

nbCcTalkCoinAcceptor

Nikola Brežnjak

Content

1. Introduction.....	2
2. ccTalk.....	3
3. Coin Acceptor.....	5
3.1. Using the application.....	5
3.2. Using the API.....	6
3.2.1. Add the reference to the ccTalk library.....	8
3.2.2. Connect to the coin acceptor.....	8
3.2.3. Modify inhibit status.....	10
3.2.4. Start/End polling for events.....	11
3.2.5. Start/Stop master inhibition.....	11
3.2.6. Accepting the coins.....	12

1. Introduction

This document contains usage instructions and code examples for communication with coin acceptor using the ccTalk protocol.

Revisions of this document along with their respective numbers, dates and notes (user and short explanation of the content change) are in the Table 1.1.

Table 1.1. Revision table

Revision	Date	Note
1	25.02.13	[Nikola Brežnjak] – First version of the document

2. ccTalk

ccTalk is a serial protocol in widespread use throughout the money transaction industry. The protocol was developed at a company [Crane Payment Solutions](#). The protocol uses an asynchronous transfer of character frames in a similar manner to RS232. The main difference is that it uses a single two-way communication data line for half-duplex communication rather than separate transmit and receives lines.

It operates at TTL voltages and is "multi-drop" i.e. peripherals can be connected to a common bus and are logically separated by a device address. Each peripheral on the ccTalk bus must have a unique address. The original protocol operated at 4800 baud with subsequent releases standardising on 9600 baud. Low cost bridge chips are now available from a number of manufacturers to allow ccTalk to run over USB at baud rates of at least 1 Mbit/s.

Advantages of ccTalk include low cost UART technology, a simple-to-understand packet structure, an easily expandable command interface and **no licensing requirements**. The latter affords the protocol a good deal of popularity in a crowded and highly competitive field similar to open-source software.

An Example ccTalk Message Packet looks like:

TX data = 002 000 001 245 008

- 002 = destination address
- 000 = zero data bytes
- 001 = source address
- 245 = command header 'Request equipment category id'
- 008 = checksum ($002 + 000 + 001 + 245 + 008 = 256 = 0 \text{ mod } 256$)

This is a message from address 1 (the host) to peripheral address 2 to find out what it is:

RX data = 001 013 002 000 067 111 105 110 032 065 099 099 101 112
116 111 114 022

- 001 = destination address
- 013 = 13 data bytes

- 002 = source address
- 000 = reply header
- 067...114 = ASCII for 'Coin Acceptor'
- 022 = checksum (sum of all packet bytes is zero)

The reply from address 2 back to address 1 identifies it as a coin acceptor.

In ccTalk a coin has a 6 character identifier <2-letter country code><3-letter value><1-letter issue code>. The country code conforms to ISO 3166. The issue code is assigned to different issue dates or special mint variations of the same coin. Few examples follow:

- US025A United States 25c
- GB010B Great Britain 10p
- EU200A Euro 2€

Bank notes follow the same pattern but 4 characters are allocated to the value and there is an associated scaling factor, usually x100, with the country. Few examples follow:

- US0001A - United States \$1
- GB0020A - Great Britain £20
- EU0005A - Euro 5€

3. Coin Acceptor

This project uses the aforementioned ccTalk protocol and it's based on [other ccTalk project](#) which has been discontinued. API is based on ccTalk generic specification issue 4.6 as specified on the [official ccTalk website](#).

It is a basic C# .NET assembly, built for 4.0 framework. Although the project on which this one is based has support for bill validators they have not been tested and the example WinForms application demonstrates only the usage for coin acceptor.

The main reason this project was made is because the aforementioned projects didn't work "out of the box" and didn't include enough documentation to make an easy start. Also, the coin acceptor didn't have full implementation for basic usage (for example, the initial coin inhibition setting was missing).

3.1. Using the application

The main and only form in the nbCcTalkCoinAcceptor project which demonstrates the usage of cctalklib API is shown on Illustration 1.

