

IND780

Terminal

.NET Custom

Applications Guide

© METTLER TOLEDO 2012

No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the express written permission of METTLER TOLEDO.

U.S. Government Restricted Rights: This documentation is furnished with Restricted Rights.

Copyright 2012 METTLER TOLEDO. This documentation contains proprietary information of METTLER TOLEDO. It may not be copied in whole or in part without the express written consent of METTLER TOLEDO.

METTLER TOLEDO reserves the right to make refinements or changes to the product or manual without notice.

COPYRIGHT

METTLER TOLEDO® is a registered trademark of Mettler-Toledo, LLC. All other brand or product names are trademarks or registered trademarks of their respective companies.

**METTLER TOLEDO RESERVES THE RIGHT TO MAKE REFINEMENTS OR
CHANGES WITHOUT NOTICE.**

Contents

Introduction	7
Installing the IND780 SDK.....	7
Installing on a Windows XP PC	7
Installing on a Windows 7 PC	8
Creating an IND780 C# Custom Application	8
The IND780 .NET Custom Application Toolkit.....	10
Using the Shared Data Client Class.....	10
What is the Shared Data Client?.....	10
Shared Data Variables	10
Shared Data Server	10
Shared Data Server Dedicated Characters	10
Referencing the Shared Data Client Class.....	11
Shared Data Client Class Methods	12
RegisterCallback	13
UnregisterCallback	14
UnregisterCallbacks.....	14
RegisterCallbackGroup	15
UnregisterCallbackGroup.....	16
UnregisterCallbackGroups	16
ReadSharedData	17
ReadSharedDataList	17
SetSharedData	17
SetSharedDataList	18
GetSharedDataType.....	18
GetSharedDataLength	19
Ctimer	20
Shared Data Client Class Table Methods	20
OpenTables.....	22
CloseTables	22
ParseRow	22
GetFirstStandardRow	23
GetFirstStandardRow	24
GetFirstStandardRow	24
GetNextStandardRow.....	25
InsertStandardRow	25
EditStandardRow	26
EditStandardItem.....	26
DeleteStandardRow	27
DeleteStandardRowEntryID	27
DeleteTable	27
Shared Data Client Class Metrology Methods	28
Clear	28

TarePB	28
TarePreset.....	29
Zero	30
SelectScale.....	31
Print.....	31
DisplayWt.....	32
StartSetup	34
KillSetup	35
Command	35
CommandCB	36
Using the Softkeys Class	37
Definition of Softkeys	38
Definition of Application and Scale Keys	39
Softkeys Methods	39
Softkeys (constructor)	39
AllocateSoftkeys.....	40
DrawSoftkeyBorders	40
LoadSoftkeys.....	41
UpdateSoftkey.....	42
DisplaySoftkeys	42
EnableSoftkey.....	42
DisplayNextSoftkeyLevel	43
SoftkeyDispatcher	44
NumberOfLevels	45
SoftkeyLevel	45
SoftkeyLevelDisplay.....	46
Application and Scale Keys Methods.....	46
LoadApplicationAndScaleKeys	46
ApplicationAndScaleKeysDispatcher	48
Using the Information Class	48
What the Information Class Includes.....	49
Recall Information Form.....	49
Weight Recall Form.....	49
System Information Recall Form	49
Metrology Recall Form.....	50
Adding the Information Softkey ⓘ	50
Registering the Softkey.....	50
Using the Information Softkey ⓘ	51
Information Class Screen Shots	52
Example Main Form with Information Softkey ⓘ	52
Recall Information Form ⓘ	52
Weight Recall Form ⓘ	53
System Information Recall Form ⓘ	53
Metrology Recall Form M	54

Using Serial I/O	54
Custom Application Restrictions	55
Legal-For-Trade	55
Not Legal-For-Trade	55
Custom Application Configuration	56
File Name	57
Automatic Start	57
Manual Start	57
Reserve Console for Application	58
Checksum Testing	58
Setup Tree Recovery	58
SQL Database	58
Debugging	59
To Prepare the IND780 for Connecting	59
To Prepare Visual Studio for Connecting	59
To Set Security and Establish the Connection	59
Revision History	60

Introduction

This .NET Custom Applications enhancement to the IND780 allows users to write custom applications within the .NET Compact Framework environment, thus is a more open and more powerful alternative to TaskExpert.

- Custom applications must implement .NET Compact Framework 2.0. This is supported by Visual Studio 2005 or 2008. Visual Studio 2010 and newer do not support .NET Compact Framework.

A set of DLLs is available in the IND780 .NET Custom Application Toolkit to allow access to some of the functionality of the IND780.

Care must be taken when writing software to run on the IND780 platform. The IND780 real time processing must be the highest priority process. The user's application must be event driven and be a lower priority so as not to interfere with the normal scale terminal processing.

The user **must install the IND780 SDK** and is encouraged to install the IND780 .NET Custom Application Toolkit. Each is covered in the following sections.

Installing the IND780 SDK

The custom application programmer must install the IND780 SDK to create programs that will run on the IND780 terminal. The installation file **Excal270_SDK.msi** is included on the IND780 .NET Custom Application Toolkit CD. This SDK supports the PXA270 (ARMV4I core) processor. Running **Excal270_SDK.msi** from the CD will extract the needed files to the **C:\Program Files\Windows CE Tools\wce500\Excal270** folder.

Notes

- Custom applications are only supported by IND780 software versions 7.3.xx.
- The user must also install the hardware security key that is included with the IND780 .NET Custom Application Toolkit.
- If the file **Excal270_SDK.msi** has to be installed manually, follow the procedures outlined below, just replace the file name.

Installing on a Windows XP PC

1. Copy the file "CustomApplicationToolKitInstaller.msi" and setup.exe to the desired path.
2. Double click on the setup.exe file to start the installation and follow the instructions.

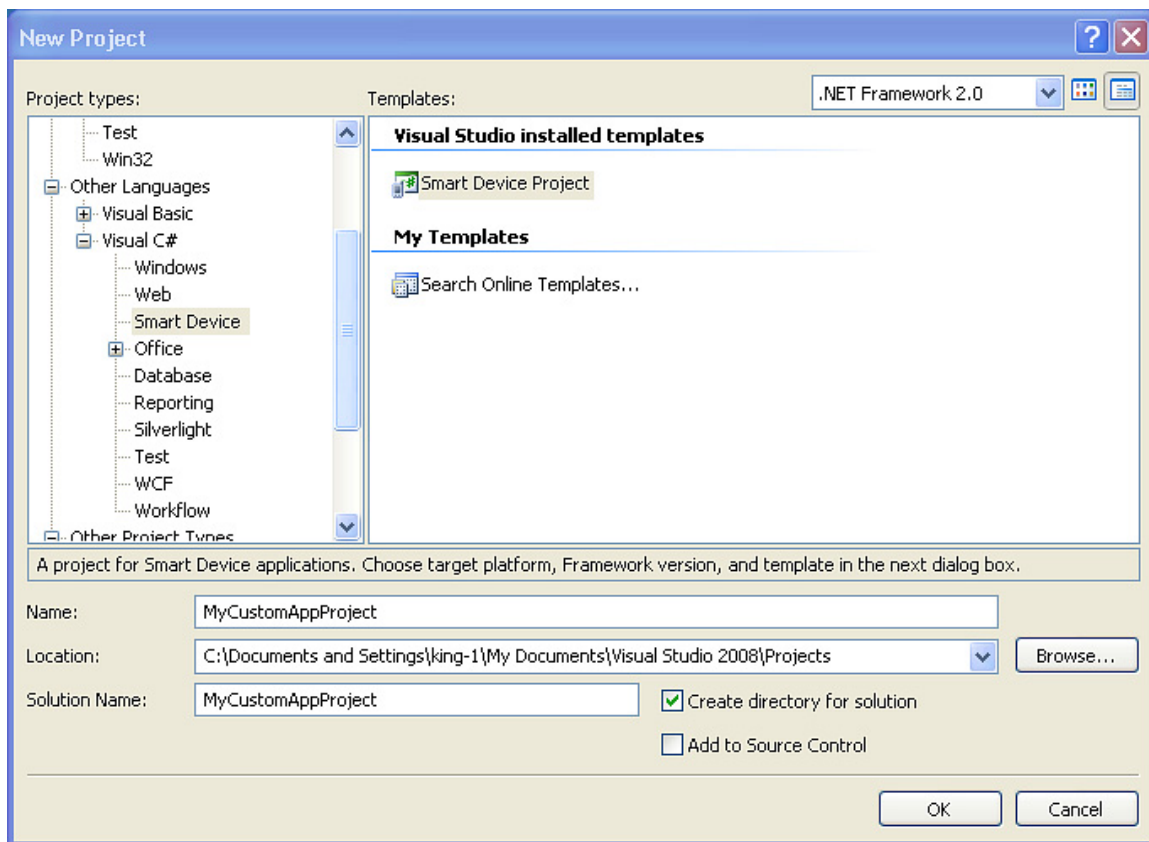
Installing on a Windows 7 PC

Installation must be done from the command prompt as follows:

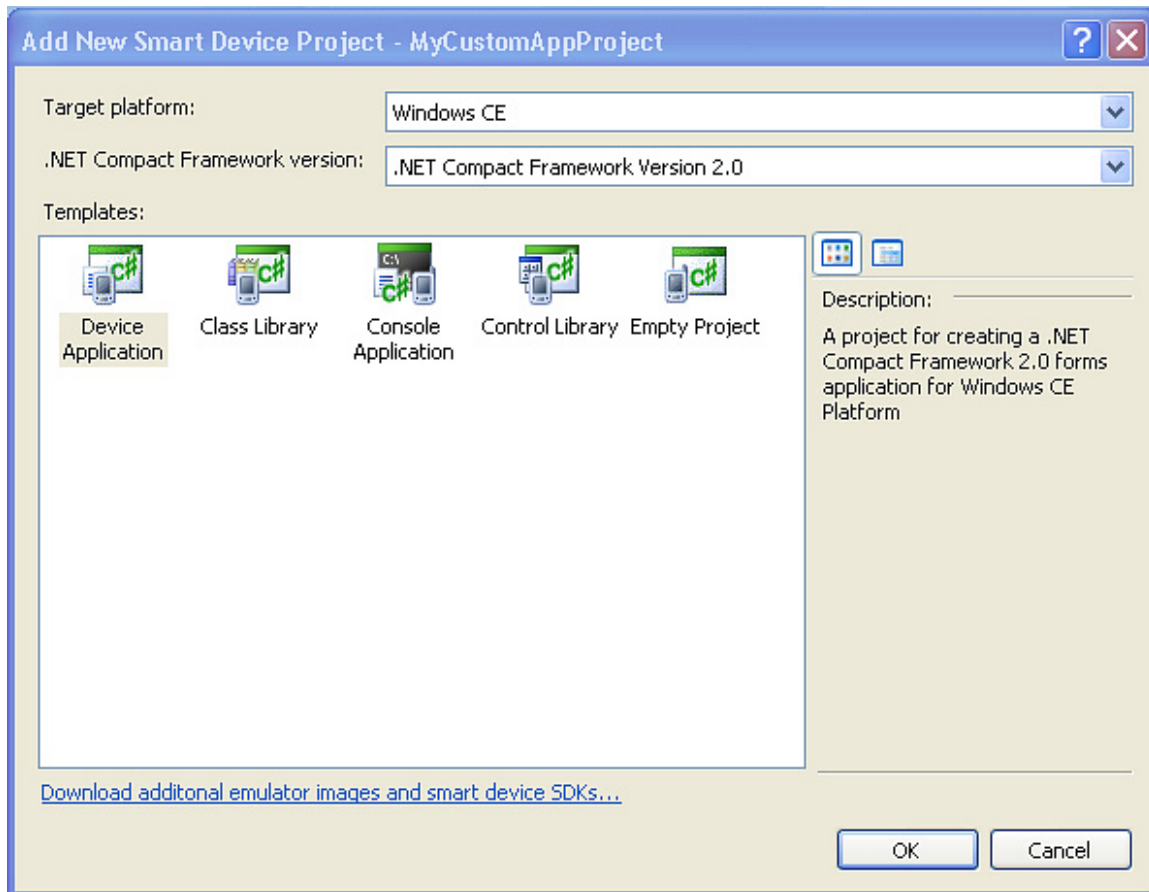
1. Copy the file "CustomApplicationToolKitInstaller.msi" to the desired path.
2. Click on the Windows 7 Start icon.
3. Type "CMD" in the "Search programs and files" box.
4. When "cmd.exe" appears, right click on it and select "Run as administrator".
5. The command prompt window will now be displayed. After the "C:\Windows\system32>" prompt type:
6. "msiexec.exe -i C:\your path\CustomApplicationToolKitInstaller.msi"
7. Follow the installation instructions.

Creating an IND780 C# Custom Application

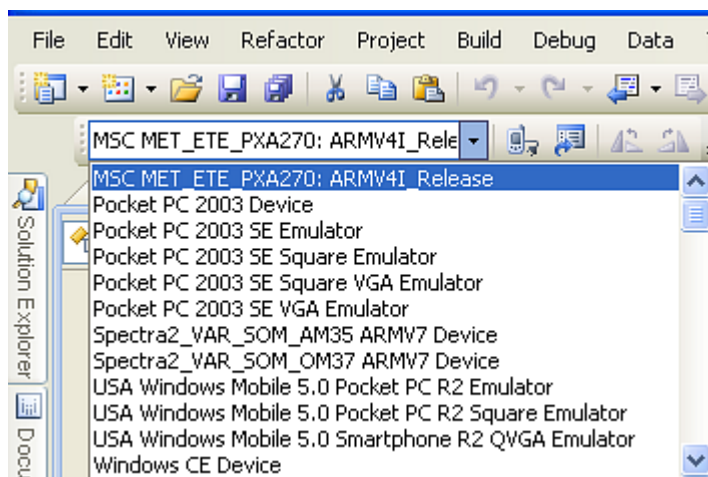
1. In Visual Studio 2008, select menu File/New/Project.
2. For Project type, select Visual C# / Smart Device.
3. Set Name and Location as desired and click OK. **Limit the name to 16 characters.**



4. A new window will open:



- For Target platform, select Windows CE.
 - For .NET Compact Framework version, select .NET Compact Framework Version 2.0 and Click OK.
5. A blank form will display. Select MSC MET_ETE_PXA270: ARMV4I_Release as the Target Device.



The IND780 .NET Custom Application Toolkit

The IND780 .NET Custom Application Toolkit includes three DLLs that supply classes for accessing IND780 internal functionality. Even though it is not necessary to implement these classes, it is highly recommended since it will save the programmer the time of duplicating the operation of these classes.

Each DLL contains a class that can be accessed by the custom application; these are the Shared Data Client Class, the Softkeys Class, and the Information Class. Each is explained in detail below.

The IND780 .NET Custom Application Toolkit CD contains the installation file **CustomApplicationToolKitInstaller.msi**. You can choose where to install the DLLs or use the default path: **C:\Program Files\Mettler Toledo\Custom Application Toolkit Installer**.

Using the Shared Data Client Class

What is the Shared Data Client?

The Shared Data Client is a DLL (Dynamic Link Library) that contains methods for reading and writing Shared Data variables stored on the IND780 terminal. Access to these variables is handled by the Shared Data Server.

Shared Data Variables

Shared Data Variables are variables that contain both configuration data as well as dynamic data about the IND780 terminal. These variables reside in Heap memory (volatile storage), Flash memory, BRAM memory, or EEPROM memory (located on the option boards). Please reference the IND780 Terminal Shared Data Reference manual 640591 10 for a list of Shared Data Variables.

Shared Data Server

The Shared Data Server resides on the IND780 and controls access to all Shared Data Variables. This document focuses on access to the Shared Data Server through a TCP/IP socket connection.

Shared Data Server Dedicated Characters

The Shared Data Server uses these characters as delimiters:

Carat ^

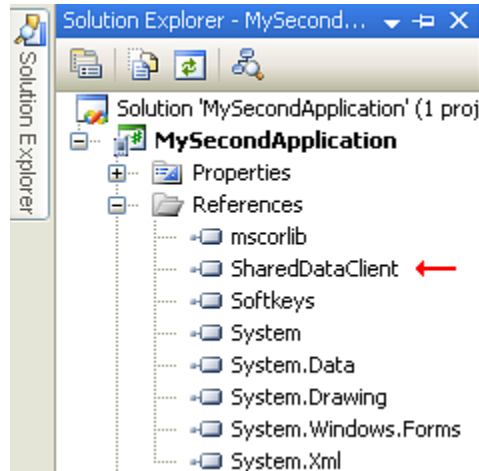
Tilde ~

Quotation "

For this reason, you should not include any of these characters in data being passed to the Shared Data Server except where noted.

Referencing the Shared Data Client Class

A reference to the SharedDataClient.dll must be added to the application as shown below. Adding the SharedDataClient.dll reference will require entering the path where this file resides on the PC. This will expose the SDClient class to the application.



Add the `using MettlerToledo.IND780.SharedDataClient` line to the application to simplify method calls. The user must create an instance of the SDClient class for each thread that requires shared data access.

```
using MettlerToledo.IND780.SharedDataClient;
namespace MyCustomApplication
{
    public partial class CustomForm : Form
    {
        public static SDclient SD_client;
        /*-----*
            * Name           :CustomForm::CustomForm
            * Description    :Constructor
            * Return Value   :None
        * -----*/
        public CustomForm()
        {
            SD_client = new SDclient();
            InitializeComponent();
            this.SharedDataServerConnect();
        }
    }
}
```

Shared Data Client Class Methods

ConnectToSharedDataServer

Format

```
int ConnectToSharedDataServer(string username, string password)
```

Description

Makes two connections to the IND780 Shared Data Server, one that handles simple reads and writes and one that handles callbacks.

By default the username is "admin" and there is no password. The user can change the password or create additional users in IND780 Setup.

It is recommended to make only one connection instance. The Shared Data Server can support multiple connections, however, these also include any external connections.

Returns

0 if successful, -1 if login failed, -2 if connection failed, -3 if Unicode mode failed.

Example

```
/*-----*
 * Name      :CustomForm::SharedDataServerConnect
 * Description :Connects to the IND780 Shared Data Server
               :Displays a green square in the upper left corner
               :if successful or a red square if not successful.
 * Return Value :True - Connected successfully
 *              :False - Failed to connect.
               :Check IND780 power and cables.
 * -----*/
private bool SharedDataServerConnect()
{
    int connectionStatus = SD_client.ConnectToSharedDataServer("admin", "");
    if (connectionStatus == 0)
    {
        this.label_connected.BackColor = Color.Lime;
        this.label_connected.Refresh();
        return true;
    }
    this.label_connected.BackColor = Color.Red;
    this.label_connected.Refresh();
    return false;
}
```

DisconnectFromSharedDataServer

Format

```
void DisconnectFromSharedDataServer()
```

Description

Disconnects the read / write and the callback connections.

Returns

Nothing.

Example

```
SD_client.DisconnectFromSharedDataServer();
```

RegisterCallback

Format

```
bool RegisterCallback(MyCallbackDelegate cb, string sdname)
```

Description

Registers the delegate callback method (cb) in the user's application for the specified shared data name (sdname). The callback method will be called every time the specified shared data variable's value changes.

The callback method must have the signature:

```
void methodName(string sdName, string sdValue)
```

Returns

True if callback registered, false if a bad shared data name or the callback did not register.

Example

```
using MettlerToledo.IND780.SharedDataClient;
namespace MyCustomApplication
{
    public partial class CustomForm : Form
    {
        public SDclient SD_client;
        /*-----*
        * Name           :CustomForm::CustomForm
        * Description    :Constructor
        * Return Value   :None
        * -----*/
        public CustomForm()
        {
            SD_client = new SDclient();
            InitializeComponent();
            this.SharedDataServerConnect();
            myForm = this;
            this.RegisterCallback();
        }
        /*-----*
        * Name           :CustomForm::RegisterCallback
        * Description    :Register the callback method.
        * Return Value   :None
        * -----*/
        private void RegisterCallback()
        {SD_client.RegisterCallback(
            (SDclient.MyCallbackDelegate)MyCallback, "di0501");
        }
        /*-----*
        * Name           :CustomForm::MyCallback
```

```
* Description :This is the callback method that invokes the
*              :DiscreteInput_1 method that resides in the *
*              :CustomForm thread.
* Return Value :None
* -----*/
public void MyCallback(string sdName, string sdValue)
{
    Invoke(new SDclient.myCallbackDelegate(DiscreteInput_1),
        new object[] { sdName, sdValue });
}
/*-----*
* Name       :CustomForm::DiscreteInput_1
* Description :Shows on or off status of the discrete I/O input 1
*              :on card slot 5
* Return Value :None
* -----*/
public void DiscreteInput_1(string sdname, string sdvalue)
{
    if (sdvalue == "1")
        this.label4.Text = "On";
    else
        this.label4.Text = "Off";
}
}
```

UnregisterCallback

Format

```
bool UnregisterCallback(string sdName)
```

Description

Removes (unregisters) a specific shared data callback.

