEngine.ProductionKind)
```

Parameters:

Name	Data Type	Description
param	String	Quantity of labels to print
kind	ProductionKind	Should not be used externally

Note:

Print method throws an exception if an error occurs.

Sample Use:

```
[Visual Basic]
pneLabel.Print("1")

[C#]
pneLabel.Print("1");
```

Load**Description:**

Load label

Signature:

Sub **Load**(ByVal *path* As String)

Parameters:

Name	Data Type	Description
path	String	Label file path

Note: External applications should use the IEngine.OpenLabel method instead of Load().

Properties:

Name**Description:**

Returns Label file's Name

Signature:

ReadOnly Property **Name**() As String

Sample Use:

```
[Visual Basic]
Dim LabelName As String = pneLabel.Name

[C#]
string LabelName = pneLabel.Name;
```

Path**Description:**

Returns Label file's Path

Signature:

ReadOnly Property **Path()** As String

Sample Use:

```
[Visual Basic]
Dim LabelPath As String = pneLabel.Path

[C#]
string LabelPath = pneLabel.Path;
```

Output

Description:

Sets or returns label Output object

Signature:

Property **Output()** As **PocketNiceEngine.IOutput**

Sample Use:

```
[Visual Basic]
pneLabel.Output = pneOutput

[C#]
pneLabel.Output = pneOutput;
```

LabelVariables

Description:

Sets or returns a collection of the label's variables.

Signature:

Property **LabelVariables()** As **PocketNiceEngine.ILabelVariables**

Sample Use:

```
[Visual Basic]
pneVariables = pneLabel.LabelVariables

[C#]
pneVariables = pneLabel.LabelVariables;
```

ProductionType

Description:

Sets or returns label's ProductionKind object

Signature:

Property **ProductionType()** As **PocketNiceEngine.ProductionKind**

3.1.4 IOutput Interface

IOutput is used by assembly (when printing) for communication with the output devices (Printer, PC, etc...). It is the only part of Engine which can access peripheral devices.

Methods:

Open

Description:

Open connection with selected output device

Signature:

Sub **Open**()

Note: The Open, Send, and Close methods are all handled internally by the ILabel.Print method and typically are not used externally.

Close

Description:

Close connection with selected device.

Signature:

Sub **Close**()

Note: The Open, Send, and Close methods are all handled internally by the ILabel.Print method and typically are not used externally.

Set

Description:

Sets current label's port from another IOutput object.

Signature:

Sub **Set**(ByVal *output* As PocketNiceEngine.IOutput)

Parameters:

Name	Data Type	Description
output	IOutput	Used another IOutput interface to copy setting from.

SetBlueTooth

Description:

Sets BlueTooth port parameters.

Signature:

Sub **SetBlueTooth**(ByVal *port* As Long)

Parameters:

Name	Data Type	Description
port	Long	Virtual COM port used by the blue tooth

Sample Use:

```
[Visual Basic]
pneOutput.Kind = PocketNiceEngine.OutputKindType.BlueTooth
pneOutput.SetBlueTooth(5)
```

```
[C#]
pneOutput.Kind = PocketNiceEngine.OutputKindType.BlueTooth;
pneOutput.SetBlueTooth(5);
```

SetCom

Description:

Sets COM port parameters.

Signature:

Sub **SetCom**(ByVal *port* As Long, ByVal *baudRate* As Integer, ByVal *dataBits* As Integer, ByVal *parity* As System.IO.Ports.Parity, ByVal *stopBits* As System.IO.Ports.StopBits)

Parameters:

Name	Data Type	Description
port	Long	COM port (e.g. COM 8,..)
baudRate	Integer	Baud rate (e.g. 9600, 56600,..)
dataBits	Integer	Data bits
parity	System.IO.Ports.Parity	Parity (e.g. None, Odd,..)
stopBits	System.IO.Ports.StopBits	Stop bits (e.g. 1, 1.5, 2)

Sample Use:

```
[Visual Basic]
pneOutput.Kind = PocketNiceEngine.OutputKindType.Com
pneOutput.SetCom(2, 9600, 8, System.IO.Ports.Parity.None, _
    System.IO.Ports.StopBits.None)
```