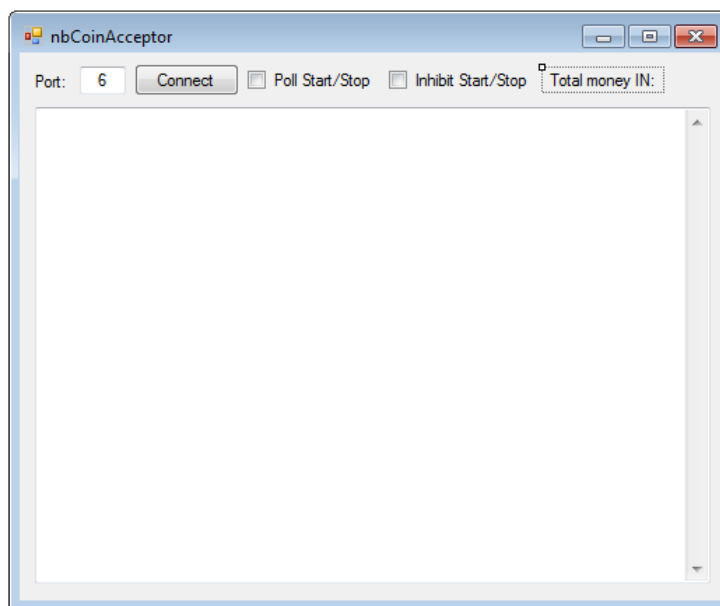


Illustration 1. Main form to demonstrate the usage of cctalklib API

You can run the application either by running the `nbCcTalkProject.exe` file in `bin` folder or by compiling it with Visual Studio. First thing to do is make sure the port number is correctly set and then click `Connect` button. After that you have to tick the `Poll Start/Stop` checkbox in order to start polling for coins. The example of these actions and few inserted coins is shown on Illustration 2 where the last two icons were inhibited due to the `Master Inhibit` status flag set intentionally during testing.

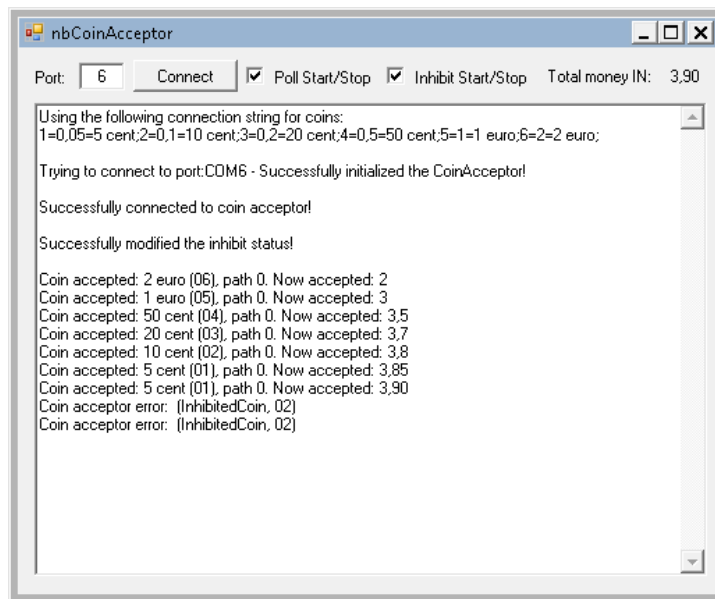


Illustration 2. Example of few inserted coins and setting the master inhibit flag

This example works with Euro currency, and if you wish to make it work with some other currency, please follow the chapter 3.2. Using the API.

3.2. Using the API

When you open up the project in Visual Studio you will see a solution explorer like the one shown on Illustration 3.

