.NET application for the SIMATIC RF350M with WiFi connection

SIMATIC RF350M / .Net / TCP/IP via WiFi

https://support.industry.siemens.com/cs/ww/en/view/109747584

Siemens Industry Online Support

Warranty and Liability

Note

The Application Examples are not binding and do not claim to be complete regarding the circuits shown, equipping and any eventuality. The Application Examples do not represent customer-specific solutions. They are only intended to provide support for typical applications. You are responsible for ensuring that the described products are used correctly. These Application Examples do not relieve you of the responsibility to use safe practices in application, installation, operation and maintenance. When using these Application Examples, you recognize that we cannot be made liable for any damage/claims beyond the liability clause described. We reserve the right to make changes to these Application Examples at any time without prior notice.
If there are any deviations between the recommendations provided in these Application Examples and other Siemens publications − e.g. Catalogs − the contents of the other documents have priority.

We do not accept any liability for the information contained in this document.
Any claims against us – based on whatever legal reason – resulting from the use of the examples, information, programs, engineering and performance data etc., described in this Application Example shall be excluded. Such an exclusion shall not apply in the case of mandatory liability, e.g. under the German Product Liability Act ("Produkthaftungsgesetz"), in case of intent, gross negligence, or injury of life, body or health, guarantee for the quality of a product, fraudulent concealment of a deficiency or breach of a condition which goes to the root of the contract ("wesentliche Vertragspflichten"). The damages for a breach of a substantial contractual obligation are, however, limited to the foreseeable damage, typical for the type of contract, except in the event of intent or gross negligence or injury to life, body or health. The above provisions do not imply a change of the burden of proof to your detriment.
Any form of duplication or distribution of these Application Examples or excerpts hereof is prohibited without the expressed consent of the Siemens AG.

Security information

Siemens provides products and solutions with industrial security functions that support the secure operation of plants, systems, machines and networks.
In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial security concept. Siemens' products and solutions only form one element of such a concept.
Customer is responsible to prevent unauthorized access to its plants, systems, machines and networks. Systems, machines and components should only be connected to the enterprise network or the internet if and to the extent necessary and with appropriate security measures (e.g. use of firewalls and network segmentation) in place.
Additionally, Siemens' guidance on appropriate security measures should be taken into account. For more information about industrial security, please visit http://www.siemens.com/industrialsecurity.

Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends to apply product updates as soon as available and to always use the latest product versions. Use of product versions that are no longer supported, and failure to apply latest updates may increase customer's exposure to cyber threats.
To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed under http://www.siemens.com/industrialsecurity.

Table of Contents

1 Introduction

1.1 Overview

The hand-held terminal SIMATIC RF350M is designed for the flexible use of radio frequency identification (RFID). The hand-held terminal is wireless and provides all standard functions for reading out or writing on the RFID transponder. The device is based on Windows CE and therefore offers the opportunity to create and use individual, user-specific applications for RFID cheaply and in a standardized way.

An SDK is offered to create own applications for the RF350M, which maps all available RFID functions. It also contains an API for the native implementation via C++ and another API for the Microsoft .NET Compact Framework.

This application example shows a customized application of RFID based on the .NET API created with C#, that demonstrates the basic functions of RFID. This includes reading, writing, stocktaking and initialization of transponders as well as the opportunity to switch the read head regarding the available air interfaces RF300 and ISO15693.

In addition, the example shows how to use the integrated WiFi interface of the RF350M to send read out transponder data to a PC or a S7-PLC.

Figure 1-1

Advantages of the application example

This application example offers you the following advantages:

- Expandable Visual Studio project for the SIMATIC RF350M with implemented RFID functions
- Simple and expandable Windows application to receive transponder data from RF350M on a PC/PG
- Simple and expandable TIA Portal project to receive transponder data from RF350M on an S7-PLC.

Assumed knowledge

The following basic user knowledge is required:

- Basics of programming in C#/.NET
- Basics of programming and configuration in the TIA Portal
- Basics of RFID
- Basics of TCP/IP communication

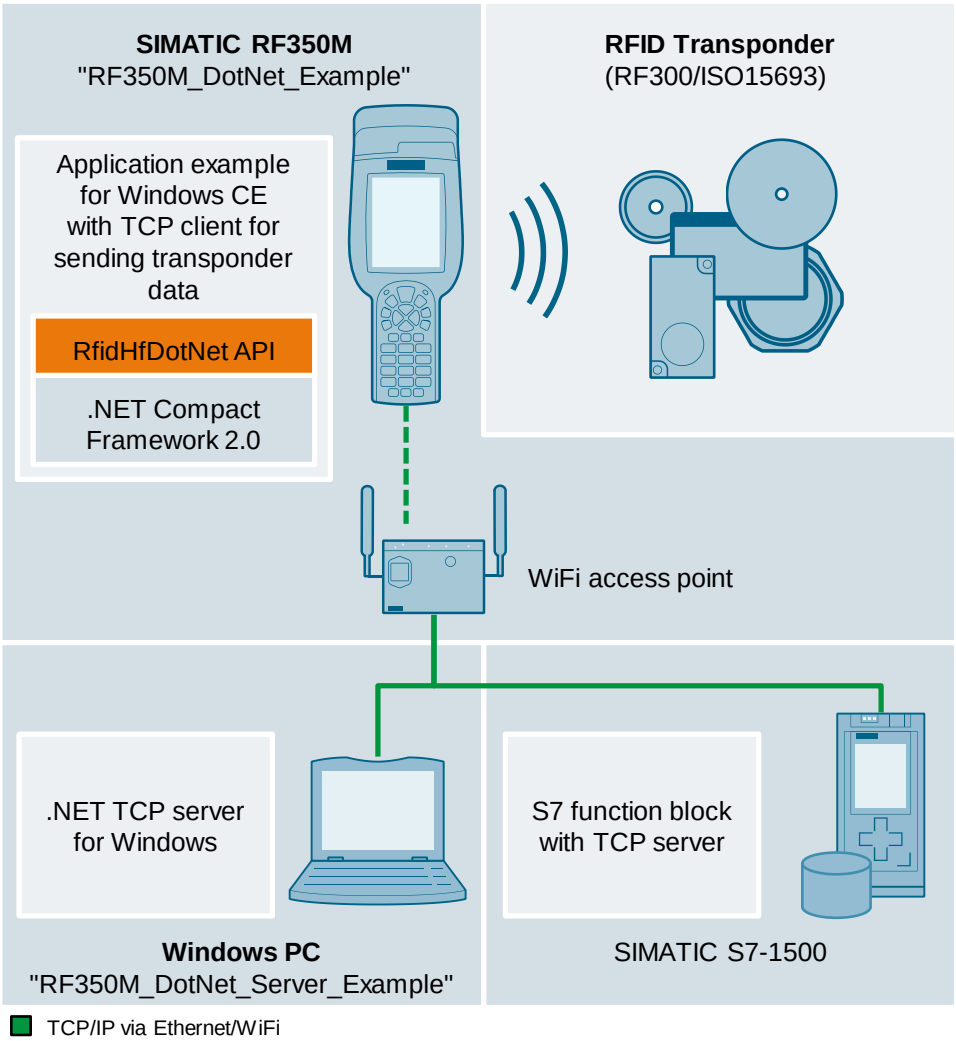
1.2 Mode of operation

Below, you will find an explanation of what components, functions and mode of operations are used in the application example.