Returns

True if the callback is unregistered, false if a bad shared data name or the callback did not unregister.

Example

```
SD_client.UnregisterCallback("di0501");
```

UnregisterCallbacks

Format

```
void UnregisterCallbacks()
```

Description

Removes (unregisters) all callbacks.

Returns

Nothing.

Example

```
SD_client.UnregisterCallbacks();
```

RegisterCallbackGroup

Format

```
bool RegisterCallbackGroup(MyCallbackGroupDelegate cg, int groupNumber, string sdNames)
```

Description

Registers the delegate group callback method (cb) assigned the group number (groupNumber) for the shared data name string (sdNames) that contains the shared data names in the group.

Shared data names are separated by spaces. Valid group numbers are 1 to 9 and the maximum number of shared data names is 12.

The group callback method must have the signature:

```
void methodName(int groupNumber, string sdNames)
```

Returns

True if group callback registered, false if a bad shared data name or the group callback did not register.

Example

```
using MettlerToledo.IND780.SharedDataClient;
namespace MyCustomApplication
{
    public partial class CustomForm : Form
    {
        public static SDclient SD_client;
        /*-----*
        * Name       :CustomForm::CustomForm
        * Description :Constructor
        * Return Value :None
        * -----*/
        public CustomForm()
        {
            SD_client = new SDclient();
            InitializeComponent();
            this.SharedDataServerConnect();
            myForm = this;
            this.RegisterCallback();
            this.RegisterWtGroupCallbacks();
        }
        /*-----*
        * Name       :CustomForm::RegisterWtGroupCallbacks
        * Description :Registers group callbacks for weight
        * Return Value :None
        * -----*/
        void RegisterWtGroupCallbacks()
        {
            SD_client.RegisterCallbackGroup(
                (SDclient.MyCallbackGroupDelegate) WtCallbackGroup, 1, "wt0102 wt0202");
        }
        /*-----*
        * Name       :CustomForm::WtCallbackGroup
        * Description :Invokes the delegate method
        * :GroupWeightUpdate
        * Return Value :None
        * -----*/
    }
}
```

```
void WtCallbackGroup(int groupNumber, string sdValues)
{
    try
    {
        Invoke(new SDclient.yCallbackGroupDelegate(GroupWeightUpdate), new
            object[] { groupNumber, sdValues });
    }
    catch
    {
    }
}

/*-----*
* Name      :CustomForm::GroupWeightUpdate
* Description :Updates the two scale weights section
* Return Value :None
* -----*/
private void GroupWeightUpdate(int groupNumber, string sdValues)
{
    string[] weights = sdValues.Split('~'); this.label_information.Text =
        weights[0] + " " + weights[1];
}
}
```

UnregisterCallbackGroup

Format

```
bool UnregisterCallbackGroup(int groupNo)
```

Description

Unregisters (removes) the specified callback group (groupNo).

Returns

True if successful, false if unregistering fails.

Example

```
SD_client.UnregisterCallbackGroup(2);
```

UnregisterCallbackGroups

Format

```
void UnregisterCallbackGroups()
```

Description

Unregisters all callback groups.

Returns

Nothing.

Example

```
SD_client.UnregisterCallbackGroups();
```

ReadSharedData

Format

```
string ReadSharedData(string sdname)
```

Description

Reads the specified shared data (sdname) from the IND780 Shared Data Server and returns its value.

Returns

The shared data value as a string.

Example

```
// read weight  
string wtValue = SD_client.ReadSharedData("wt0101");
```

ReadSharedDataList

Format

```
string ReadSharedDataList(string sdnameList)
```

Description

Reads a list of shared data (sdnameList) from the IND780 Shared Data Server. The shared data names in the list must be separated by spaces such as "WT0101 WT0201 WT0301"

Returns

A list of shared data values separated by a tilde (~) such as:
"100.00~203.25~1546.82"

Example

```
string wtValues = SD_client.ReadSharedData("wt0101 wt0201 wt0301");  
string[] wtArray = wtValues.Split('~');
```

SetSharedData

Format

```
bool SetSharedData(string sdname, string sdvalue)
```


Description

Writes the value in sdvalue to the specified shared data sdname.

Returns

True if successful or false if not successful.

Example

```
string scaleName = "Scale A";  
bool status = SD_client.SetSharedData("cs0103", scaleName);
```

SetSharedDataList

Format

```
bool SetSharedDataList(string sharedDataNameList,  
string sharedDataValueList)
```

Description

Writes values separated with a tilde ~ to the shared data list separated by a space.
Example: SetSharedDataList("bx0101 bx0102", "1234~5678")

Returns

True if successful or false if not successful.

Example

```
double freq = 3.4;  
string scaleName = "My Scale";  
bool minWeigh = true;  
byte minWeighByte = Convert.ToByte(minWeigh);  
string nameList = "cs0103 cs0129 cs0114";  
string valueList = scaleName + "~" +  
minWeighByte.ToString() + "~" + freq.ToString();  
bool status = SD_client.SetSharedDataList(nameList,  
valueList);
```

GetSharedDataType

Format

```
string GetSharedDataType(string sdname)
```

Description

Gets the shared data type, see Returns.

Returns

The shared data type:

"ABl xx"	- Array of Bools of length xx
"ABy xx"	- Array of Bytes of length xx
"AL xx"	- Array of Longs of length xx
"Bl"	- Bool
"By"	- Byte
"D"	- Double Precision Floating Point
"F"	- Single Precision Floating Point
"L"	- Long
"S xx"	- String of length xx
"US"	- Unsigned Short
"UL"	- Unsigned Long

■ Note: A null string is returned if an invalid shared data name is used.

Example

```
long lValue = 0;
unsigned long ulValue = 0;
string temp = SD_client.ReadSharedData("xd0157");
string type = SD_client.GetSharedDataType("xd0157");
if (type == "UL")
{
    ulValue = ulong.Parse(temp);
}
else
{
    lValue = long.Parse(temp);
}
```

GetSharedDataLength

Format

```
string GetSharedDataLength(string sdname)
```

Description

Gets the length in bytes that the shared data uses

Returns

Length in bytes or a null string if an invalid shared data name is used.

Example

```
string notes = "";
int length = int.Parse(SD_client.GetSharedDataLength("xd0153"));
if (length < 60)
    notes = SD_client.ReadSharedData("xd0153");
```

Ctimer

Format

```
bool Ctimer(int timeValue)
```

Description

Sets the shared data callback timer value. This adjusts how fast a shared data callback can occur. Valid values are 5 to 60000 (msec)

Returns

True if successful or false if the callback value is out of range.

Example

```
SD_client.Ctimer(25); // set callback time to 25 msec.
```

Shared Data Client Class Table Methods

The IND780 Standard Database Tables reside in a SQL CE database. These standard tables have the following physical characteristics:

- The file containing the tables resides in Compact Flash. This file is limited by the application to 100 MB. As records are inserted the size is checked.
- There are ten tables, A0 - A9 in \Storage Card\Terminal\standard.sdf.
- You can access the records by the record ID (GUID) or the short ID. The record ID is the primary key for each table. SQL CE automatically assigns the record ID (GUID) when you insert a new entry in the table. This way, SQL CE ensures that the primary key for each row is unique.
- Each table has twenty columns with the following column names and types.

Column Name	Type
recordID	GUID
shortID	NVARCHAR(16)
description	NVARCHAR(40)

Column Name	Type
data1	NVARCHAR(16)
data2	NVARCHAR(16)
data3	NVARCHAR(16)
data4	NVARCHAR(16)
data5	NVARCHAR(16)
data6	NVARCHAR(16)
data7	NVARCHAR(16)
data8	NVARCHAR(16)
data9	NVARCHAR(16)
data10	NVARCHAR(16)
data11	NVARCHAR(16)
data12	NVARCHAR(16)
data13	NVARCHAR(40)
data14	NVARCHAR(40)
data15	NVARCHAR(40)
data16	NVARCHAR(40)
data17	NVARCHAR(40)

An NVARCHAR is a variable-length Unicode string.

The IND780 uses table A1 as the Tare Table.

Column Name	Tare Field
recordID	Record ID (GUID)
shortID	ID
description	Description
data1	Tare
data2	Units
data3	Total Count
data4	Total Weight

The IND780 uses table A2 as the Target Table.

Column Name	Target Field
recordID	Record ID (GUID)
shortID	ID
description	Description
data1	Target
data2	Units
data3	Negative Target
data4	Positive Target
data5	Fine Feed

Column Name	Target Field
data6	Spill
data7	Negative Tolerance
data8	Positive Tolerance
data9	Negative Percent
data10	Positive Percent

OpenTables

Format

```
bool OpenTables()
```

Description

Open the database tables for reading and writing.

Returns

True if successful, false if not successful.

Example

```
bool status = SD_client.OpenTables();
```

CloseTables

Format

```
bool CloseTables()
```

Description

Closes the database tables; no further access is possible.

Returns

True if successful, false if not successful.

Example

```
bool status = SD_client.CloseTables();
```

ParseRow

Format

```
void ParseRow(string row)
```

Description

This method should be used to interpret the string returned by methods `GetFirstStandardRow` and `GetNextStandardRow`.

This method parses `row` and uses the information to load the members of `rowData`.

<code>rowData.endOfData</code>	bool
<code>rowData.recordID</code>	string
<code>rowData.shortID</code>	string
<code>rowData.description</code>	string
<code>rowData.data1</code>	string
...	
<code>rowData.data17</code>	string

If `row` does not contain any row data, `rowData.endOfData` will be true and all the strings in `rowData` will be null.

If `row` does contain row data, `rowData.endOfData` will be false and the strings in `rowData` will contain the field data.