```
[C#]
pneOutput.Kind = PocketNiceEngine.OutputKindType.Com;
pneOutput.SetCom(2, 9600, 8, System.IO.Ports.Parity.None,
    System.IO.Ports.StopBits.None);
```

SetFile

Description:

Print to file parameters

Signature:

Sub **SetFile**(ByVal *file* As String)

Parameters:

Name	Data Type	Description
file	String	Output file name (full path)

SetIrda

(SetIrda is not implemented in current version of software)

SetJobFile

Description:

Sets location for writing a JOB file to a network-accessible directory.

Signature:

Sub **SetJobFile**(ByVal *file* As String)

Parameters:

Name	Data Type	Description
file	String	Output JOB file path

Sample Use:

[Visual Basic]

```
pneOutput.Kind = PocketNiceEngine.OutputKindType.Job
pneOutput.SetJobFile("My Device\My Documents\Sample.job")
```

[C#]

```
pneOutput.Kind = PocketNiceEngine.OutputKindType.Job;
pneOutput.SetJobFile("My Device\\My Documents\\Sample.job");
```

SetJobTcpIp

Description:

Set parameters for sending a JOB file to NiceWatch via TCP/IP.

Signature:

Sub **SetJobTcpIp**(ByVal *address* As String, ByVal *port* As Long, ByVal *validationMsg* As String)

Parameters:

Name	Data Type	Description
address	String	IP address
port	Long	TCP port
validationMessage	String	Security future: on connect NiceWatch send's Validation message.

Sample Use:

[Visual Basic]

```
pneOutput.Kind = PocketNiceEngine.OutputKindType.JobTcpIP
pneOutput.SetTcpIp("192.168.0.100", 6108, "Custom Message")
```

[C#]

```
pneOutput.Kind = PocketNiceEngine.OutputKindType.JobTcpIp;
pneOutput.SetJobTcpIp("192.168.0.100", 6108, "Custom Message");
```

SetTcpIp

Description:

Sets TCP/IP (WIFI) printing parameters.

Signature:

Sub **SetTcpIp**(ByVal *address* As String, ByVal *port* As Long)

Parameters:

Name	Data Type	Description
address	String	IP address
port	Long	TCP port

Sample Use:

```
[Visual Basic]
pneOutput.Kind = PocketNiceEngine.OutputKindType.TcpIP
pneOutput.SetTcpIp("192.168.0.100", 6108)
```

```
[C#]
pneOutput.Kind = PocketNiceEngine.OutputKindType.TcpIP;
pneOutput.SetTcpIp("192.168.0.100", 6108);
```

Send

Description:

Send data to output device

Signature:

Sub **Send**(ByVal *data*() As Byte, ByVal *len* As Long)

Parameters:

Name	Data Type	Description
data	Byte()	Data to be send to the connected device
len	Long	Data length in bytes

Note: The Open, Send, and Close methods are all handled internally by the ILabel.Print method and typically are not used externally.

Properties:

The following properties are only applicable when the current IOutput.Kind supports that parameter. E.G. Address is only applicable to TcpIp and JobTcpIp OutPutTypes.

Address

Description:

Returns the IP Address.

Signature:

ReadOnly Property **Address**() As String

Sample Use:

```
[Visual Basic]
Dim PortAddress As String = pneOutput.Address
```

```
[C#]  
string PortAddress = pneOutput.Address;
```

BaudRate

Description:

Returns baud rate.

Signature:

ReadOnly Property **BaudRate()** As Integer

Sample Use:

```
[Visual Basic]  
Dim PortBaud As Int32 = pneOutput.BaudRate
```

```
[C#]  
Int32 PortBaud = pneOutput.BaudRate;
```

DataBits

Description:

Returns data bits.

Signature:

ReadOnly Property **DataBits()** As Integer

Sample Use:

```
[Visual Basic]  
Dim PortDataBits As Int32 = pneOutput.DataBits
```

```
[C#]  
Int32 PortDataBits = pneOutput.DataBits;
```

Kind

Description:

Sets or returns type of output port (e.g. Bluetooth, COM, TCP/IP, etc...)

Signature:

Property **Kind()** As **PocketNiceEngine.OutputKindType**

Sample Use:

```
[Visual Basic]  
Dim PortKind As PocketNiceEngine.OutputKindType = pneOutput.Kind
```

```
[C#]  
PocketNiceEngine.OutputKindType PortKind = pneOutput.Kind;
```

Parity

Description:

Returns Parity

Signature:

ReadOnly Property **Parity()** As **System.IO.Ports.Parity**

Sample Use:

```
[Visual Basic]  
Dim PortParity As System.IO.Ports.Parity = pneOutput.Parity
```