General function description

The following figure shows the function principle of this application example and the key components:

Figure 1-2



The sample application for the hand-held terminal SIMATIC RF350M was created with Visual Studio and is based on the .NET Compact Framework 2.0. The "RfidHfDotNet" API provided with this application example was used to implement the RFID functions (read, write, initialize, select RFID protocol) of the read head. The user interface, the program flow and the communication to PC and PLC are implemented with .NET standard components.

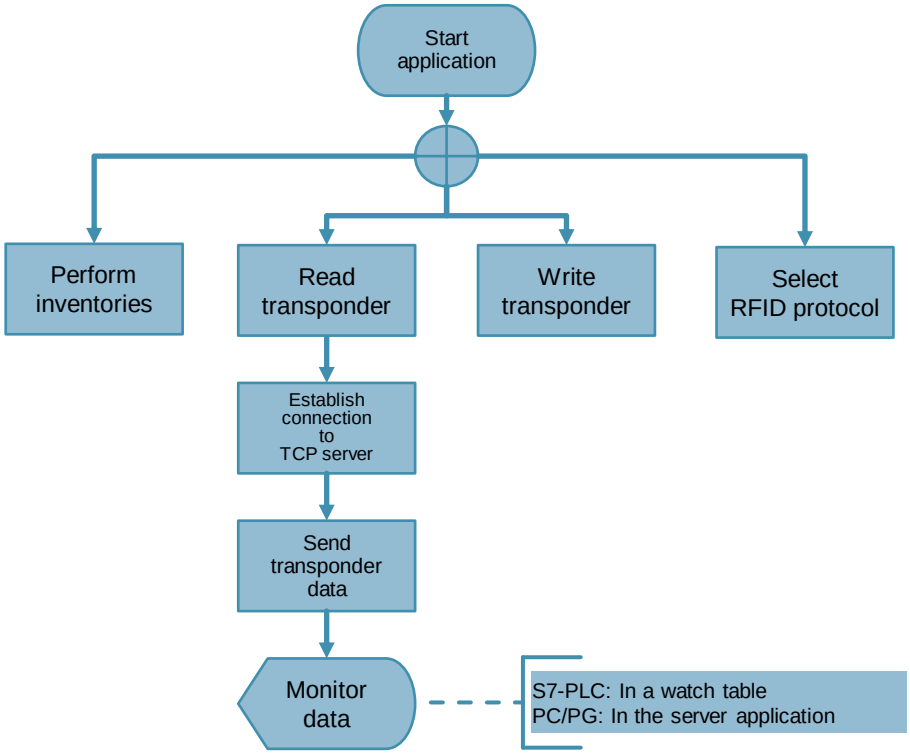
The .NET TCP server for Windows was created with Visual Studio and is exclusively based on .NET standard components.

The TCP server for the S7-PLC is implemented via the SFB "TRCV_C". The S7 project was created with TIA Portal V15.1.

Functional sequence

The following functional sequence applies for this application example:

Figure 1-3



1.3 Components used

This application example was created with the following hardware and software components:

Table 1-1

Component	Number	Article number	Note
SIMATIC RF350M	1	6GT2803-1BA00	-
SIMATIC RF300 Loading/docking station	1	6GT2803-0BM00	-
Windows PC/PG	1	-	You need Windows 7/8/10 for x86/x64
SIMATIC S7-1500 CPU 1513-1 PN/DP	1	6ES7 513-1AL01-0AB0	Alternatively, you can use a CPU from the SIMATIC S7-1200 family (FW > 4.0)
SIMATIC RF300T	n	-	You can use all RF300 transponders
MDS D	n	-	You can use all Moby D transponders

Component	Number	Article number	Note
SCALANCE W7881-PRO	1	6GK5788-1AA60-2AA0	Alternatively, you can use any WiFi Access Point supporting the standard IEEE 802.11 b/g
Microsoft Visual Studio 2005/2008	1	-	-
TIA Portal V15.1	1	6ES7822-1..05-..	-

This application example consists of the following components:

Table 1-2

Component	File name	Note
Documentation	109747584_RF350M_DotNet_DOC_V11_en.docx	This document
RF350M .Net SDK	109747584_RF350M_SDK_V10.zip	-
VS project	109747584_RF350M_DotNet_CODE_V10.zip	-
VS project	109747584_RF350M_DotNet_Server_CODE_V10.zip	-
TIA project	109747584_RF350M_S7_Server_CODE_V11.zip	-

2 Engineering

2.1 Explanation on the API "RfidHfDotNet"

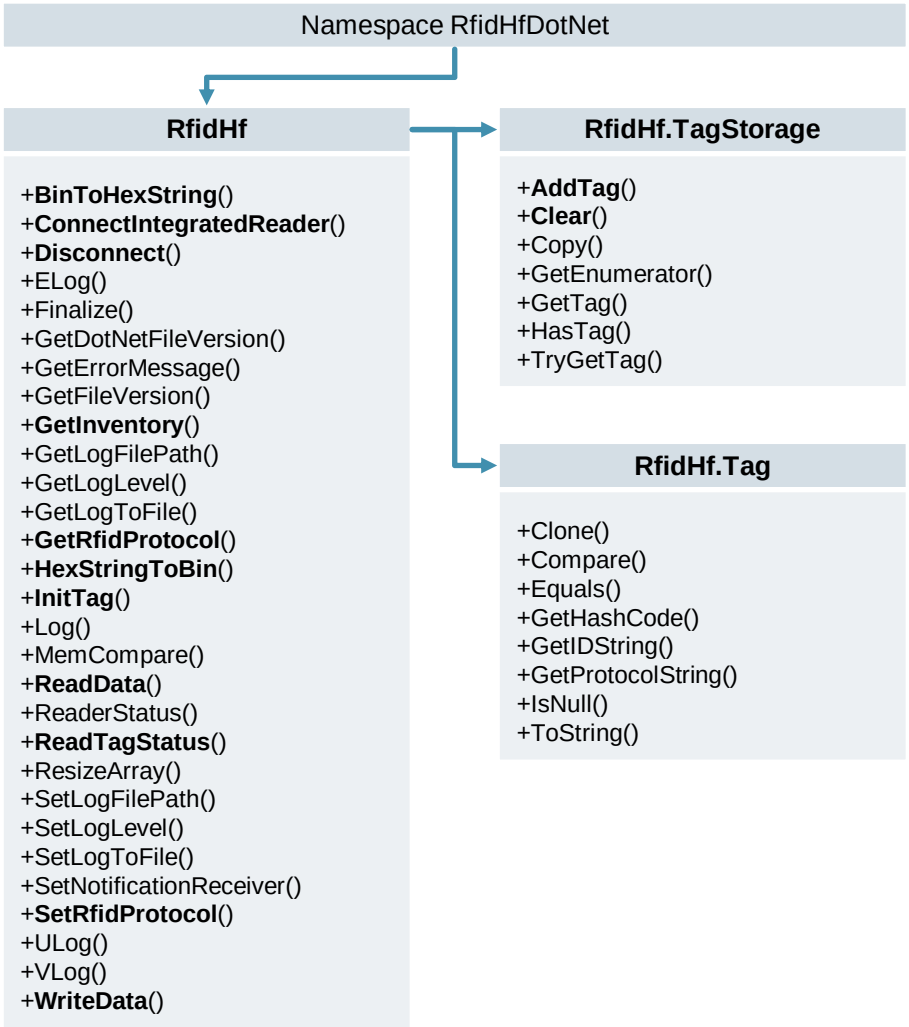
The API RfidHfDotNet (file "RfidHfDotNet.dll") is created with the namespace "RfidHfDotNet". The namespace contains all objects required to implement the RFID functions of the RF300 read head. In addition, the API offers helper methods and structures to simplify programming.