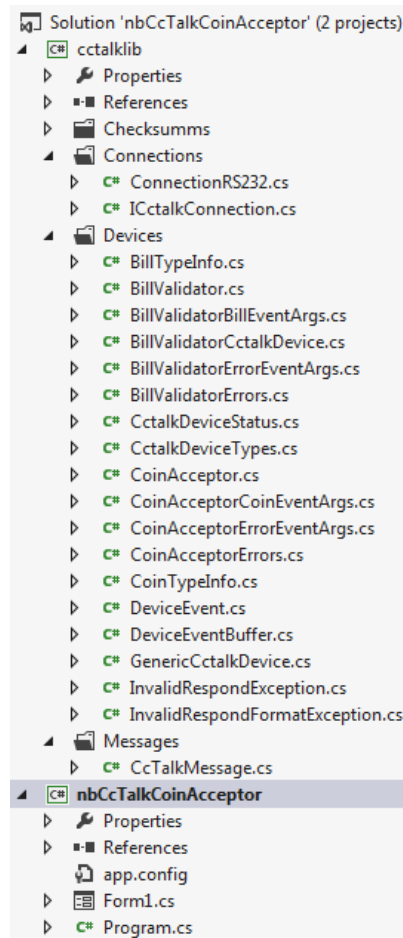


Illustration 3. Solution explorer

As you can see, there are two projects, `cctalklib` and `nbCcTalkCoinAcceptor`, where the latter uses the `ccTalk` wrapper functions defined in project `cctalklib`.

The whole code for the `nbCcTalkCoinAcceptor` project is in the `Form1.cs` file, which by itself is not a favorable approach (don't mix the presentation with logic), but it will suffice in this case of a quick demonstration on how to use the `cctalklib` API.

The steps that follow should be done exactly in that order when implementing your own solution based on the `CctalkLib.dll` library. Here I'm referring to `CctalkLib.dll` because if you build the `cctalklib` project that's the file you will get as an output, since the project is set to output type of `Class library` as shown on Illustration 4.

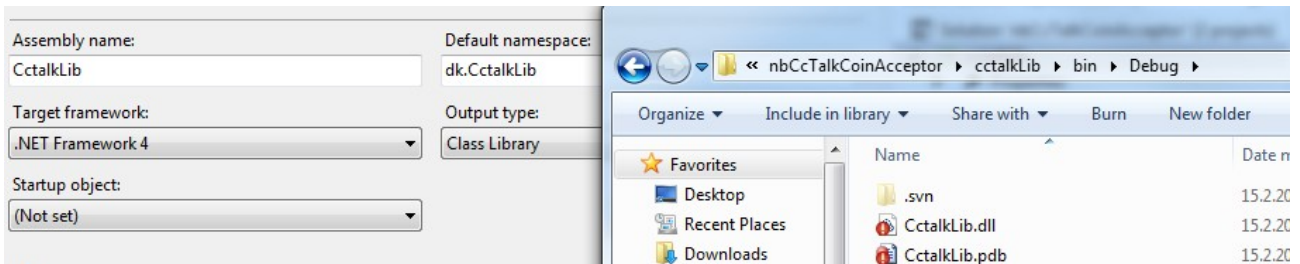


Illustration 4. cctalklib's project output set to Class Library and the output is CctalkLib.dll

3.2.1. Add the reference to the ccTalk library

First thing to do is to add a reference to the CctalkLib.dll file by using the Add Reference menu, as shown on Illustration 5. You can choose to only reference the direct dll file which you can find in the bin folder of the cctalklib project named CctalkLib.dll, as shown on Illustration 4.

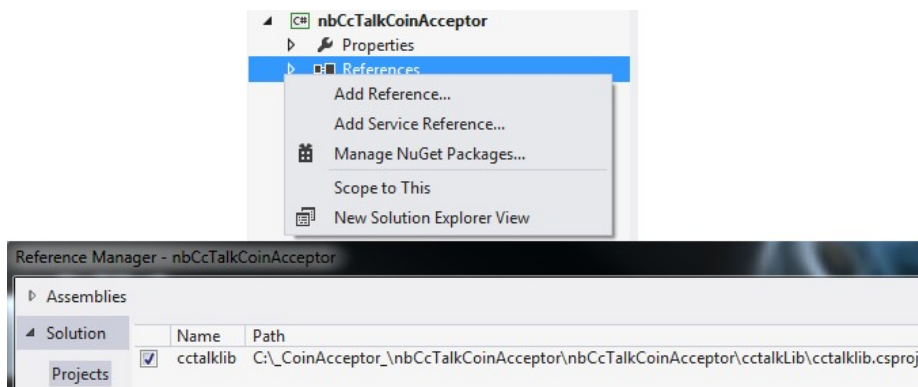


Illustration 5. Adding reference to the cctalklib project

After this you have to reference the library in your code by typing the following two `using` commands:

```
using dk.CctalkLib.Connections;
using dk.CctalkLib.Devices;
```