Returns

Nothing.

Example

```
string firstRow = SD_client.GetFirstStandardRow(SDclient.Table.A7);
SD_client.ParseRow(firstRow);
string description = "";
string data4 = "";
if (SD_client.rowData.endOfData == false)
{
    description = SD_client.rowData.description;
    data4 = SD_client.rowData.data4;
}
```

GetFirstStandardRow

Format

```
string GetFirstStandardRow(Table tableName)
```

Description

Obtains the first row of data in the specified table Table.A0 to Table.A9.

Returns

First table row data as a string with comma separated data fields or "99R001~~" if an error occurs.

Example

```
string firstRow = SD_client.GetFirstStandardRow(SDclient.Table.A7);
SD_client.ParseRow(firstRow);
string description = "";
string data4 = "";
if (SD_client.rowData.endOfData == false)
{
    description = SD_client.rowData.description;
    data4 = SD_client.rowData.data4;
}
```

GetFirstStandardRow

Format

```
string GetFirstStandardRow(Table tableName, string shortID)
```

Description

Obtains the first row of data in the specified table Table.A0 to Table.A9 with the specified shortID value.

Returns

First table row data as a string with comma separated data fields

Example

```
string firstRow = SD_client.GetFirstStandardRow(SDclient.Table.A4, "Water Feed");
SD_client.ParseRow(firstRow);
string description = "";
string data5 = "";
if (SD_client.rowData.endOfData == false)
{
    description = SD_client.rowData.description;
    data5 = SD_client.rowData.data5;
}
```

GetFirstStandardRow

Format

```
string GetFirstStandardRow(Table tableName, string sql_where, string sql_order_by)
```

Description

Obtains the first row of data in the specified table Table.A0 to Table.A9 based on the sql commands in the parameters *sql_where* and *sql_order_by*.

Returns

First table row data as a string with comma separated data fields

Example

```
string productName = "Marbles";
```

```

string firstRow = SD_client.GetFirstRow(SD_client.Table.A9,
    "shortid = '" + productName + "' and"
    (data1='CLEAR' or data1='BLUE' or
    data1='RED')", "data2,data3");
SD_client.ParseRow(firstRow);
string description = "";
string data5 = "";
string data6 = "";
string data7 = "";
if (SD_client.rowData.endOfData == false)
{
    description = SD_client.rowData.description;
    data5 = SD_client.rowData.data5;
    data6 = SD_client.rowData.data6;
    data7 = SD_client.rowData.data7;
}

```

GetNextStandardRow

Format

```
string GetNextStandardRow()
```

Description

Gets the next row that meets the requirements of the previous filter.

Returns

Table row data as a string with comma separated data fields, "END_OF_DATA" if no more data, or "99R001~~" if an error occurs.

Example

```
string tableRow = SD_client.GetNextStandardRow();
```

InsertStandardRow

Format

```
bool InsertStandardRow(Table tableName, string shortID, string description, string[] data)
```

Description

Creates a new row in the specified database table. The *shortID* must be unique. The string array *data* should contain between 0 and 17 strings.

Returns

True if successful, False if failure.

Example

```

// load data1 - data4, skip data5 & data6, load data7,
// skip data8, load data9, skip data10, load data11 &
// data12

```



```
string rawData = "MAKE_SOUP,Tomato Soup,1,0,,,Paste,,20 kg,,0.5kg,0.5kg";
string[] data = rawData.Split(','); // data array
bool status = SD_client.InsertStandardRow(SDclient.Table.A3,
    "Tomato Soup", "Classic Tomato Soup", data);
if (!status)
    // show error
else
    // continue on
```

EditStandardRow

Format

```
bool EditStandardRow(Table tableName, string shortID, string
description, string[] data)
```

Description

Modifies the specified row in the specified database table Table.A0 to Table.A9

Returns

True if successful, False if the shortID is not found or other data error.

Example

```
// load data1 - data4, skip data5 & data6, load data7,
// skip data8, load data9, skip data10, load data11 &
// data12
string rawData = "MAKE_SOUP,Potato Soup,1,0,,,Paste,,40
    kg,,1.5kg,1.0kg";
string[] data = rawData.Split(','); // data array
bool status = SD_client.EditStandardRow(SDclient.Table.A3,
    "Potato Soup", "Classic Potato Soup", data);
if (!status)
    // show error
else
    // continue on
```

EditStandardItem

Format

```
bool EditStandardItem(Table tableName, string entryID, Column columnName, string data)
```

Description

Modifies the specified Item in the specified database table Table.A0 to Table.A9. The *entryID* is a unique identifier (GUID) for a row. The *columnName* ranges from Column.Description, Column.Data1 to Column.Data17.

Returns

True if successful, False if failure.

Example

```
bool status = SD_client.EditStandardItem(SDclient.Table.A1,
    "{01EC6F7B-D6C3-DA7B-DFA1-33118B3BA1C7}",
    SDclient.Column.Data3, "22000");
```

DeleteStandardRow

Format

```
void DeleteStandardRow(Table tableName, string shortID)
```

Description

Deletes the specified row(s) by the shortID in the specified standard database table

Returns

Nothing.

Example

```
SD_client.DeleteStandardRow(SDclient.Table.A1, "TOMATO_SOUP");
```

DeleteStandardRowEntryID

Format

```
void DeleteStandardRowEntryID(Table tableName, string entryID)
```

Description

Deletes the specified row with the specified entryID (GUID) in the specified database table Table.A0 to Table.A9

Returns

Nothing

Example

```
SD_client.DeleteStandardRowEntryID(SDclient.Table.A3,
    "{4CBE0B33-57AF-BD5D-2E3D-F698F72FCB13}");
```

DeleteTable

Format

```
void DeleteTable(Table tableName)
```

Description

Deletes all rows in the specified standard database table

Returns

Nothing.

Example

```
SD_client.DeleteTable(SDclient.Table.A4);
```

Shared Data Client Class Metrology Methods

Clear

Format

```
bool Clear(ref string clearStatus)
```

Description

Clears tare and loads *clearStatus* with the status of the clear operation for the selected scale.

Returns

True if successful - no status message is loaded

False if not successful - a status message is loaded and is one of the following:

- "Tare Disabled"
- "Scale in Invalid Mode"
- "IDNet Scale Error"
- "Status has not updated in the time allowed"
- "Clear Tare Failed, status = xxx"

Example

```
string status = "";  
if (!SD_client.Clear(ref status))  
    MessageBox.Show(status);
```

TarePB

Format

```
bool TarePB(ref string tareStatus)
```

Description

Performs a pushbutton tare and loads *tareStatus* with the status of the tare operation for the selected scale.

Returns

True if successful - no status message is loaded

False if not successful - a status message is loaded and is one of the following:

- "Tare Failure - In Motion"
- "Tare Disabled"
- "Tare Failure - Value Too Small"
- "Tare Failure - Scale Overcapacity"
- "Tare Failure - Below Zero"
- "Tare Value > Scale Capacity"
- "Scale in Invalid Mode"
- "IDNet Scale Error"
- "Status has not updated in the time allowed"
- "Tare Failed, status = xxx"

■ Note: The response can take up to one second to occur.

Example

```
string status = "";  
if (!SD_client.TarePB(ref status))  
    MessageBox.Show(status);
```

TarePreset

Format

```
bool TarePreset(string preset, ref string tareStatus)
```

Description

Uses the tare value in *preset* to perform a preset tare and loads *tareStatus* with the status of the tare operation for the selected scale. The *preset* weights units are assumed to match that of the selected scale.

Returns

True if successful - no status message is loaded

False if not successful - a status message is loaded and is one of the following:

- "Tare Failure - In Motion"
- "Tare Disabled"
- "Tare Failure - Value Too Small"
- "Tare Failure - Scale Overcapacity"
- "Tare Failure - Below Zero"
- "Tare Value > Scale Capacity"
- "Scale in Invalid Mode"
- "Invalid Tare"
- "IDNet Scale Error"
- "Status has not updated in the time allowed"
- "Tare Failed, status = xxx"

Example

```
string status = "";  
if (!SD_client.TarePreset("150", ref status))  
    MessageBox.Show(status);
```

Zero

Format

```
bool Zero(ref string zeroStatus)
```

Description

Zeroes the selected scale (if in range) or flow meter and loads *zeroStatus* with the status of the zero operation

Returns

True if successful - no status message is loaded

False if not successful - a status message is loaded and is one of the following:

- "Zero Failed - Scale in Net Mode"
- "Zero Failed - Out of Range"
- "Zero Failed - In Motion"
- "Zero Failed - Zero Disabled"
- "Zero Failed - IDNet Zero Timeout"
- "Zero Failed - IDNet Command Timeout Error"
- "Zero Failed - IDNet Scale Communications Disabled"
- "Scale in Invalid Mode"

- "Status has not updated in the time allowed"
- "Zero Failed, status = xxx"

Example

```
string status = "";  
if (!SD_client.Zero(ref status))  
    MessageBox.Show(status);
```

SelectScale

Format

```
bool SelectScale(string scaleNo, ref string selectScaleStatus)
```

Description

Selects the scale *scaleNo* for control (tare, clear, print, etc.) and loads *selectScaleStatus* with the status of select scale operation.

Returns

True if successful - no status message is loaded

False if not successful - this status message is loaded:

"Invalid Selected Scale"

Example

```
string status = "";  
if (!SD_client.SelectScale("2", ref status))  
    MessageBox.Show(status);  
else  
    SD_client.ZeroScale(ref status);
```

Print

Format

```
bool Print(ref string printStatus)
```

Description

Initiates a print for the selected scale and loads *printStatus* with the status of the print operation.

Returns

True if successful - no status message is loaded

False if not successful - a status message is loaded and is one of the following:

- "Printing in Progress"
- "Print Connection Not Found"
- "Printing Busy"
- "Printing Error"
- "Not Ready to Print"
- "Print Failed, Scale in Motion"
- "Print Failed, Scale Overcapacity"
- "Print Failed, Scale Under Zero"
- "Print Not Allowed"
- "Printing Not Enabled"
- "Weight Under Minimum Print Weight"
- "Print Error xxx"

Example

```
string status = "";  
if (!SD_client.SelectScale("2", ref status))  
    MessageBox.Show(status);  
else  
    SD_client.Print(ref status);
```

DisplayWt

Format

```
int DisplayWt(params Display[] displayMode)
```

Description

This method changes the weight display format and returns the y position of the bottom of the weight display.