The .NET API is based on the native API "RfidHf" and therefore requires the DLL "RfidHf.dll".

Class diagram

The following class diagram shows the API class "RfidHf" with the most important sub-classes and all available methods:

Figure 2-1



The sample application uses the API classes and methods marked bold.

Note A detailed list of all members and method interfaces of the API class can be found in the CHM help file. You can find the file in the "Docs" directory of the unzipped SDK.

Explanation of the classes within the "RfidHf" API

The following table explains the most important classes within the API:

Table 2-1

Class	Description
RfidHf	This class includes all methods and fields for controlling the RFID read head. It also contains methods for logging the reader and API activities and simple helper methods.
RfidHf.TagStorage	This class implements a memory object for the storage and management of tag objects.
RfidHf.Tag	This class implements a memory object for the storage and management of transponder data. The tag objects can be stored in the TagStorage.

Explanation of the methods within the API classes

The table below explains the methods used in the sample application:

Table 2-2

Method	Description
BinToHexString()	This method converts a byte array into a hex string. Transponder data (e.g. the UID) are usually stored as byte array. Displaying a string on a user interface is easier to realize than a byte array.
ConnectIntegratedReader()	This method connects the API with the RFID read head. Without an established connection, you cannot perform reader operations (e.g. read write).
Disconnect()	This method ends an existing connection to the RFID read head.
GetInventory()	This method sends an inventory command to the RFID read head. An inventory returns the transponders found in the antenna field. Only the UID of the transponder is transmitted. If there is no transponder in the field, you will receive an exception. Note: The RF350M does not support multi tag mode. This means that several transponders must be read in one after the other.
GetRfidProtocol()	This method reads the currently active RFID protocol (RF300 or ISO15693) of the read head.
HexStringToBin()	This method converts a hex string into a byte array. User inputs on the interfaces are often read in as a string. This method converts a string directly to the byte array required for the RFID read head.
InitTag()	This method initializes a transponder in the field with a predefined byte value. The entire user memory of the transponder will be overwritten with the predefined byte value.
ReadData()	This method sends a read command to the RFID read head. You can define a start address and the read length in byte. If there is no transponder in the field, you will receive an exception.

Method	Description
ReadTagStatus()	This method sends a TagStatus command to the RFID read head. to read out the transponder meta data. The data include, for example, memory size, UID, transponder type and error counter. If there is no transponder in the field, you will receive an exception.
SetRfidProtocol()	This method sets the currently active RFID protocol (RF300 or ISO15693) of the read head.
WriteData()	This method sends a write command to the RFID read head. Start address and writing length in byte can be predefined. If there is no transponder in the field, you will receive an exception.
TagStorage.AddTag()	This method adds a transponder (object RfidHf.Tag) to the TagStorage.
TagStorage.Clear()	This method deletes the contents of the TagStorage.

Note An extensive description of the method interfaces can be found in the CHM help file. You can find the file in the "Docs" directory of the unzipped SDK.

2.2 Programming the API functions

This chapter describes the programming interfaces of the .NET-API "RfidHfDotNet" for the integration of the RFID functionalities in individual applications for the SIMATIC RF350M.

Please read chapter 3 Valuable Information before you start programming.

2.2.1 API integration and connection to the RFID read head

In order to the functions and methods of the "RfidHfDotNet" API in your Visual Studio project, you have to integrate the API and connect it to the RFID read head as follows:

1. Load the RF350M SDK from the entry page (\2\) of this application example and unzip the file to a directory of your choice.
2. Create a Windows CE project (CE 5.0 or 6.0) for C# with Visual Studio 2005/2008.
3. Integrate the file "RfidHfDotNet.dll" into your project as a reference. You can find the DLL in the directory "RfidHfApi > dotnet" of the unzipped file.
4. Integrate the file "RfidHf.dll" as an element into your project and change the setting "Copy to output directory" to "Copy when more recent". This step is required as the .NET-API is based on the native API (C# wrapper) and this has also be available in the user directory. You can find the DLL in the directory "RfidHfApi > native" of the unzipped file.
5. Use the Using directive "RfidHfDotNet" in your C# files to connect the namespace of the API with the namespace of your application:

```
using RfidHfDotNet;
```

6. Declare an object instance (C# class) of the API. This makes the API available to the entire class.

```
RfidHf myRfidHfApi;
```

7. Initialize an object of the "RfidHf" class. You can do this in the constructor of a Windows Forms class, for example.

```
myRfidHfApi = new RfidHf();
```

8. Connect the class object with the RFID read head with the method "ConnectIntegratedReader". You can do this in the constructor of a Windows Forms class, for example, to have the API connected to the reader directly after starting the form instance.

```
myRfidHfApi.ConnectIntegratedReader();
```

   We recommend to execute the method in a "try catch" command, to diagnose possible connection problems.

Note One API instance can connect to the RFID read head at a time.

2.2.2 Implementing RFID functions

This chapter describes how to implement the key RFID functions. Before you start, connect the API instance with the RFID read head.