```
[C#]  
System.IO.Ports.Parity PortParity = pneOutput.Parity;
```

Port

Description:

Returns Port number.

Signature:

ReadOnly Property **Port()** As Long

Sample Use:

```
[Visual Basic]  
Dim PortNum As Int64 = pneOutput.Port
```

```
[C#]  
Int64 PortNum = pneOutput.Port;
```

PrintFile

Description:

Returns PrintFile path.

Signature:

ReadOnly Property **PrintFile()** As String

Sample Use:

```
[Visual Basic]  
Dim PrintFile As String = pneOutput.PrintFile
```

```
[C#]  
string PrintFile = pneOutput.PrintFile;
```

StopBits

Description:

Returns Stop Bits.

Signature:

ReadOnly Property **StopBits()** As System.IO.Ports.StopBits

Sample Use:

```
[Visual Basic]  
Dim PortStopBits As System.IO.Ports.StopBits = pneOutput.StopBits
```

```
[C#]  
System.IO.Ports.StopBits PortStopBits = pneOutput.StopBits;
```

Validation Message

Description:

Returns Validation Message

Signature:

ReadOnly Property **ValidationMessage()** As String

Sample Use:

```
[Visual Basic]
Dim PortValidMsg As String = pneOutput.ValidationMessage

[C#]
string PortValidMsg = pneOutput.ValidationMessage;
```

3.1.5 IVariable Interface

Properties:

DataFormat

Description:

Sets or returns variable's data format.

Signature:

Property **DataFormat()** As **PocketNiceEngine.IFormat**

Sample Use:

```
[Visual Basic]
Dim VarDataFormat As PocketNiceEngine.IFormat
VarDataFormat = pneVariable.DataFormat

[C#]
PocketNiceEngine.IFormat VarDataFormat = pneVariable.DataFormat;
```

DefaultValue

Description:

Sets or returns variable's default value (can also be set when designing the label).

Signature:

Property **DefaultValue()** As String

Sample Use:

```
[Visual Basic]
Dim VarDefaultVal As String = pneVariable.DefaultValue

[C#]
string VarDefaultVal = pneVariable.DefaultValue;
```

Id

Description:

Sets or returns the variable's ID.

Signature:

Property **Id()** As Integer

Sample Use:

```
[Visual Basic]  
Dim VarID As Int32 = pneVariable.Id
```

```
[C#]  
Int32 VarID = pneVariable.Id;
```

InputKind

Description:

Returns variable's input type (e.g. Prompt, Database, Generated)

Signature:

ReadOnly Property **InputKind()** As **PocketNiceEngine.InputType**

Sample Use:

```
[Visual Basic]  
Dim itKind As PocketNiceEngine.InputType = pneVariable.InputKind
```

```
[C#]  
PocketNiceEngine.InputType itKind = pneVariable.InputKind;
```

IsFixedLength

Description:

Sets or returns if this variable requires a fixed length value

Signature:

Property **IsFixedLength()** As Boolean

Sample Use:

```
[Visual Basic]  
Dim IsVarFixed As Boolean = pneVariable.IsFixedLength
```

```
[C#]  
bool IsVarFixed = pneVariable.IsFixedLength;
```

IsQuantity

Description:

Sets or returns if this variable's value is used as the quantity of labels to print.

Signature:

Property **IsQuantity()** As Boolean

Sample Use:

```
[Visual Basic]  
Dim IsVarQuant As Boolean = pneVariable.IsQuantity
```

```
[C#]  
bool IsVarQuant = pneVariable.IsQuantity;
```

IsUsed

Description:

Returns where or not this variable is used on the label.

Signature:

ReadOnly Property **IsUsed**() As Boolean

Sample Use:

```
[Visual Basic]
Dim IsVarUsed As Boolean = pneVariable.IsUsed

[C#]
bool IsVarUsed = pneVariable.IsUsed;
```

Length

Description:

Set or returns the maximum number of characters this variable can hold.

Signature:

Property **Length**() As Integer

Sample Use:

```
[Visual Basic]
Dim VarLen As Int32 = pneVariable.Length

[C#]
Int32 VarLen = pneVariable.Length;
```

Name

Description:

Sets or returns the name of the variable.

Signature:

Property **Name**() As String

Sample Use:

```
[Visual Basic]
Dim VarName As String = pneVariable.Name

[C#]
string VarName = pneVariable.Name;
```

Prefix

Description:

Sets or returns a static value displayed immediately before the variable's value.

Signature:

Property **Prefix**() As String

Sample Use:

```
[Visual Basic]
Dim Prefix As String = pneVariable.Prefix
```

```
[C#]  
string Prefix = pneVariable.Prefix;
```

Suffix

Description:

Sets or returns a static value displayed immediately after the variable's value.

Signature:

Property **Suffix()** As String

Sample Use:

```
[Visual Basic]  
Dim Suffix As String = pneVariable.Suffix
```

```
[C#]  
string Suffix = pneVariable.Suffix;
```

Value

Description:

Sets or returns the current value to which the variable is set.

Signature:

Property **Value()** As String

Sample Use:

```
[Visual Basic]  
Dim VarValue As String = pneVariable.Value
```

```
[C#]  
string VarValue = pneVariable.Value;
```

Events:

OnContentsChange

Description:

This event is triggered every time a variable value is updated.

Signature:

Event **OnContentsChange**(ByVal sender As Object, ByVal e As **PocketNiceEngine.VariableEventArgs**)

3.1.6 IPromptVariable Interface

IPromptVariable extends IVariable with two additional properties.

Properties:

Prompt

Description:

Returns variable's prompt (message to user indicating what the variable will be used for)

Signature:

Property **Prompt()** As String

Sample Use:

```

[Visual Basic]
Dim pneVar As PocketNiceEngine.IPromptVariable
For Each pneVar In pneVariables
    pneVar.Value = InputBox(pneVar.Prompt, pneVar.Name, _
        pneVar.Default Value)
Next

[C#]
foreach (PocketNiceEngine.IPromptVariable pneVar in pneVariables)
{
    pneVar.Value = Microsoft.VisualBasic.Interaction.InputBox( _
        pneVar.Prompt, pneVar.Name, pneVar.DefaultValue, 200, 200);
}

```

3.1.7 IDateTimeVariable Interface

This interface is currently functionally equal to the IVariable interface. Additional functionality will be added in future releases of the software.

3.1.8 ILabelVariables Interface

Methods:

GetAt

Description:

Gets variable at position. If variable at the selected position exists then its reference is returned, if not null is returned.

Signature:

Function **GetAt**(ByVal *index* As Integer) As **PocketNiceEngine.IVariable**

Parameters:

Name	Data Type	Description
index	Integer	Variable's position in the list

Sample Use:

```

[Visual Basic]
pneVariable = pneLabel.LabelVariables.GetAt(value)