3.2.2. Connect to the coin acceptor

Connection to the coin acceptor will be made on the click event of the Connect button, as shown previously on Illustration 2. The function listing is shown below and the explanation follows after it:

```
private bool ConnectToCoinAcceptor()
{
    Dictionary<byte, CoinTypeInfo> coins;
    coins = CoinAcceptor.DefaultConfig;
```

```

byte deviceNumber = 2; //device number defaults to 2 - coin acceptor

//can be changed to ones liking by using SetCoins(coinsDefaultText, out coins)
//default from .dll is:
//1=0,05=5 cent; 2=0,1=10 cent; 3=0,2=20 cent; 4=0,5=50 cent; 5=1=1 euro;6=2=2 euro;

string coinsDefaultText = CoinAcceptor.ConfigWord(CoinAcceptor.DefaultConfig);
txtLog.Text += "Using the following connection string for coins:" + Environment.NewLine +
    coinsDefaultText + newline;

try
{
    string port = "COM" + txtPortNumber.Text;
    var connection = new ConnectionRs232
    {
        PortName = port,
        RemoveEcho = true
    };

    txtLog.Text += "Trying to connect to port:" + port + " - ";

    _coinAcceptor = new CoinAcceptor(deviceNumber, connection, coins, null);

    _coinAcceptor.CoinAccepted += _coinAcceptor_CoinAccepted;
    _coinAcceptor.ErrorMessageAccepted += _coinAcceptor_ErrorMessageAccepted;

    _coinAcceptor.Init(true);

    if (_coinAcceptor.IsInitialized)
    {
        txtLog.Text += "Successfully initialized the CoinAcceptor!" + newline;
        return true;
    }

    txtLog.Text += "Failed initializing the CoinAcceptor!" + newline;
    DisposeCoinAcceptor();
    connection.Close();
}
catch (Exception ex)
{
    MessageBox.Show(ex.ToString());
}

return false;
}

```

First we need to set the coins definition, and if in your case the currency is different from Euro (€), you can change the coin definition accordingly by using the `SetCoins()` function by conforming to the format listed in the code:

```
1=0,05=5 cent; 2=0,1=10 cent; 3=0,2=20 cent; 4=0,5=50 cent; 5=1=1 euro;6=2=2 euro;
```

Next, we need to set up the connection by providing the port, and most importantly setting the `RemoveEcho` to `true`. Then we set the `_coinAcceptor` variable to an instance of the of the `CoinAcceptor` object by providing it's constructor with device number (nuber 2 by default – you may need to check this with your setting), connection variable and coin definition. After this you just need to subscribe to two events from `CoinAcceptor`, call the `Init()` function and eventually check for errors like this:

```
_coinAcceptor.CoinAccepted += _coinAcceptor_CoinAccepted;
_coinAcceptor.ErrorMessageAccepted += _coinAcceptor_ErrorMessageAccepted;
```

```
_coinAcceptor.Init(true);  
  
if (_coinAcceptor.IsInitialized)  
{  
    txtLog.Text += "Successfully initialized the CoinAcceptor!" + newline;  
    return true;  
}  
  
txtLog.Text += "Failed initializing the CoinAcceptor!" + newline;  
DisposeCoinAcceptor();  
connection.Close();
```


3.2.3. Modify inhibit status

After successfully connecting to the coin acceptor you have to set the inhibit status for coins, which basically means setting which coins will be accepted and which won't. To do that, you have to call the `ModifyInhibitStatus()` function which takes two parameters which **are optional**. As stated in Part 2 official manual of the ccTalk protocol (can be obtained on the official web site: <http://www.cctalk.org/> on the right hand side of the page named Download ccTalk Part 2 v4.6 as shown on Illustration 6 – and it's also included in the Documentation folder along with this document that you're reading) you have to send two bytes of data as follows:

```
Send: [Dir] [2] [2] [231] [Data 1] [Data 2] [Chk]  
Reply: [1] [0] [Dir] [0] [Chk] -> ACK without data  
  
[Data 1] = Inhibit byte 1 (LSB), coins 1 to 8  
[Data 2] = Inhibit byte 2 (MSB), coins 9 to 16  
  
bit 0 (in Data 1): coin 1  
bit 15 (in Data 2, equivalent to bit 7 of Data 2): coin 16
```

So, `Data1` byte sets the coins 1 to 8 and `Data2` byte sets the coins 9 to 16. Basically, if you convert the byte to bits then each bit which is set to 1 means that this coin will be set to be able to accept the coins. So if we convert number 255 to bits it equals 11111111.