Reading out the active RFID protocol

To read out the RFID protocol (RF300 or ISO15693) active on the RFID read head, proceed as follows:

1. Declare the enumeration "RF300_RFID_PROTOCOL".

```
RfidHf.RF300_RFID_PROTOCOL protocol;
```

2. Call up the method "GetRfidProtocol" of the API instance and assign the return value to the protocol object.

```
protocol = myRfidHfApi.GetRfidProtocol();
```

3. The value of the enumeration outputs the currently active RFID protocol.

```
// Enumeration values
RfidHf.RF300_RFID_PROTOCOL.RFID_PROTOCOL_RF300
RfidHf.RF300_RFID_PROTOCOL.RFID_PROTOCOL_ISO15693
```

Enabling the active RFID protocol

To enable the RFID protocol of the RFID read head, call up the method "SetRfidProtokoll" of the API instance and transfer your desired protocol value of the enumeration "RF300_RFID_PROTOKOLL" and the Boolean value "true" to enable the protocol.

```
// Set RFID protocol for RF300 TAGs
myRfidHfApi.SetRfidProtocol(
RfidHf.RF300_RFID_PROTOCOL.RFID_PROTOCOL_RF300, true);
// Set RFID protocol for ISO15693 TAGs
myRfidHfApi.SetRfidProtocol(
RfidHf.RF300_RFID_PROTOCOL.RFID_PROTOCOL_ISO15693, true);
```

Performing inventories

To perform an inventory, proceed as follows:

1. Declare an undefined array with API data type "RF300_TAG_ID".

```
RfidHf.RF300_TAG_ID[] tagIDs;
```

2. (Optional) Create a "TagStorage" type object to store the found transponder in it.

```
RfidHf.TagStorage tagStorage = new RfidHf.TagStorage();
```

3. Call up the method "GetInventory" of your API instance and assign your return value to the array created beforehand. The read out transponder object will be stored in the array index 0.

```
tagIDs = myRfidHfApi.GetInventory();
```

We recommend to execute the method in a "try catch" command, to be able to handle inventory cycles with transponders detected.

4. (Optional) Store the found transponder with the method "AddTag" in the TagStorage and transfer a new "Tag" type object to the method. TagStorage simplifies the management of several detected transponders.

```
tagStorage.AddTag(new RfidHf.Tag(tagIDs[0]));
```

Repeat steps 3 and 4 in a while loop to inventorize several transponders subsequently.

Reading the transponder status

To read out the transponder status of a transponder, proceed as follows:

1. Create an object of the "RF300_TAG_STATUS" type. The read out transponder status is stored in this object.

```
RfidHf.RF300_TAG_STATUS tagStatus =
     new RfidHf.RF300_TAG_STATUS();
```

2. Call up the method "ReadTagStatus" of the API instance and assign the return value to the created Status object.

```
tagStatus = myRfidHfApi.ReadTagStatus();
```

We recommend to execute the method in a "try catch" command, to handle commands with no transponder.

Reading the user memory

To read out the user memory of a transponder, proceed as follows:

1. Declare and define an Int variable for the start address (in this example address 0) from where you want to read the user memory. Declare and define a second Int variable for the data length to be read (in this example address 100 bytes).

```
int length = 0;
int startAddress = 100;
```

2. Declare a byte array to store the read out data.

```
byte[] data;
```

3. Call up the method "ReadData" of your API instance and transfer an empty object of the "RF300_TAG_ID" type, the start address and the data length. Assign the return value of the method to the declared byte array.

```
data = myRfidHfApi.ReadData(
new RfidHf.RF300_TAG_ID(), startAddress, length);
```

We recommend to execute the method in a "try catch" command, to handle commands with no transponder or incorrect transfer parameters.

Writing on the user memory

To write on the user memory of a transponder, proceed as follows:

1. Declare and define an Int variable for the start address (in this example address 0) from where you want to write on the user memory. Declare and define a second Int variable for the data length to be written (in this example address 100 bytes).

```
Int length = 0;
int startAddress = 100;
```

2. Declare and define a byte array to store the data to be written. If your data are available as a string, you can use the helper method "HexStringToBin" to convert the string to a byte array.

```
byte[] writeData = RfidHf.HexStringToBin(stringDataToWrite);
```

3. Call up the method "WriteData" of your API instance and transfer an empty object of the "RF300_TAG_ID" type, the start address, the data length and the byte array containing the data to be written.

```
myRfidHfApi.WriteData(
new RfidHf.RF300_TAG_ID(), startAddress, length, writeData);
```

We recommend to execute the method in a "try catch" command, to handle commands with no transponder or incorrect transfer parameters.

Initializing the transponder

To initialize a transponder with a specific byte value, proceed as follows:

1. Read out the transponder status and save it in a Status object (RF300_TAG_STATUS).
2. Declare and define a byte (in this example 255 or FF) the value of which you want to use to initialize a transponder.

```
byte initByte = 255;
```

3. Call up the method "InitTag" of the API instance and transfer an empty object of the "RF300_TAG_ID" type, the transponder type (field "bTagType2" in the status object) and the Byte value to be initialized.

```
myRfidHfApi.InitTag(
new RfidHf.RF300_TAG_ID(), tagStatus.bTagType2, initByte);
```

We recommend to execute the method in a "try catch" command, to handle commands with no transponder.

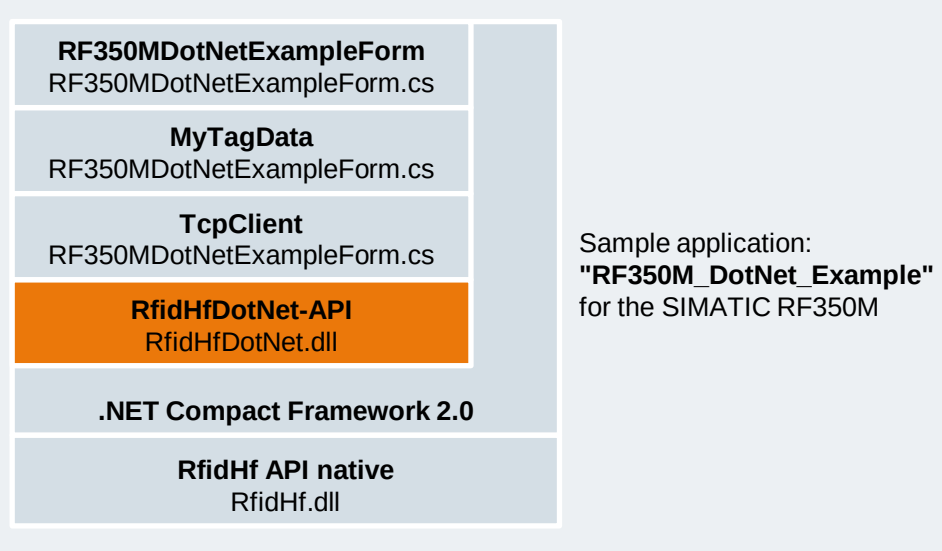
2.3 Explanations on the sample application

This chapter explains how the API functions of the example project have been implemented.

2.3.1 Structural configuration of the application

The following diagram shows the setup of the sample application:

Figure 2-2



The sample application is based on a Windows Forms project from Visual Studio and uses the .NET Compact Framework 2.0, because the hand-held terminal RF350M is based on Windows CE.

The RFID functionality is implemented within the "RF350MDotNetExampleForm" class. So, from this class, the methods of the RfidHfDotNet API are called up, the return values are processed and displayed to the user with different Windows Forms objects. The class "MyTagData" implements a data pair made up of transponder address and data byte and serves as a memory for transponder data. A simple display for the user interface can be created from this structure.