The Resident Scale Task (RST) controls the use of the weight display area. This task is part of the IND780's main executable program that is always running. Any legal-for-trade custom application that uses the weight display must not place any part of its form(s) within the weight display area. To do so violates the legal-for-trade approval of the terminal. The RST will shut down the custom application if it attempts to place any part of its form(s) within the weight display area.

The author of the custom application should use this method to determine the bottom Y position of the weight display. This position can be used for the top of the custom application form(s) that require the use of the weight display.

This method formats the weight display according to the *displayMode* parameters used.

DisplayWt can accept up to six parameters. Usually just one or two parameters are used. Even though this method does not restrict the ordering of parameters, it is recommended to follow the ordering below:

Parameter 1 - {`Display.On` | `Display.Off`}

- Turns the weight display on or off.

Parameter 2 - {`Display.All` | `Display.Active` | `Display.ActiveDevice`}

`Display.All` - Display all active scales / flow meters
`Display.Active` - Display selected scale or flow meter
`Display.ActiveDevice` - Display one selected scale or flow meter

Parameter 3 - {`Display.Tare` | `Display.Always` | `Display.Never` | `Display.Rate`}

`Display.Tare` - Display Tare when available
`Display.Always` - Always display tare
`Display.Never` - Never display tare
`Display.Rate` - Display Rate when available

Parameter 4 - {`Display.Compress` | `Display.Uncompress`}

- Compresses or uncompresses the weight display

Parameter 5 - {`Display.Small` | `Display.Medium` | `Display.MediumHalf` | `Display.Large` | `Display.LargeHalf` | `Display.Larger` | `Display.Huge`}

- Changes the size of the weight display for scales 1 to 4 and flow meters

Parameter 6 - {`Display.SumSmall` | `Display.SumMedium` | `Display.SumMediumHalf` | `Display.SumLarge` | `Display.SumLargeHalf` | `Display.SumLarger` | `Display.SumHuge`}

- Changes the size of the sum scale weight display.

Returns

The Y-coordinate of the bottom of the weight display area.

Examples

To compress weight:

```
SD_client.DisplayWt(Display.Compress);
```

To uncompress weight:

```
SD_client.DisplayWt(Display.Uncompress);
```

To turn off weight:

```
SD_client.DisplayWt(Display.Off);
```

To turn on weight:

```
SD_client.DisplayWt(Display.On);
```

To display only the active scale weights:

```
SD_client.DisplayWt(Display.Active);
```


To display all scale weights:

```
SD_client.DisplayWt(Display.All);
```

To display medium-half weights for the scales and sum scale:

```
SD_client.DisplayWt(Display.MediumHalf,  
                    Display.SumMediumHalf);
```

To display large weights for the scales and sum scale:

```
SD_client.DisplayWt(Display.Large, Display.SumLarge);
```

To display medium weights for the scales and sum scale:

```
SD_client.DisplayWt(Display.Medium, Display.SumMedium);
```

To always display the tare:

```
SD_client.DisplayWt(Display.Always);
```

To display tare only when active:

```
SD_client.DisplayWt(Display.Tare);
```

Using the return value:

```
int wtDisplayBottomY = SD_client.DisplayWt(Display.On);  
this.Top = wtDisplayBottomY; // set top edge of form
```

StartSetup

Format

```
void StartSetup()
```

Description

Starts the control panel program setup.exe in the IND780. This method should only be called from within a form closing event handler.

Returns

Nothing

Example

```
/*-----*  
* Name      :MyForm::MyForm_Closing  
* Description :Event handler called when form is closing  
* Return Value :none  
* -----*/  
void MyForm_Closing(object sender, System.ComponentModel.CancelEventArgs e)  
{  
    SD_client.StartSetup();  
}
```

KillSetup

Format

```
void KillSetup()
```

Description

Shuts down the control panel program setup.exe if it is running in the IND780. Normally the setup tree program setup.exe is not running when the custom application is executing. This method gives the programmer the ability to shut down the control panel setup tree program if needed.

Returns

Nothing

Example

```
if (doneWithSetup)
{
    SD_client.KillSetup();
}
```

Command

Format

```
string Command(string commandString)
```

Description

Issues a raw shared data command to the Shared Data Server. The user must strip off the header (i.e. "\n\r00R002~") and/or trailer ("\n\r>") from the raw data.

The following commands are supported:

READ or R	LOAD	CLOSETABLES
WRITE or W	SAVE	SETROW
SYSTEM	HELP	SELECTID
CALLBACK	NOOP	SELECTROW
XCALLBACK	CONTOUT	SELECTALL
GROUP	XCONTOUT	NEXTROW
RGROUP	PRINTOUT	SETID
XGROUP	XPRINTOUT	SETITEM
CTIMER	PARAM	DELID
UNICODE	OPENTABLES	DELROW

DELTABLE	ENDTRANS	CLOSEFILE
SQLTABLE	TRANSTIME	READFILE
BEGINTRANS	OPENFILE	WRITEFILE

- Please reference the IND780 Terminal Technical Manual 64057242 or the IND780 Terminal Shared Data Reference 64059110 for details on how to use these commands.

Returns

The raw shared data response string

Example

```
String rawData = SD_client.Command("Read wt0102");
if (rawData.Length > 12) // header + trailer length
{
    // typical read header = "\n\r00R001~"
    if (rawData.Substring(2,2) == "99")
    {
        // Should be "00", "99" means error
        // add code here to display error
    }
    else
    {
        string goodData = "";
        // subtract off header and trailer lengths:
        int length = rawData.Length - 12;
        // get actual data:
        goodData = rawData.Substring(9, length);
        // more code here to process the good data
    }
}
else
{
    // didn't get a valid read for some reason, this
    // should never occur, but an error should be
    // displayed here
}
```

CommandCB

Format

```
string Command(string commandString)
```

Description

Issues a raw shared data command to the Shared Data Server Callback connection. The user must strip off the header (i.e. "\n\r00R002~") and/or trailer ("\n\r>") from the raw data.

The following commands are supported:

CALLBACK

XCALLBACK

GROUP

XGROUP

- Please reference the IND780 Terminal Technical Manual 64057242 or the IND780 Terminal Shared Data Reference 64059110 for details on how to use these commands.

Returns

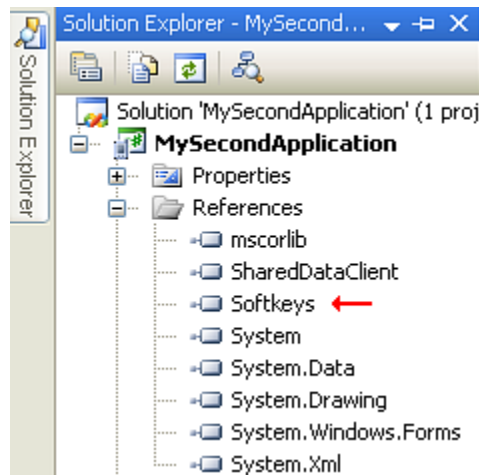
The raw shared data response string

Example

```
String rawData = SD_client.CommandCB("xcallback all");
if (rawData.Length > 13) // header + trailer length
{
    // typical delete all callbacks response =
    // "\n\r00X001~OK\n\r>"
    if (rawData.Substring(2,2) == "99")
    {
        // Should be "00", "99" means error
        // add code here to display error
    }
    else if (rawData.Substring(9,2) == "OK")
    {
        // All callbacks were deleted
        // add other code here
    }
}
else
{
    // didn't get a valid response for some reason, this
    // should never occur, but an error should be
    // displayed here
}
```

Using the Softkeys Class

A reference to the Softkeys.dll must be added to the application as shown below. Adding the Softkeys.dll reference requires entering the path where this file resides on the PC. This exposes the Softkeys class to the application.



Add the “`using MettlerToledo.IND780.Softkeys`” line to the application to simplify method calls.

The user must create an instance of the `Softkeys` class for each thread that requires `Softkeys` access.

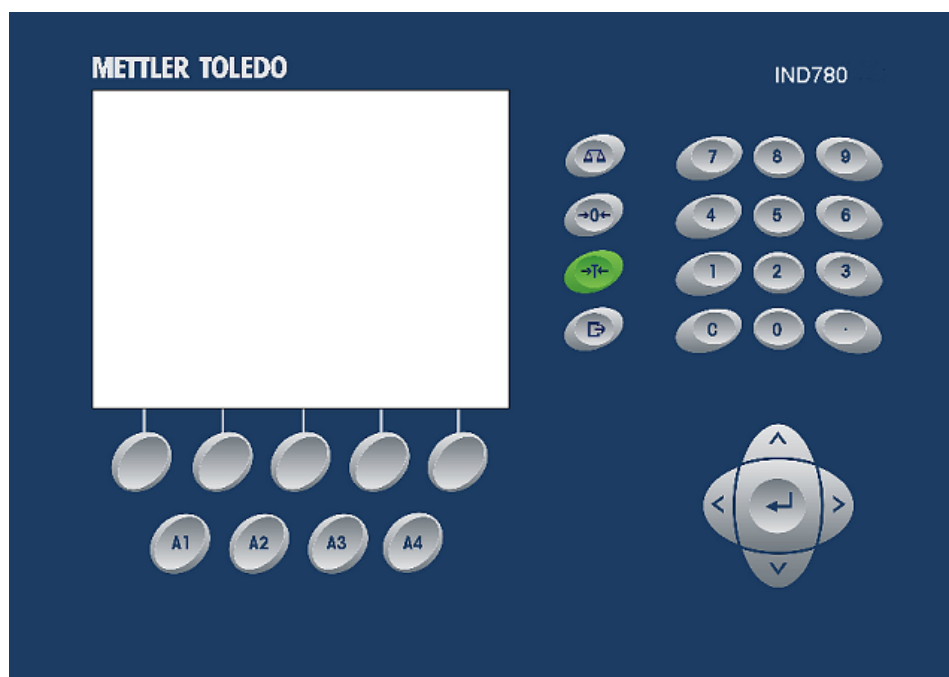
```
using System.Windows.Forms;
using MettlerToledo.IND780.Softkeys;
namespace MyThirdApplication
{
    public partial class Form1 : Form
    {
        private Softkeys mySoftkeys = null;
```

Definition of Softkeys

Softkeys refer to the first row of five keys directly below the display. Each key can be assigned a specific function. A graphic directly above the key indicates its operation such as setting the time and date, initiating a report printout, or starting a process. It is up to the application to define the keys usage.