As mentioned, the parameters to the `ModifyInhibitStatus` function are optional, because by default they are set to numbers **255** (as `Data1` parameter) and **0** (as `Data2` parameter), which means that first 8 coins are set to be accepted.



CRANE
 PAYMENT SOLUTIONS

Search site for...

Search

Home
Products
Markets
Support
Trends & Technologies
News
About Us
Contact Us

You are here: [Home](#) > [Products](#) > [Integration & Management](#) > [ccTalk](#)

Products

Coin Payment

Bill Payment

Cashless Payment

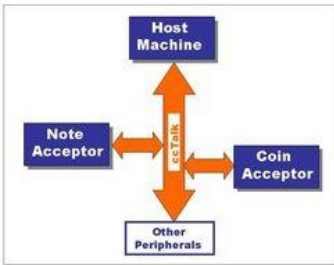
Integration & Management

Show All Integration & Management

Audit & Cash Management


System Integration

ccTalk



Summary
Benefits
Specifications
Support

Product Brochure



Click the links below to download a product brochure/s (PDF files) for this specific product.

- [Download TSP168 DES Encryption for Coin Acceptors and Bill Validators](#)
- [Download Master Coin Codes](#)
- [Download ccTalk Part 2 v4.6](#)
- [Download TSP167 DES Encryption for Hoppers v2.1](#)
- [Download ccTalk Part 3 v 4.6](#)
- [Download ccTalk Part 4 v 4.6](#)

Illustration 6: Part 2 of the official documentation where the ccTalk list of commands are explained

3.2.4. Start/End polling for events

To start polling for events you have to call the `StartPoll()` function of the `CoinAcceptor` class (on your object instance of course) and to end the polling you have to call the `EndPoll()` function. In my code example I'm doing this change on the checkbox's `CheckedChanged` event:

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    if (_coinAcceptor == null) return;

    if (!_coinAcceptor.IsInitialized)
        _coinAcceptor.Init(true);

    if (cbPolling.Checked)
        _coinAcceptor.StartPoll();
    else
        _coinAcceptor.EndPoll();
}
```

3.2.5. Start/Stop master inhibition

To start inhibiting all coins no matter of other settings per coin you have to set the `IsInhibiting` property of your `CoinAcceptor` object. In my code example I'm doing this change on the checkbox's `CheckedChaned` event:

```
private void cbInhibit_CheckedChanged(object sender, EventArgs e)
{
    if (_coinAcceptor != null)
        _coinAcceptor.IsInhibiting = cbInhibit.Checked;
}
```

3.2.6. Accepting the coins

As mentioned, in the chapter 3.2.2. Connect to the coin acceptor, we subscribed to the coin acceptor `CoinAccepted` event like this:

```
_coinAcceptor.CoinAccepted += _coinAcceptor_CoinAccepted;
```

where we defined that the function `_coinAcceptor_CoinAccepted` will be handling the event when a coin is accepted. The code listing for this function is below:

```
void _coinAcceptor_CoinAccepted(object sender, CoinAcceptorCoinEventArgs e)
{
    if (InvokeRequired)
    {
        Invoke((EventHandler<CoinAcceptorCoinEventArgs>)_coinAcceptor_CoinAccepted, sender, e);
        return;
    }
    _coinValue += e.CoinValue;
    txtLog.Text += String.Format("Coin accepted: {0} ({1:X2}), path {3}. Now accepted: {2}",
        e.CoinName, e.CoinCode, _coinValue, e.RoutePath) + Environment.NewLine;

    labTotalMoneyIn.Text = _coinValue.ToString();

    // There is simulator of long-working event handler
    Thread.Sleep(1000);
}
```

The `InvokeRequired` checks whether the caller must call an `invoke` method when making method calls to the control because the caller is on a different thread than the one the control was created on. Value of the inserted coin is stored in the `CoinValue` attribute, name of the coin is stored in the `CoinName` attribute, code of the coin is stored in the `CoinCode` attribute and finally the route through which the coin was accepted is stored in the `RoutePath` attribute of the `CoinAcceptorCoinEventArgs` class.