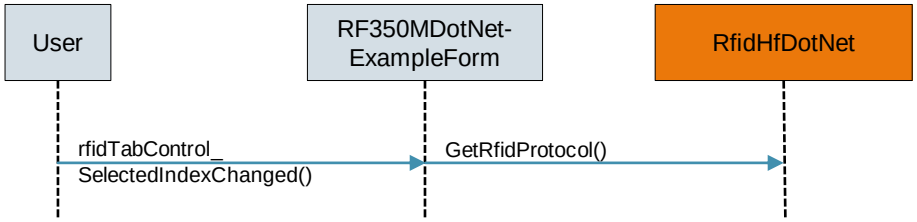
The .NET class "TcpClient" and its methods are used for transmission of the transponder data to a PC/PG or a PLC.

2.3.2 Implementing the API functions in the sample application

The following sequence diagrams show the implementation of key API functions within the sample application. The design of the Forms objects of the user interface will not be explained.

Reading out the active RFID protocol

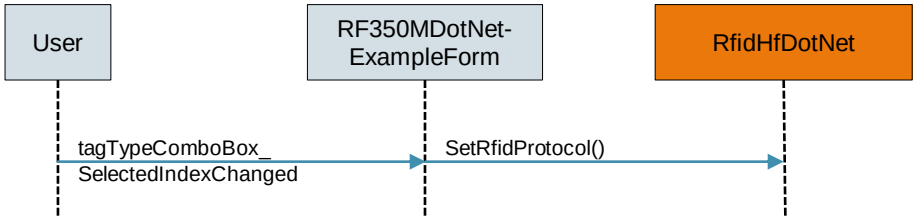
Figure 2-3



When the user goes to the "Settings" register card of the mobile sample application, the event handler "rfidTabControl_SelectedIndexChanged" is called up. Within the event handler, the API method "GetRfidProtocol" is executed, which returns the active RFID protocol of the RFID read head. This information will be logged in the drop-down list showing the active RFID protocol.

Enabling the active RFID protocol

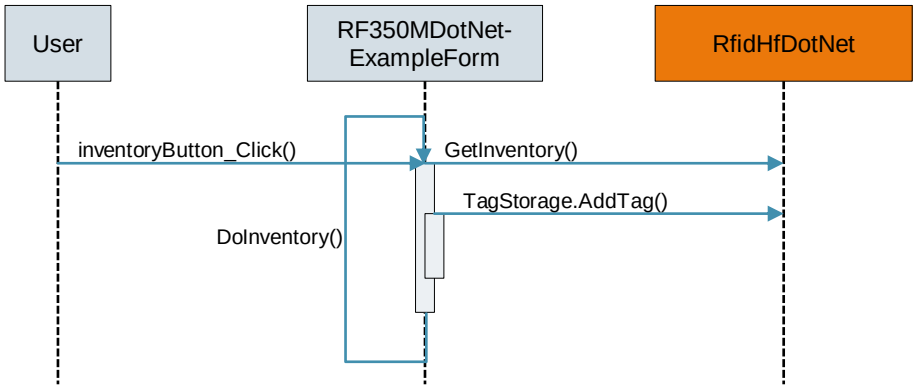
Figure 2-4



By selecting entries in the drop-down list in the "Settings" register card of the mobile application, the event handler "tagTypeComboBox_SelectedIndexChanged" is called up. The API method "SetRfidProtocol" is executed within the event handler. Depending on the selection of the entry in the drop-down list, the RFID protocol to be activated is transferred to the method "SetRfidProtocol".

Performing inventories

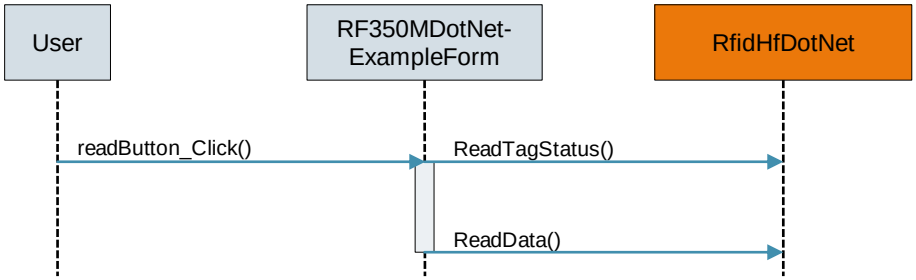
Figure 2-5



When the user clicks on the "Start Inventory" button in the "Inventory" register, the event handler "inventoryButton_Click" is called up. The method "DoInventory" is executed within the event handler. This method is started in a separate thread so it will not block the user interface. Within DoInventory, the API method "GetInventory" is executed in a loop to constantly look for transponders. All found transponders are stored in a TagStorage object via "TagStorage.AddTag". The application will carry out inventories until the user clicks on this button again.

Reading the user memory and transponder status

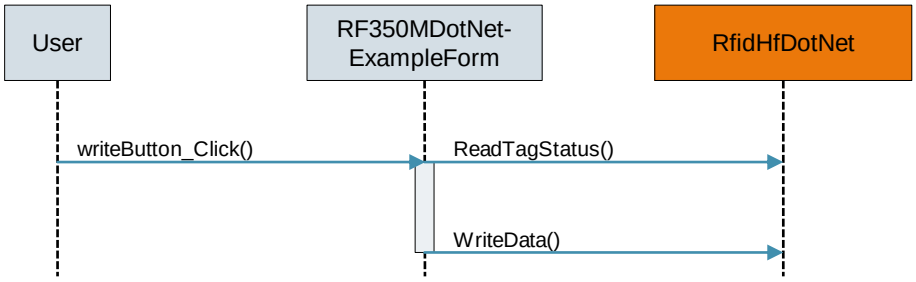
Figure 2-6



When the user clicks on the "Read TAG" button in the "Read/Write" register, the event handler "readButton_Click" is called up. The API method "ReadTagStatus" is executed within the event handler to determine the size of the user memory of the transponder. The size is used to initialize a byte array that can hold the entire user memory of the transponder.
Then the transponder is read out via the API method "ReadData". The return value is stored in the initialized byte array.

Writing on the user memory

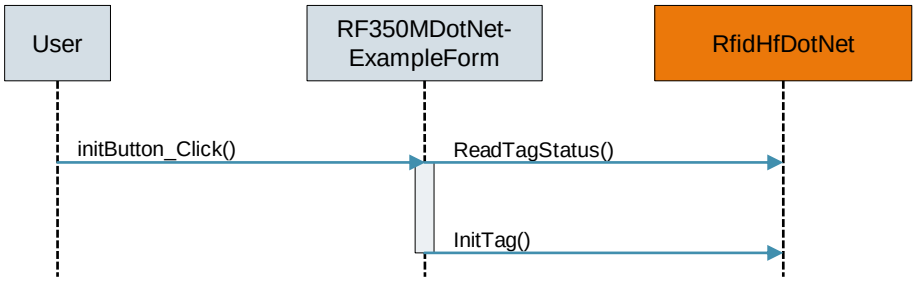
Figure 2-7



When the user clicks on the "Write TAG" button in the "Read/Write" register, the event handler "writeButton_Click" is called up. The API method "ReadTagStatus" is executed within the event handler to determine the size of the user memory of the transponder. The user must know the size of the user memory of the transponder, if he wants to write on its complete user memory.
Then the transponder is written on via the API method "WriteData".