There can be up to five levels of Softkeys. Five Softkeys are available if a single level is used. Multiple levels require the use of the fifth Softkey for navigation between levels. Refer to the figure below:

Definition of Application and Scale Keys



The Application keys are the four keys labeled A1 to A4. The user's application can assign each key a specific function.

The Scale keys are the four keys to the right of the display. They are normally used to switch scales, zero the scale, tare the scale, and print. The user's application can assign new functions to these keys.

Unlike the Softkeys, the Application and Scale keys only have one level.

Softkeys Methods

Softkeys (constructor)

Format

```
public Softkeys()
```

Description

Constructor

Returns

Nothing

Example

```
Softkeys mySoftkeys = new Softkeys();
```

AllocateSoftkeys

Format

```
bool AllocateSoftkeys(System.Windows.Forms.Form Form, int  
softkeysTopPosition, string bitmapRootPath)
```

Description

Creates picture boxes on the Form to hold the bit maps for the Softkeys. The parameter softkeysTopPosition specifies the Y coordinate for the top of the softkeys. A value of 0 will default to a position 170 pixels down from the top of the Form. The parameter bitmapRootPath is the root path to the users Softkey graphics.

Returns

Always true

Example

```
/*-----*  
* Name      :MyForm::SoftkeysInitialization  
* Description :Creates the five picture boxes and draws  
*             :the softkey borders used by the softkeys  
* Return Value :None  
* -----*/  
private void SoftkeysInitialization()  
{  
    mySoftkeys = new Softkeys();  
    mySoftkeys.AllocateSoftkeys(this, 195, @"Storage Card\Terminal\SKBMP\COLOR\");  
}
```

DrawSoftkeyBorders

Format

```
bool DrawSoftkeyBorders(System.Windows.Forms.Form Form)
```

Description

Draws the Softkeys borders at the bottom of the display.

Returns

Always true

Example

```
/*-----*  
* Name      :MyForm::MyForm_Paint  
* Description :The MyForm Paint event handler.  
* Return Value :None  
* -----*/
```

```
void MyForm_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{
    mySoftkeys.DrawSoftkeyBorders(this);
}
```

LoadSoftkeys

Format

```
bool LoadSoftkeys(int level, int position, bool enabled,
string bitmapPath, mySKMethodDelegate mySoftkeyMethod)
```

Description

The softkey is enabled by setting parameter *enabled* to true. The *bitmapPath* is the full path and filename of the bitmap. For example:

```
@\"Storage Card\Terminal\SKBMP\COLOR\escape.bmp\"
```

The method *mySoftkeyMethod* will be called when the softkey is pressed.

The method must have the signature:

```
void mySoftkeyMethod()
```

The parameter *level* has the range of 0 to 4. The parameter *position* can be from 0 to 4.

Returns

True if successful, false if the *level* or *position* is out of range.

Example

```
/*-----*
* Name      :MyForm::RegisterSoftkeys
* Description :Registers the softkey graphic and method used
               :by each softkey.
* Return Value :None
* -----*/
private void RegisterSoftkeys()
{
    // level 0 softkeys
    mySoftkeys.LoadSoftkeys(0, 0, true, @"Storage
Card\Terminal\SKBMP\COLOR\exit.bmp",
(Softkeys.mySKMethodDelegate)WeighExit);
    mySoftkeys.LoadSoftkeys(0, 3, true, @"Storage
Card\Terminal\SKBMP\COLOR\stop.bmp",
(Softkeys.mySKMethodDelegate)WeighStop);
    mySoftkeys.LoadSoftkeys(0, 4, true, @"Storage
Card\Terminal\SKBMP\COLOR\report.bmp",
(Softkeys.mySKMethodDelegate)WeighReport);
    mySoftkeys.NumberOfLevels = 1;
    mySoftkeys.DisplaySoftKeys(0);
}
```


UpdateSoftkey

Format

```
bool UpdateSoftkey(int level, int position, bool enabled,  
string bitmapPath)
```

Description

The Softkey is enabled by setting parameter *enabled* to true. The *bitmapPath* is the full path and filename of the bitmap. For example:

```
@\"Storage Card\Terminal\SKBMP\COLOR\escape.bmp\"
```

The parameter *level* has the range of 0 to 4. The parameter *position* can be from 0 to 4. UpdateSoftkey can re-assign a bitmap to another location, but it still uses the mySoftKeyMethod assigned by LoadSoftkeys.

Returns

True if successful, false if the *level*/or position is out of range.

Example

```
mySoftkeys.UpdateSoftkey(0, 0, false, @\"Storage Card\Terminal\SKBMP\COLOR\exit.bmp\");
```

DisplaySoftkeys

Format

```
bool DisplaySoftKeys(int level)
```

Description

The graphics for all of the enabled Softkeys for that *level* are displayed. The parameter *level* can be from 0 to 4.

Returns

True if successful, false if the *level* is out of range or a graphic is not found.

Example

```
mySoftkeys.DisplaySoftKeys(0); // Displays the first level
```

EnableSoftkey

Format

```
bool EnableSoftKey(int level, int position, bool state)
```

Description

Enable or disable the specified softkey. The parameter *level* can be from 0 to 4. The parameter *position* can be from 0 to 4.

Returns

True if successful, false if the *level* or *position* is out of range.

Example

```
mySoftkeys.EnableSoftKey(1, 2, false); // disable Softkey
```

DisplayNextSoftkeyLevel

Format

```
bool DisplayNextSoftkeyLevel()
```

Description

Displays the next level of Softkeys if multiple levels are used.

Returns

True if multiple levels are used, false if not.

Example

In the following example, when the fourth softkey in level 1 is pressed, the method NextSoftKeysLevel is called. This in turn calls the DisplayNextSoftkeyLevel method.

```
/*-----*
 * Name      :MyForm::RegisterSoftkeys
 * Description :Registers the softkey graphic and
 *             :method used by each softkey.
 * Return Value :None
 * -----*/
private void RegisterSoftkeys()
{
    // level 0 softkeys
    mySoftkeys.LoadSoftkeys(0, 0, true, @"Storage
Card\Terminal\SKBMP\COLOR\exit.bmp",
        (Softkeys.mySKMethodDelegate)WeighExit);
    mySoftkeys.LoadSoftkeys(0, 1, true, @"Storage
Card\Terminal\SKBMP\COLOR\start.bmp",
        (Softkeys.mySKMethodDelegate)WeighStart);
    mySoftkeys.LoadSoftkeys(0, 2, true, @"Storage
Card\Terminal\SKBMP\COLOR\stop.bmp",
        (Softkeys.mySKMethodDelegate)WeighStop);
    mySoftkeys.LoadSoftkeys(0, 3, true, @"Storage
Card\Terminal\SKBMP\COLOR\report.bmp",
        (Softkeys.mySKMethodDelegate)WeighReport);
}
```

```
mySoftkeys.LoadSoftkeys(0, 4, true, @"Storage
Card\Terminal\SKBMP\COLOR\advance.bmp",
    (Softkeys.mySKMethodDelegate)NextSoftKeysLevel);
// level 1 softkeys
mySoftkeys.LoadSoftkeys(1, 0, true, @"Storage
Card\Terminal\SKBMP\COLOR\edit.bmp",
    (Softkeys.mySKMethodDelegate)WeightEdit);
mySoftkeys.LoadSoftkeys(1, 4, true, @"Storage
Card\Terminal\SKBMP\COLOR\advance.bmp",
    (Softkeys.mySKMethodDelegate)NextSoftKeysLevel);
mySoftkeys.NumberOfLevels = 2;
mySoftkeys.DisplaySoftKeys(0); // display first level
}
/*-----*
* Name      :MyForm::NextSoftKeysLevel
* Description:Displays the next level of softkeys
* Return Value :None
* -----*/
private void NextSoftKeysLevel()
{
    mySoftkeys.DisplayNextSoftkeyLevel();
}
```

SoftkeyDispatcher

Format

```
bool SoftkeyDispatcher(int keycode)
```

Description

This method calls the Softkey method assigned to the specific Softkey by the LoadSoftkeys method. Below is a list of key codes assigned to each Softkey:

Softkey	Key Code
0 (far left)	112
1	113
2	114
3	115
4 (far right)	116

Any control that can have the focus on the form must call a common key up event handler. If there are no controls on the form then the form itself must have a key up event handler. This common key up event handler must call the SoftkeyDispatcher method. It is up to the form designer to add the necessary logic to handle keys that are outside of the Softkeys key code range.

Returns

True if successful, false if no method has been assigned.

Example

```
/*-----*
* Name      :MyForm::Common_KeyUp
* Description :Common key-up event handler for all keys.
*           :Calls the softkeys dispatcher
* Return Value :None
* -----*/
void Common_KeyUp(object sender, System.Windows.Forms.KeyEventArgs e)
{
    // If key is intended for the sender, consume it
    // and exit, else fall through to the dispatcher
    mySoftkeys.SoftkeyDispatcher((int)e.KeyCode);
}
```

NumberOfLevels

Format

```
int NumberOfLevels
```

Description

This property represents the number of levels used. The set operation ignores values less than 1 or values greater than 5.

Returns

Returns the number of levels used.

Examples

```
int levelsUsed = mySoftkeys.NumberOfLevels; // get levels
mySoftkeys.NumberOfLevels = 3; // set # of levels to 3
```

SoftkeyLevel

Format

```
int SoftkeyLevel
```

Description

This property represents the current softkey level.

Returns

The current softkey level (1 to 5)

Example

```
if (mySoftkeys.SoftkeyLevel == 5) // if level 5
    mySoftkeys.SoftkeyLevel = 1; // change to level 1
```

SoftkeyLevelDisplay

Format

```
bool SoftkeyLevelDisplay
```

Description

This property represents the state of the Softkey level display. If SoftkeyLevelDisplay is true, the softkey level number is shown to the right of the far right softkey.

Returns

True if the level number is being displayed, false if not.