[C#]
pneVariable = pneLabel.LabelVariables.GetAt(value);

```

Properties:

Item (+1 overloads)

Description:

ILabelVariable's default property which sets or returns an IVariable object from the collection by ID.

Signature:

Default Property **Item**(ByVal *id* As Integer) As **PocketNiceEngine.IVariable**

Default Property **Item**(ByVal *name* As String) As **PocketNiceEngine.IVariable**

Sample Use:

[Visual Basic]
pneVariable = pneVariables.Item(1)

[C#]
pneVariable = pneVariables.Item[1];

Parameters:

Name	Data Type	Description
id	Integer	Variable's unique id
name	String	Variable's name

VariableQuantity

Description:

Returns the IVariable object that is set as the label's print quantity. Returns null if the label was not designed with a variable to be used as the print quantity.

Signature:

ReadOnly Property **VariableQuantity**() As **PocketNiceEngine.IVariable**

Sample Use:

[Visual Basic]
pneVariable = pneVariables.VariableQuantity

[C#]
pneVariable = pneVariables.VariableQuantity;

3.1.9 IFormat Interface

Used for verifying variable data. Data is controlled by 256 flags (every flag is representing 1 ASCII character) in a form 32 Bytes x 8 flags.

Methods:

IsDataOk (+1 overloads)

Description:

Verify if data is compatible with this IFormat (2 overloaded methods)

Signature:

Function **IsDataOk**(ByVal *data*() As Byte) As Boolean

Function **IsDataOk**(ByVal *str* As String) As Boolean

Parameters:

Name	Data Type	Description
data	Byte()	Data to be verified
str	String	Text (data) to be verified

Set

Description:

Set control bytes for custom format

Signature:

Sub **Set**(ByVal *flags*() As Byte)

Parameters:

Name	Data Type	Description
flags	Byte()	Format control flags (32 x 8 bits 256 flags (one for every character in ASCII table))

3.1.10 DataKindType Enum

Const	Value	Meaning
None	0	Custom production kind (cannot be used because IProduction interface is currently not exposed)
Lvx	1	Generate print stream

3.1.11 FormatKind Enum

Const	Value	Meaning
All	0	Format supports All ASCII characters
Numeric	1	Only numeric (0-9) characters
AlphaNumeric	2	Only alphanumeric characters
Letters	3	Only characters from capital A to small z
SevenBit	4	ASCII characters from 1 to 127
Hex	5	Only hex decimal characters are supported by this format
Date	6	Date format
Time	7	Time format
DigCaps	8	Digitals + Capital letters
Custom	9	Custom format (user has set format specifying flags)

3.1.12 InputType Enum

Const	Value	Meaning
Unknown	0	General input type (used by assembly internally)
Prompt	1	Variable value should be set before print by an external source
Generated	2	Mostly output value from functions (used by assembly internally)
Database	3	Currently not supported
ContentsProvider	4	Currently not supported

Note: Usually only Prompt variables are used outside of the assembly.

3.1.13 OutputKindType Enum

Const	Value	Meaning
None	0	No output device selected
TcpIP	1	TCP/IP (WiFi)
File	2	Print to file
BlueTooth	3	Print via BlueTooth (virtual COM port)
IrDA	4	Unsupported
Com	5	Com port
Job	6	Distributed printing – drop file
JobTcpIp	7	Distributed printing via TCP/IP

Note: **None** Outputkind type is set by Engine assembly internally when new IOutput interface is created and marks that output device is not set.

3.1.14 ProductionKind

Const	Value	Meaning
Custom	0	Custom production kind (cannot be used because IProduction interface is currently not exposed)
Xml	1	Generate print stream
Job	2	Generate JOB file (used in combination with Nice Watch) for distributed printing

3.1.15 PromptType Enum

Const	Value	Meaning
Start	0	At start of printing
Every	1	For every label printed
BOQ	2	Based on variable quantity
No Prompt	3	User is not prompted for value

3.1.16 StepKind Enum

Const	Value
Integer	0
Date	1
Time	2

4 ErrorHandler Class Library

4.1 ErrorHandler Namespace

4.1.1 Exception Class

Exception is a custom class designed to handle exceptions thrown by PocketNiceEngine. This class is inherited from System.Exception and the extended properties are shown below.

Properties:

ErrorID

Description:

Returns the identifying number of the error thrown. See ErrorID Enum for possible values.

Signature:

Public ReadOnly Property **ErrorId**() As Integer

Sample Use

```
[Visual Basic]
Catch pneEx As ErrorHandler.Exception
    MessageBox.Show("PNE Error: " & pneEx.ErrorId)

[C#]
catch (ErrorHandler.Exception pneEx)
{
    MessageBox.Show("PNE Error: " + pneEx.ErrorId);
}
```

Message

Description:

Returns the description of the error thrown.

Signature:

Public ReadOnly Property **Message**() As String

Sample Use

```
[Visual Basic]
Catch pneEx As ErrorHandler.Exception
    MessageBox.Show("PNE Error: " & pneEx.Message)