Initializing the transponder

Figure 2-8



When the user clicks on the "Initialize TAG" button in the "Init" register, the event handler "initButton_Click" is called up. The API method "ReadTagStatus" is executed within the event handler to determine the type of the transponder. The transponder type is required for the initialization method.
Then the transponder is initialized via the API method "InitTag".

2.3.3 Implementing the TCP data exchange

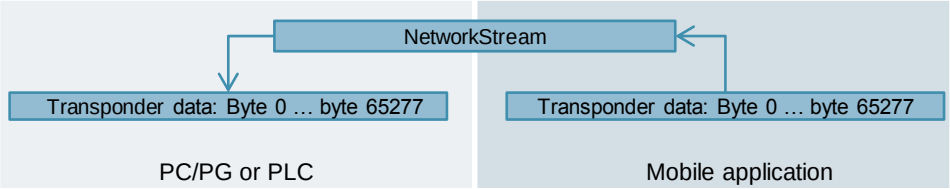
Data transfer between the two applications is always unidirectional and asynchronous from the hand-held terminal to the PC/PG or the PLC. The "TcpClient" .NET class based on Systems.Net.Sockets is used as a software basis for TCP communication.

The transfer is done with data packages with predefined size. A data packages is made up of the biggest possible user memory (65 bytes) of an RF300 transponder. If you have read a smaller amount of data from a transponder, the remaining bytes are filled up with the value "0".

A WiFi connection to the PC/PG or the PLC must be available to perform the data transfer. Please note chapter 2.6.1 Establishing a WiFi connection.

Transmitting the data package

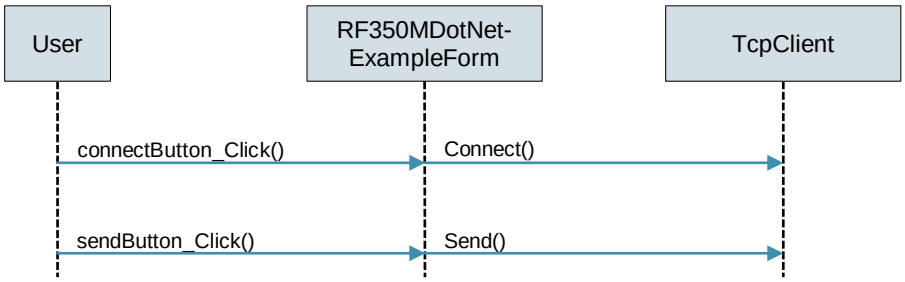
Figure 2-9



A data package is written in the NetworkStream of an existing TCP connection with the .NET method "TcpClient.Send" after the read transponder data have been aggregated in a byte array by the sample application. The PC application or PLC program reads the data package from this NetworkStream.

Sequence diagram for the data transfer steps in the mobile application

Figure 2-10



When the user clicks on the "Connect to Server" button in the "Settings" register, the event handler "connectButton_Click" is called up. The TcpClient method "Connect" is executed within the event handler to establish a connection to the TCP server (PC/PG or PLC). After the connection has been established successfully, the "Transfer" button in the "Read/Write" register is activated. When the user clicks on the "Transfer" button after the connection has been established, the event handler "sendButton_Click" is called up. The data to be sent are aggregated within the event handler and the method "Send" is carried out to send the data.

2.4 Explanations on the TCP server for Windows

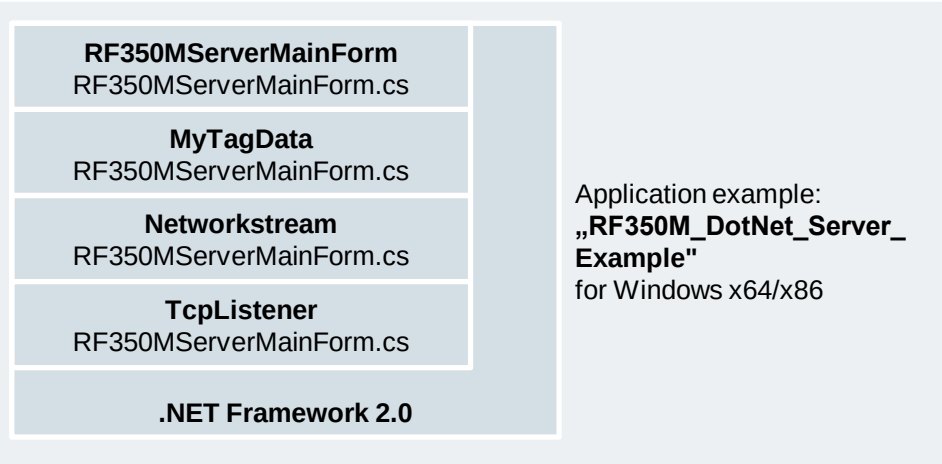
The simple TCP server for Windows was created with Visual Studio 2005 and .Net 2.0.

The application only opens a TCP socket and waits for a TCP connection transmitting user data. The user data are displayed in a DataGrid and can be exported to CSV format.

2.4.1 Structural configuration of the application

The following figure shows the structure of the server application:

Figure 2-11



2.4.2 Mode of operation

This chapter describes how to implement the TCP server. It does not describe all functions of the application.

Enabling the TCP listener

The application enables a TCP listener directly after the form has been loaded (load event):

```
myTcpListener = new TcpListener(IPAddress.Any, 49151);
myTcpListener.Start();

myTcpListener.BeginAcceptTcpClient(
new AsyncCallback(AcceptTcpClient), myTcpListener);
```

The TcpListener listens to port 49151 on all network interfaces.

The asynchronous method BeginAcceptTcpClient() of the listeners is used to process connection requests by TCP clients. The advantage of BeginAcceptTcpClient() is that this method is started in an own thread managed by .NET and thus does not block the UI thread for user inputs. Transfer the TcpListener object to the method BeginAcceptTcpClient().

Accepting connection requests and extracting the NetworkStream

The callback AcceptTcpClient() of the callup BeginAcceptTcpClient() processes the actual connection requests:

```
TcpListener tcpListener = (TcpListener)ar.AsyncState;
TcpClient tcpClient = myTcpListener.EndAcceptTcpClient(ar);
myNetworkStream = tcpClient.GetStream();
```

The TcpListener object is passed on to you via the AsyncState interface. EndAcceptTcpClient() accepts the connection request and creates a TcpClient object delivering the actual network stream with the method GetStream().

Use the network stream to receive user data via TCP (Read).

Reading out the NetworkStream

After a TCP connection has been established and the according network stream is available, read out the stream via the asynchronous method BeginRead() of the network stream:

```
byte[] readBuffer = new byte[1024];
myNetworkStream.BeginRead(
readBuffer, 0, readBuffer.Length, new syncCallback(HandleReadData),
readBuffer);
```

The advantage of BeginRead() is that this method is started in an own thread managed by .NET and thus does not block the UI thread for user inputs. Transfer the readBuffer receive buffer to the method BeginRead(). The callback method HandleReadData() is called up as soon as the receive buffer readBuffer is full. You manage the received data within the callback.