Example

```
if (!mySoftkeys.SoftkeyLevelDisplay)           // if off
    mySoftkeys.SoftkeyLevelDisplay = true;    // turn it on
```

Application and Scale Keys Methods

LoadApplicationAndScaleKeys

Format

```
void LoadApplicationAndScaleKeys(int position, bool enable,
mySKMethodDelegate myApplScaleMethod)
```

Description

Assigns a delegate method to a specific Application or Scale key and enables or disables the key operation.

The *position* value for each key assignment is as follows:

Position	Key
0	A1 Application key
1	A2 Application key
2	A3 Application key
3	A4 Application key
4	Select Scale key
5	Zero Scale key
6	Tare Scale key
7	Print Scale key

Setting *enable* to true enables the Application / Scale key.

The delegate method *myApplScaleMethod* handles the Application / Scale function.

The method must have the signature:

```
void myApplScaleMethod()
```

Returns

Nothing

Example

```
private void InitializeSoftkeys()
{
    mySoftkeys.LoadApplicationAndScaleKeys(0, true,
        (Softkeys.mySKMethodDelegate)myApplicationMethodForA1);
    mySoftkeys.LoadApplicationAndScaleKeys(1, true,
        (Softkeys.mySKMethodDelegate)myApplicationMethodForA2);
    mySoftkeys.LoadApplicationAndScaleKeys(2, true,
        (Softkeys.mySKMethodDelegate)myApplicationMethodForA3);
    mySoftkeys.LoadApplicationAndScaleKeys(3, true,
        (Softkeys.mySKMethodDelegate)myApplicationMethodForA4);
    mySoftkeys.LoadApplicationAndScaleKeys(4, true,
        (Softkeys.mySKMethodDelegate)
        myApplicationMethodForSelectScale);
    mySoftkeys.LoadApplicationAndScaleKeys(5, true,
        (Softkeys.mySKMethodDelegate)
        myApplicationMethodForZeroScale);
    mySoftkeys.LoadApplicationAndScaleKeys(6, true,
        (Softkeys.mySKMethodDelegate)
        myApplicationMethodForTareScale);
    mySoftkeys.LoadApplicationAndScaleKeys(7, true,
        (Softkeys.mySKMethodDelegate)
        myApplicationMethodForPrintScale);
}
// Softkeys delegate methods
private void myApplicationMethodForA1()
{
    // application code to handle the A1 key here
}
private void myApplicationMethodForA2()
{
    // application code to handle the A2 key here
}
.
.
.
private void myApplicationMethodForPrintScale()
{
    // application code to handle Print key here
}
```

ApplicationAndScaleKeysDispatcher

Format

```
bool ApplicationAndScaleKeysDispatcher(int keycode)
```

Description

Calls the method assigned in LoadApplicationAndScaleKeys per the *keycode*.

Keycode	Application / Scale key
121	A1 Application key
122	A2 Application key
123	A3 Application key
93	A4 Application key
117	Switch Scale key
118	Zero Scale key
119	Tare Scale key
120	Print Scale key

Any control that can have the focus on the form must call a common key up event handler. If there are no controls on the form then the form itself must have a key up event handler. This common key up event handler must call the ApplicationAndScaleKeysDispatcher method. It is up to the form designer to add the necessary logic to handle keys that are outside of the application and scale key code range.

Returns

True if successful, false if no delegate method found.

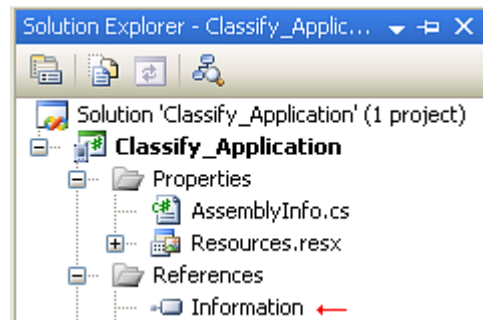
Example

```
void Common_KeyUp(object sender, Forms.KeyEventArgs e)
{
    int keyCode = (int)e.KeyCode;
    // If key is intended for the sender, consume it and exit, else
    // fall through to the dispatchers
    if (!mySoftkeys.SoftkeyDispatcher(keyCode))
    {
        mySoftkeys.ApplicationAndScaleKeysDispatcher(keyCode);
    }
}
```

Using the Information Class

The Information Class supplies boiler plate Legal-For-Trade information about the IND780 terminal. This class duplicates the sequence used on the IND780 terminal for displaying scale weight data, terminal software serial numbers, and metrological information.

A reference to the Information.dll must be added to the application as shown below. Adding the Information.dll reference will require entering the path where this file resides on the PC. This will expose the Information class to the application.







Add the “`using MettlerToledo.IND780.Information`” line to the application to simplify method calls.

```
using MettlerToledo.IND780.Information
namespace Classify_Application
{
    public partial class ClassifyMain : Form
```


It is highly recommended that the custom application developer use this class if the application is intended for Legal-For-Trade.

What the Information Class Includes


Recall Information Form

The Recall Information form displays the IP Address of the terminal. It also includes four Softkeys; one to exit the form  , one to open the Weight Recall form  , one to open the System Information Recall form  , and one to open the Metrology Recall form  .


Weight Recall Form

The Weight Recall form displays the gross, tare, and net weights for the attached scales. This form also includes an exit Softkey  .

System Information Recall Form


The System Information Recall form displays the ID numbers assigned to the IND780, hardware part numbers and serial numbers, system software numbers, and installed custom application names. This form also includes an exit Softkey  .

Metrology Recall Form

The Metrology Recall form displays the main software version number and checksum, security switch status, and calibration dates of the attached scales. This form also includes an exit Softkey .

Adding the Information Softkey

Registering the Softkey

Two methods are called from the form's constructor to add Softkeys to the form, SoftkeysInitialization and RegisterSoftkeys. The first method creates a new instance of the Softkeys class and loads the path that contains the softkey's graphics. The RegisterSoftkeys method places the Softkeys at specific locations and references the methods used by the Softkeys. A red box indicates the line of code that registers the Information Softkey .

```
/*-----*
 * Name      :ClassifyMain::ClassifyMain
 * Description :Constructor
 * Return Value :None
 * -----*/
public ClassifyMain()
{
    SD_client = new SDclient();
    InitializeComponent();
    this.SharedDataServerConnect();
    this.SoftkeysInitialization();
    this.RegisterSoftkeys();
    ClassifyMain.GetLanguage();
}

/*-----*
 * Name      :Form1::SoftkeysInitialization
 * Description :Creates the five picture boxes and
 *             :draws the softkey borders used by
 *             :the softkeys
 * Return Value :None
 * -----*/
private void SoftkeysInitialization()
{
    mySoftkeys = new Softkeys();
    mySoftkeys.AllocateSoftkeys(this, 175, @"Storage
    Card\Terminal\SKBMP\COLOR\");
}


/*-----*
 * Name      :Form1::RegisterSoftkeys
 * Description :Registers the softkey graphic and
 *             :method used by each softkey.
 * Return Value :None
 * -----*/
private void RegisterSoftkeys()
{
    // level 0 softkeys
    mySoftkeys.LoadSoftkeys(0, 0, true, @"Storage
    Card\Terminal\SKBMP\COLOR\exit.bmp",
    (Softkeys.mySKMethodDelegate)ClassifyExit);
    mySoftkeys.LoadSoftkeys(0, 1, true, @"Storage
    Card\Terminal\SKBMP\COLOR\classifyColors.bmp",
    (Softkeys.mySKMethodDelegate)ClassifyColors);
}
```

```

mySoftkeys.LoadSoftkeys(0, 2, true, @"Storage
Card\Terminal\SKBMP\COLOR\classifyThresholds.bmp",
(Softkeys.mySKMethodDelegate)ClassifyThresholds);
mySoftkeys.LoadSoftkeys(0, 3, true, @"Storage
Card\Terminal\SKBMP\COLOR\info.bmp",
(Softkeys.mySKMethodDelegate)ClassifyInfo);
mySoftkeys.LoadSoftkeys(0, 4, true, @"Storage
Card\Terminal\SKBMP\COLOR\advance.bmp",
(Softkeys.mySKMethodDelegate)ClassifyNextSoftkeyLevel);
mySoftkeys.LoadSoftkeys(1, 0, true, @"Storage
Card\Terminal\SKBMP\COLOR\com.bmp",
(Softkeys.mySKMethodDelegate)ClassifyComConfigure);
mySoftkeys.LoadSoftkeys(1, 1, false, @"" , null);
mySoftkeys.LoadSoftkeys(1, 2, true, @"Storage
Card\Terminal\SKBMP\COLOR\start.bmp",
(Softkeys.mySKMethodDelegate)ClassifyStart);
mySoftkeys.LoadSoftkeys(1, 3, true, @"Storage
Card\Terminal\SKBMP\COLOR\stop.bmp",
(Softkeys.mySKMethodDelegate)ClassifyKillSetup);
mySoftkeys.LoadSoftkeys(1, 4, true, @"Storage
Card\Terminal\SKBMP\COLOR\advance.bmp",
(Softkeys.mySKMethodDelegate)ClassifyNextSoftkeyLevel);
mySoftkeys.NumberOfLevels = 2;
mySoftkeys.DisplaySoftKeys(0);
}

```

Using the Information Softkey

The ClassifyInfo method below opens the Information form when the Information Softkey  is pressed.




```

/*-----*
* Name      :Form1::ClassifyInfo
* Description :Opens the Information class form
* Return Value :None
* -----*/
private void ClassifyInfo()
{
    this.label_ClassifyApplication.Text = "Loading..."
    this.Refresh();
    Information.InfoClass info = new InfoClass();
}



```

Information Class Screen Shots


Example Main Form with Information Softkey

IP=172.18.55.11				06/Mar/2012 10:10			
Classify Application							
	1 2 12		1 2 12			1	


Recall Information Form

IP=172.18.55.11				06/Mar/2012 10:23			
Recall Information							
IP Address				172.18.55.11			
		i	M			1	


Weight Recall Form

IP=172.18.55.11		06/Mar/2012 10:39	
Weight Recall			
Scale			Net
1	466 kg	0 kg T	466 kg
2	114.1 kg	0.0 kg T	114.1 kg
			1

System Information Recall Form


IP=172.18.55.11		06/Mar/2012 10:46	
System Information Recall			
	ID	PN	Software
▶	ID1	IND780	
	ID2	Mettler Toledo	
	ID3		
	HMI Color	64083260	xxxxxxxxxxxx: C
	Baseboard	64084167	xxxxxxxxxxxx: P
	MTA-ETC	V2.0	1.0
			1