[C#]
catch (ErrorHandler.Exception pneEx)
{
    MessageBox.Show("PNE Error: " + pneEx.Message);
}
```

4.1.2 ErrorID Enum

Const	Value	Description
ACTION_FAILED_TO_EXECUTE	10002	Action [name] has failed to execute.
APPLICATION_IDENTIFIER_GROUP_HAS_FIXED_LENGTH	1034	[appidname] has fixed length of [formatlength].
APPLICATION_IDENTIFIER_GROUP_HAS_LENGTH	1035	[appidname] has length of [length].
APPLICATION_IDENTIFIER_HAS_FIXED_LENGTH	1031	[appidname] or one of it's parts has fixed length of [formatlength].
APPLICATION_IDENTIFIER_HAS_FORMAT_ALPHA	1033	[appidname] or one of it's parts dos not accept digits, value [value] is invalid for this application identifier.
APPLICATION_IDENTIFIER_HAS_NUMERIC_FORMAT	1032	[appidname] or one of it's parts has numeric format, value [value] is invalid for this application identifier.
CANNOT_CONNECT_TO_THE_OUTPUT_DEVICE	1024	Cannot connect to the output device.
CANNOT_FIND_ITEM	10004	Cannot find item [item] in the current database.
CANNOT_LOAD_FORM	10006	Cannot load form. Please check the form settings.
CANNOT_LOAD_LABEL	1006	Cannot load label print data.
CANNOT_PREPROCESS_LABEL_DATA	1021	Cannot preprocess label data.
CONNECTION_NOT_OPENED	1017	You must open connection before calling Send method.
DATA_FORMAT_NOT_SET	1011	Data format is not set.
DEMO_FUNCTIONALITY	1036	Demo functionality, software is limited to max. one variable.
DIVISION_BY_ZERO	1026	Cannot divide by zero.
FORM_ACTION_FAILED_TO_EXECUTE	10003	Form action has failed to execute Please check the action parameters.
HOOK_EXTENSION_PROCESSING_FAIL	1020	Cannot process hook extension.
INCOMPATIBLE_PRODUCTION	1003	Production class not compatible with used data class.
INPUT_VARIABLE_DOES_NOT_EXIST	1025	Input variable does not exist.

INVALID_BARCODE_DATA	1015	Data [value] is invalid for this barcode type.
INVALID_CD_APPID	1037	Invalid check digit for [appid] value.
INVALID_FORMAT_DESCRIPTION_LENGTH	1010	Format description length is invalid.
INVALID_START_HOOK	1004	Start page Hook is invalid.
INVALID_SUBSET_LENGTH	1028	Subset length cannot be longer than its offset.
INVALID_VAR_DATA_FORMAT	1007	Data is invalid for current variable format.
INVALID_VARIABLE_DATA_LENGTH	1016	Data is invalid for current variable format.
INVALID_VARIABLE_LENGTH	10001	Value of variable [name] should be [length] characters long!
INVALID_VARIABLE_PICTURE	1029	Variable [name] has picture [picture], value [value] is invalid for this picture.
INVALID_VALUE_LINEAR	1027	Only numerical values are allowed in linear function.
LABEL_PRINT_FAIL	1005	Unable to print label.
OPEN_LABEL_FAIL	1014	Cannot open label [name].
OPERATOR_PROCESSING_FAIL	1019	Processing operator [name] failed.
OUTPUT_VARIABLE_DOES_NOT_EXIST	1030	Output variable does not exist.
REGISTRATION_CODE_INVALID	10005	Activation code is not valid.
SYSTEM_ERROR	1002	System error.
UNABLE_TO_SEND_DATA	1018	Unable to send data on to the output device.
UNKNOWN_ERROR	1001	Unknown error
UNKNOWN_HOOK_EXTENSION_KIND	1012	Unknown hook extension kind [name].
UNKNOWN_HOOK_KIND	1013	
VALIDATION_MESSAGE_NOT_VALID	1023	Validation message [msg] differs from the one received.
VAR_FIXED_LENGTH	1022	Value of variable [name] should be [length] characters long.
VAR_ID_NOT_FOUND	1008	Variable with id=[id] not found in label variable list.

VAR_NAME_NOT_FOUND	1009	Variable with name=[name] not found in label variable list.
VARIABLE_NOT_SET	10000	Variable [name] is not initialized yet!

5 Appendix

Euro Plus d.o.o. and Niceware International, LLC

Euro Plus d.o.o. and Niceware International, LLC develop, supply and support software for automatic identification and data collection (AIDC) solutions on the desktop PC, the corporate server or the mobile enterprise environment. Our flagship product NiceLabel has become one of the world's major label design and printing software combining easy-to-use interfaces with the integration of advanced thermal transfer technology, ERP systems solutions, RFID technology and data collection tools. NiceLabel cooperates with printer manufacturers, partners and customers from all over the world.

Microsoft has certified all NiceLabel products with the "Designed for Windows XP and 2000" and "Designed for Windows Vista" logos, indicating reliability and operational compliance in the latest Windows environments. As a Microsoft Certified Partner, Niceware and Euro Plus present an excellent business opportunity for all those searching for a reliable, high-tech and advanced partner in the automatic identification and data collection industry.

NiceLabel Product Overview

NiceLabel is the most advanced professional labeling software for desktop, mobile and enterprise users. NiceLabel offers an easy-to-use interface and meets any label design and printing requirement for efficient label printing solutions to users in retail, logistics, healthcare, chemical, automotive and other industries. NiceLabel offers three main product series.

NiceLabel Standard Series