Aggregating and processing received data

The callback HandleReadData makes the received data from the receive buffer available (1024 bytes in this example).

```
byte[] received = (byte[])ar.AsyncState;
int readLength = myNetworkStream.EndRead(ar);
```

The receive buffer is passed on to you via the AsyncState interface. The method EndRead() ends the read process from the network stream. The method returns the received data length.

To receive further data, call up BeginRead() again within the callback:

```
byte[] readBuffer = new byte[1024];
myNetworkStream.BeginRead(
readBuffer, 0, readBuffer.Length, new syncCallback(HandleReadData),
readBuffer);
```

In this example, the receive buffer is copied to a larger buffer which can take the maximum length of the transponder user data. This means that you have to read the network stream until the number of read bytes was aggregated to 65277. Once the value has been reached, you have received a complete frame from the sample application of the RF350M. You can now output the data in a DataGrid, for example, or process otherwise.

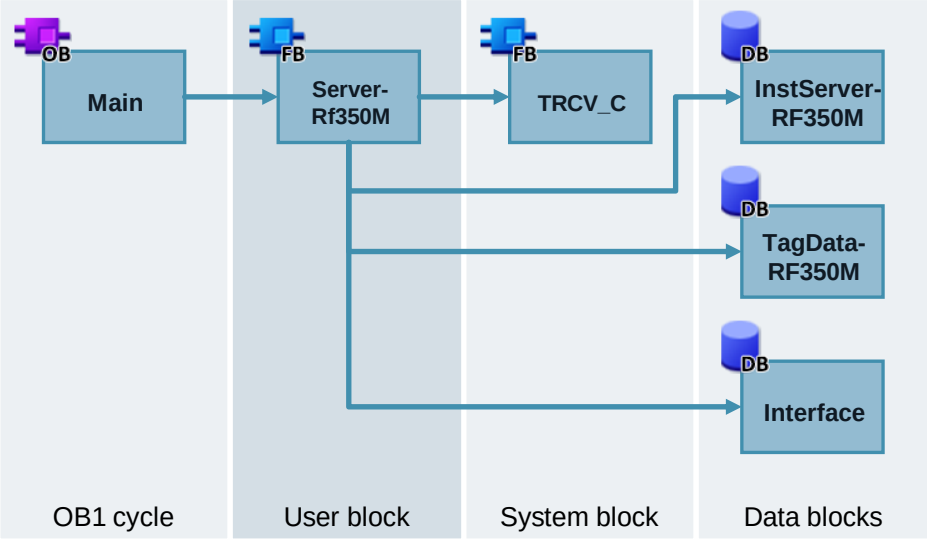
2.5 Explanations on the S7 server block for a PLC

The S7 project was created with TIA Portal V15.1 and is suitable for all S7-1200 and S7-1500 controllers. The project includes an S7 user block which receives transponder data from the SIMATIC RF350M and stores them in a data block. From there, the transponder data can be used for further processing.

2.5.1 Structure of the S7 user program

The following figure shows the call hierarchy of the S7 user program.

Figure 2-12



2.5.2 Mode of operation

The S7 user block "ServerRF350M" opens a TCP socket for the S7-PLC via the internally called system block "TRCV_C". The mobile application of the RF350M uses TCP/IP to connect to this socket.

After the RF350M application has established a connection to the PLC, the S7 user block is waiting for user data.

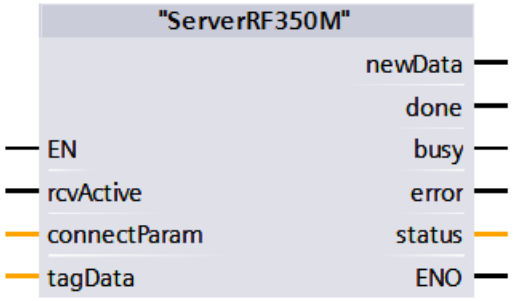
As soon as a complete user data package (65277 bytes) has been received by the sample application og zhr RF350M, a bit is set signaling the user that new data have been received. In the same cycle, the received data are copied in the data block "TagDataRF350M", where they are available for further processing.

All relevant controllable tags of the sample project are stored in the data block "Interface". This includes the connection parameters of the TCP block.

Block interface in the TIA Portal

The following figure shows the block interface of the S7 user block "ServerRF350M":

Figure 2-13



Input/output parameters

Table 2-3

Name	Data type	Description
rcvActive	Bool	FALSE: TCP server inactive. TRUE: TCP server is active and ready to receive.
connectParam	TCON_IP_V4	Reference to the connection parameters for the block TRCV_C.
tagData	Array[0..65276] of Byte	Reference to the receive area for transponder data.

Output parameters

Table 2-4

Name	Data type	Description
newData	Bool	TRUE: New transponder data have been received. When this output is set to TRUE, the received transponder data are copied into the "tagData" memory area.
done	Bool	TRUE: Connection successfully established/disconnected or new data received.
busy	Bool	TRUE: Connection is established or disconnected or block is waiting for data.
error	Bool	TRUE: Error when establishing or disconnecting connection or when receiving data. Refer to the output "status" for more detailed information.
status	Word	Current block status. Refer the online help on the TRCV_C block for more detailed information on the status.

Note The output parameters "newData" and "done" are each only pending for one cycle.

2.6 Commissioning

The following chapters describe how to commission the applications of this sample.

Note In this example, we assume that the hand-held terminal, the PLC, the PC and the WiFi Access Point are in the same IP subnet.

The following subnet is used: 192.168.0.0/24

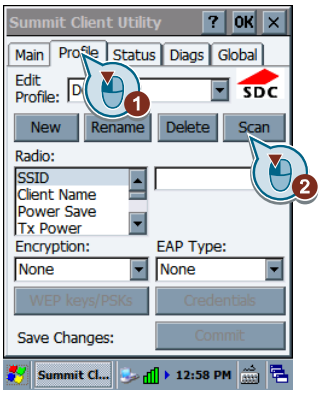
2.6.1 Establishing a WiFi connection

The steps required to connect the hand-held terminal to an access point via WiFi are explained below. At this Access Point, either a PC/PG must be connected to the TCP server or a PLC must be connected with the server block.

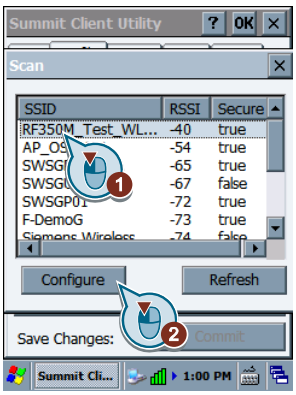
1. Set up your access point. The hand-held terminal has WiFi support according to the IEEE 802.11 b/g standards.

Note In order to follow the startup description of this chapter, make sure to assign a visible SSID to your WiFi. In addition, provide the network with a WPA2 key to ensure basic security.