Metrology Recall Form M

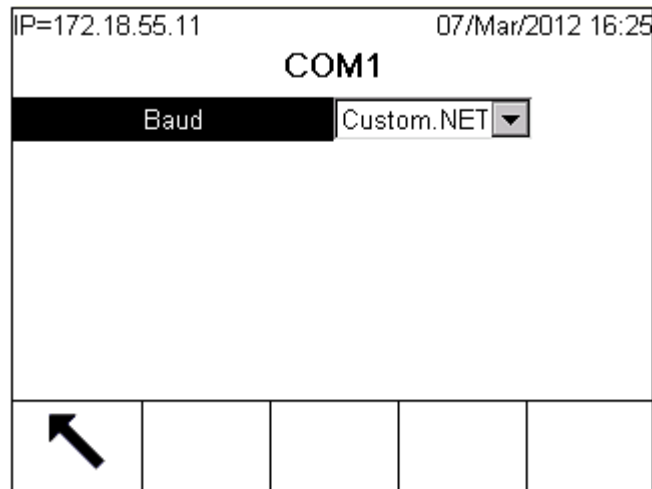
IP=172.18.55.11		06/Mar/2012 10:52	
Metrology Recall			
Version Number		7.2.06b 251636	
Security Switch		Not approved Unlocked	
Scale	Ident Code	Last Calibration	
1	--	02/Mar/2012 08:46:28	
2	--	19/Jul/2011 08:01:27	
		1	

Using Serial I/O

There are two serial ports available to a custom application. Both COM1 and COM2 are configurable for RS232 operation using the IND780 Setup tree. Serial ports COM3 and COM4 are not available for use with a custom application. The baud rate for the selected port must be set to Custom.NET to free it for use with the custom application.

IP=172.18.55.11		07/Mar/2012 16:18	
COM1			
Baud	Custom.NET ▼		
Data Bits / Parity	19200 ▲ ▼		
Flow Control	38400		
Interface	57600		
Character Set	115200		
	Custom.NET ▼		
	CP 1252 ▼		
			

Selecting Custom.NET will hide the other parameter selections on this page



The custom application must set the parameters for the selected port using the SerialPort class. The IND780 port names are “**COM1:**” and “**COM2:**”.

Custom Application Restrictions

Legal-For-Trade

If the IND780's Legal-For-Trade switch is on, the custom application must follow some special rules.

The custom application must display the Information softkey.

The custom application can still enable or disable the weight display, but the following rules apply:

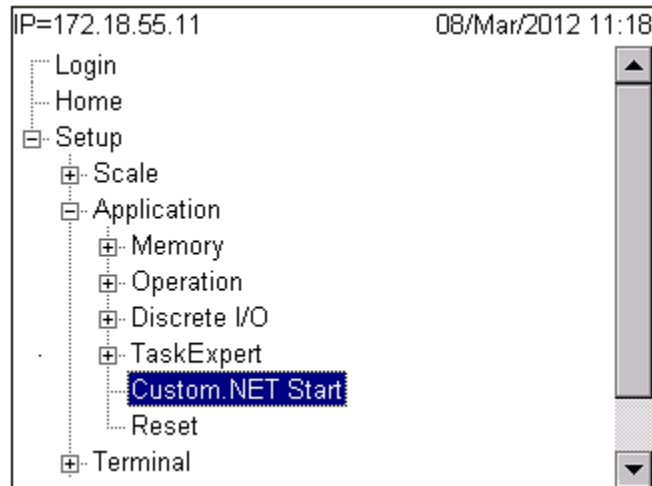
- While the weight display is enabled, the custom application may not draw anything in the status line or weight display areas. If it does, the IND780 will terminate the application and display an error message.
- While the weight display is disabled, the custom application may not draw anything in the status line area. The status line uses the top 20 pixels of the display. The status line will contain the text "Display Not Legal-for-trade". If the custom application draws anything in the status line area, the IND780 will terminate the application and display an error message.

Not Legal-For-Trade

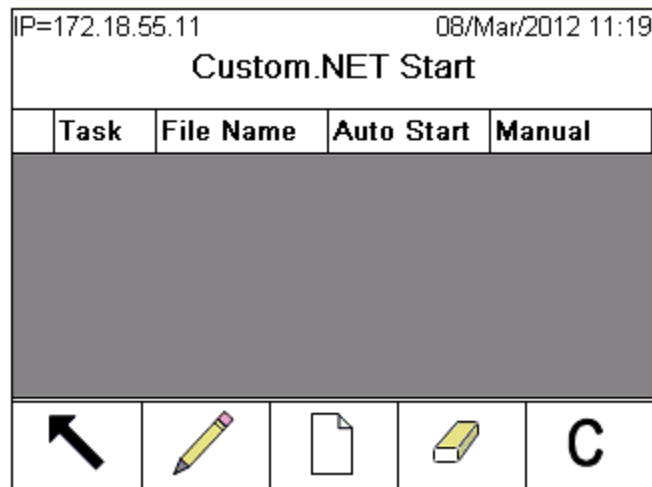
There are no display restrictions for this mode. It is still recommended to leave the top 20 pixels of the display free so that the IND780 can display any system errors.

Custom Application Configuration

The custom application must be registered with the IND780 so that it can execute properly. To do this, enter setup and navigate to the following screen:



Navigate to the Custom.NET Start selection and press Enter. The following screen will appear:



Select the New soffkey . The next screen will appear:

IP=172.18.55.11		08/Mar/2012 11:21	
Custom.NET Start New			
Task	01		
File Name	<input type="text" value="MycustomApp.exe"/>		
Auto Start	<input type="button" value="Enabled"/>		
Manual Start	<input type="button" value="Disabled"/>		
Reserve Console for App	<input type="button" value="Enabled"/>		
<div> <div>Esc</div> <div></div> <div></div> <div></div> <div>OK</div> </div>			

Enter the file name of the custom application, whether it will be started automatically or manually, and if the application will reserve the display for its use.

File Name

File names must be entered with the full file name, including the .exe.

The file name is limited to 20 characters. The custom application executable file(s) must be copied to the **Storage Card\CustomNET\Programs** folder.





Automatic Start

If Auto Start is enabled, the custom application will be started on power up and on exit of setup. Only one application should be configured to autostart. If more than one application is configured to autostart, only the first autostart application will start.

If the custom application has reserved the console, it will be its responsibility to start the IND780 setup tree application by calling the StartSetup method. In the case that the user enters the setup tree for parameter editing, the setup tree application will automatically restart the custom application upon exit and will then terminate itself.

Manual Start

A custom application can be manually started from either the control panel or from a discrete input. Either or both conditions are configurable from the setup menu.

To start a custom application from the control panel, the user must navigate to the **Terminal - Softkeys** page and add a task start softkey  ¹,  ²,  ³, or . The first icon is assigned to custom application 1, the second to custom application 2, the third to custom application 3, and the fourth brings up the list of custom applications.

To start a custom application using a discrete input, the user must navigate to the **Application - Discrete I/O - Inputs** page and enter the input address and select an assignment of either Task 1, 2 or 3.

Reserve Console for Application

Custom applications that require use of the display area must reserve the console.

If the custom application reserves the console for its use, the IND780 setup tree application will be terminated upon starting the custom application. It is up to the custom application to restart the setup tree application prior to termination. In the case that the user enters setup tree for parameter editing, the setup tree application will automatically restart the custom application upon exit and will then terminate itself.

Checksum Testing

The IND780 will perform a checksum test on any custom application prior to starting it. If the checksum fails, the custom application will not be started and an error message will be displayed.

Setup Tree Recovery

If the custom application fails to start the setup tree application and access to the setup tree application is required, the user can restart the setup tree application by powering down the IND780, turning on switch 2 on the main board, and then powering up the IND780. This will allow access to the setup tree application. Be sure to turn off switch 2 before starting the custom application.

SQL Database

The IND780 uses SQL Server 2005 Compact Edition (aka Server Compact 3.1).

The programmer must install the SQL Server 2005 Compact Edition Developer SDK on the development PC if database functionality will be used. Once this is installed, a reference to ServerCe.dll must be added to the project. The full path is:

**C:\Program Files\Microsoft SQL Server Compact
Edition\v3.1\SDK\bin\wce500**

System.Data.SqlServerCe.dll

Below is the link to the SDK:

<http://www.microsoft.com/en-us/download/details.aspx?id=10070>

Debugging

Other Versions:

- Visual Studio 2005

The first two steps, preparing the IND780 and Visual Studio, have to be done only once. The last set of steps, setting security and establishing the connection, must be repeated any time that you want to connect from a new instance of Visual Studio.

- **Note:** Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements.

To Prepare the IND780 for Connecting

1. Make an FTP connection to the IND780.
2. In the root directory, rename AutoCE.ini to AutoCE_Release.ini.
3. Copy AutoCE_Debug.ini from \CustomNET\Debug Support\ to the root directory.
4. In the root directory, rename AutoCE_Debug.ini to AutoCE.ini.

To Prepare Visual Studio for Connecting

1. On the Visual Studio **Tools** menu, click **Options**, expand **Device Tools**, and then click **Devices**.
2. Select platform Excal270 and device MSC MET_ETE_PXA270: ARMV4I_Release.
3. Click **Properties**.
4. To the right of the **Transport** box, click **Configure**.
5. In the **Configure TCP/IP Transport** dialog box, select **Use specific IP address**, and then type the IP address of the IND780.
6. Close the dialog boxes.

To Set Security and Establish the Connection

1. Cycle power on the IND780.
 2. Within three minutes, connect to the device.
- If you establish your first connection within three minutes, you can continue to deploy and debug indefinitely as long as you are using the same Visual Studio

instance. If you have to connect from another instance of Visual Studio, you must perform these security steps again.

Revision History

29 May 2012

- Added reference on installing Excal270_SDK.msi manually.
- Changed "Custom Application Toolkit" to "IND780 .NET Custom Application Toolkit".

5 June 2012

- General clean-up.

30 June 2012

- Formatted

METTLER TOLEDO

1900 Polaris Parkway
Columbus, Ohio 43240

METTLER TOLEDO® is a registered
trademark of Mettler-Toledo, LLC

©2012 Mettler-Toledo, LLC



64090418