The Standard Series is a line of NiceLabel core products for bar code and RFID label design, printing and entry-level integration. Products are easy to use but offer powerful functions expected from advanced label designers. The Standard Series includes NiceLabel Suite, NiceLabel Pro, NiceLabel Express and Pocket NiceLabel.

NiceLabel Suite: The full-featured, modular labeling solution for label design, integration and professional printing requirements. Multiple connectivity options allow users to perform stand-alone printing or integrate label printing into any network environment. NiceForm is a module in NiceLabel Suite that allows creating data entry and printing applications to make label production simple and error free. NiceLabel Suite includes NiceLabel Pro, NiceWatch, NiceForm, NicePrint, NicePrintQueue and Pocket NiceLabel for Windows CE support to create mobile printing applications.

NiceLabel Pro: The advanced label designer for professional bar code and RFID labeling, including complete database support and integration options. A wide range of features and options make NiceLabel Pro a perfect and easy-to-use tool for any labeling requirement. NiceLabel Pro includes NicePrintQueue, NiceData and NiceMemMaster.

NiceLabel Express: The basic label designer using wizards to fulfill basic bar code labeling needs. This entry-level software includes many design elements from the Pro edition with the emphasis on simplified user interaction.

Pocket NiceLabel: The label printing software for Windows CE mobile devices. Pocket NiceLabel enables Windows CE compatible computers and terminals to print bar code and RFID smart labels on any type of thermal printer that is supported by NiceLabel printer drivers.

NiceLabel Enterprise Series

The Enterprise Series is a line of NiceLabel products designed for centralized printing systems management, monitoring and integrated high-volume printing. The Enterprise Series includes NiceLabel Print Center and NiceWatch Enterprise.

NiceLabel Print Center: The enterprise solution for client-based label printing and centralized systems management. NiceLabel Print Center product includes two modules. The NiceLabel Enterprise Print Manager module manages the printing process centrally while label design and printing occurs locally on the client computers hosting the NiceLabel Suite module.

NiceWatch Enterprise: The enterprise solution for centralized integration of multi-threaded and high-volume label printing. NiceWatch Enterprise integrates the label printing process into enterprise-level products, such as Enterprise Resource Planning (ERP) systems, Warehouse Management Systems (WMS), Hospital Information Systems (HIS), and others.

NiceLabel Developer Series

The Developer Series is a line of NiceLabel products designed for software publishers looking for a way to integrate label printing functionality into their own applications. The Developer Series includes NiceLabel SDK, NiceLabel WebSDK and NiceLabel Pocket SDK.

NiceLabel SDK: Enables software publishers to reduce label printing development costs and add additional value to their Windows applications. NiceLabel SDK can be embedded into existing applications or information systems to support label printing.

NiceLabel WebSDK: Similar to NiceLabel SDK, the NiceLabel WebSDK offers software publishers to include bar code and RFID smart label printing in their Web applications. The NiceLabel WebSDK enables end-users to print labels without installing the NiceLabel software on client computers.

NiceLabel PocketSDK: Enables software publishers to integrate bar code and RFID smart label printing in Windows CE mobile applications. Software publishers deploy one application interface to print labels from a mobile device to more than 1300 thermal printers.

6 Online Support

You can find the latest builds, updates, workarounds for problems and Frequently Asked Questions (FAQ) under the Support section on our Web site at www.nicelabel.com. If you cannot solve the problem on your own, please contact your local vendor or representative offices listed in the topic Contact Information.

For more information please refer to:

- Support FAQ: <http://www.nicelabel.com/Support/FAQ>
- NiceLabel Tutorials: <http://www.nicelabel.com/Learning-center/Tutorials>
- NiceLabel Forums: <http://forums.nicelabel.com/>

7 Contact Information

Head Office

Euro Plus d.o.o.

Poslovna cona A 2
SI-4208 Šenčur, Slovenia
Tel: +386 4 280 50 00
Fax: +386 4 233 11 48
www.europlus.si
info@europlus.si
sales@europlus.si
support@europlus.si

German Office

NiceLabel Germany GmbH
Liebknechtstr. 29
63179 Obertshausen
Germany
+49 (0)6104 405 400 Tel
+49 (0)6104 405 4020 Fax
info@nicelabel.de
www.nicelabel.de

North American Office

Niceware International, LLC
200 South Executive Drive, Suite
200
Brookfield, Wisconsin 53005
USA
Telephone: (888) 894-NICE (6423)
Fax: (262) 784-2495
E-mail: sales@nicewareintl.com
support@nicewareintl.com
Web: www.nicewareintl.com