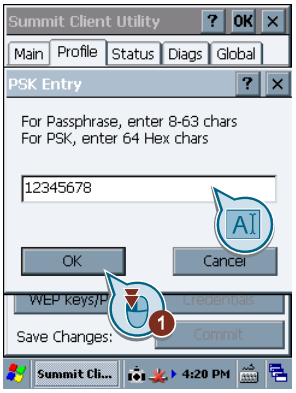
2. Start the preinstalled "Summit Client Utility" application on the mobile hand-held reader. You can find this application in "Start" > "Programs" > "Summit" > "SCU" on the hand-held terminal.
3. Go to the "Profile" tab and then click "Scan" to search for WiFi networks within range.



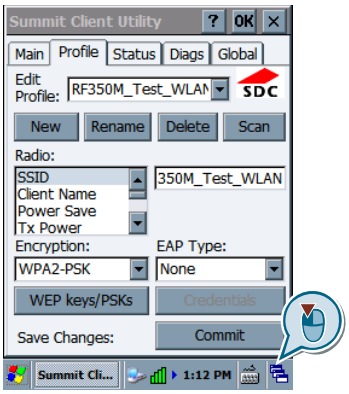
4. Select the SSID you have configured in your access point and click "Configure".



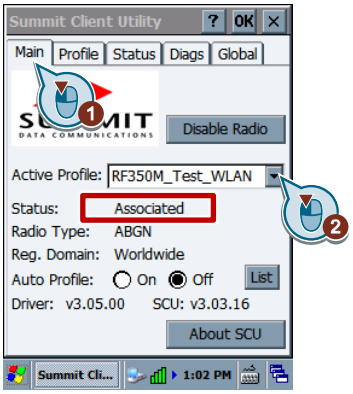
5. Enter the WiFi key you have defined and select "OK" to confirm.



6. Select "Commit" to confirm.



7. Go to the "Main" tab and in the "Active Profile" drop-down menu, select the connection profile you have just created.
   If you have entered all the data correctly, the hand-held terminal will connect to the WiFi network. Now "Status" displays "Associated".



2.6.2 Upload and start of the RFID application on the hand-held terminal

How to copy the mobile sample application to the hand-held terminal and how to start it is explained below:

8. Follow the instructions of "Establishing a connection to the SIMATIC RF350M" in chapter 3 Valuable Information.
9. Download the "109747584_RF350M_DotNet_CODE_V10.zip" file from the entry page of this application example (\1\).
10. Unpack the archive to a directory of your choice.
11. Go to directory "RF350M_DotNet_Example > Application" of the unpacked project.
12. Copy the contained files "RF350M_DotNet_Example.exe", "RfidHF.dll" and "RfidHfDotNet.dll" to the clipboard.
13. Start the Windows Mobile Device Center.
14. Hold the mouse over the button "Connect without setting up your device" and click on "File Management > Browse the contents of your device".
15. Open the directory "Flash" by double-clicking it.
16. Paste the data from the clipboard to the folder.
17. On the hand-held terminal, go to "My device > Flash" and start the application "RF350M_DotNet_Example.exe".

2.6.3 Commissioning the server application for Windows

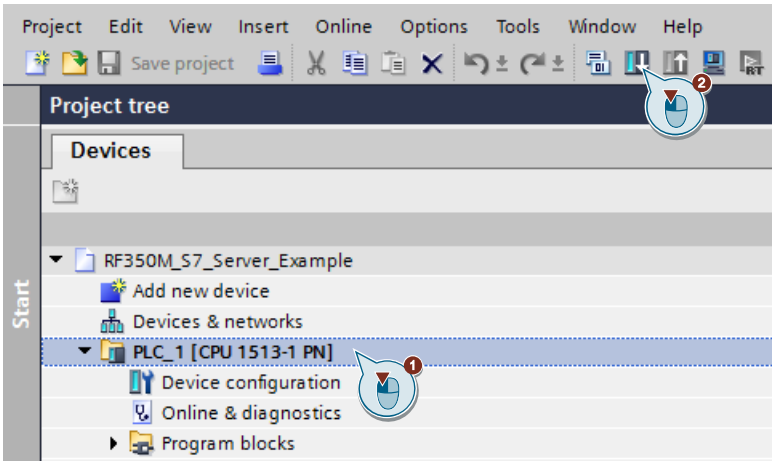
How to start the server application for Windows is explained below:

1. Download the "109747584_RF350M_DotNet_Server_CODE_V10.zip" file from the entry page of this application example (\1\).
2. Unpack the archive to a directory of your choice.
3. Go to directory "RF350M_DotNet_Server_Example > Application" of the unpacked project.
4. Start the TCP server with a double-click on the application "RF350M_DotNet_Server_Example.exe".

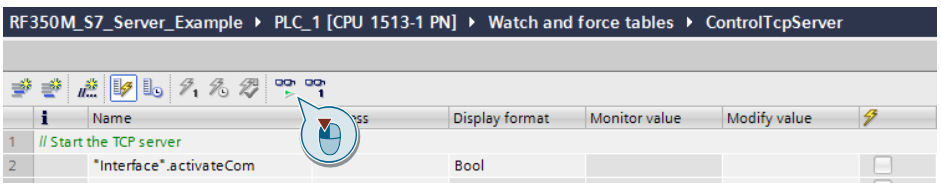
2.6.4 Commissioning the S7 server block

The following table explains how to commission the STEP 7 V15.1 project:

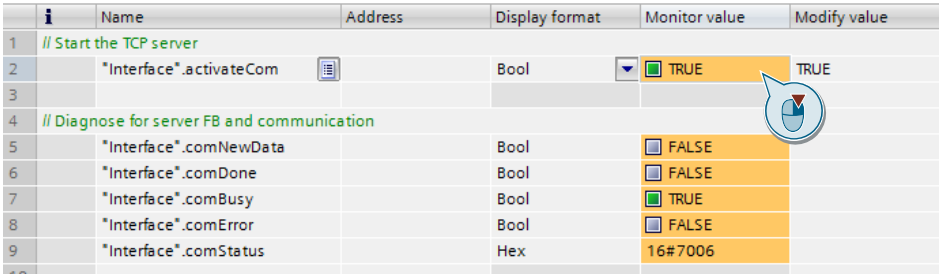
1. Download the zip file "109747584_RF350M_S7_Server_CODE_V11.zip" from the entry page of this application example (\1\).
2. Unpack the archive to a directory of your choice.
3. Open the project by double-clicking on "RF350M_S7_Server_Example_V15.1.ap15_1".
4. Select the CPU in the project tree and click on "Download to device".



5. In the TIA Portal project, go to "PLC_1 > Watch and force tables" and open the watch table "ControlTcpServer" with a double-click.
6. Click on the "Monitor all" button.



7. Set the tag "Interface.activateCom" to TRUE, to activate the TCP server.



		Name	Address	Display format	Monitor value	Modify value
1	// Start the TCP server					
2		"Interface".activateCom		Bool	TRUE	TRUE
3						
4	// Diagnose for server FB and communication					
5		"Interface".comNewData		Bool	FALSE	
6		"Interface".comDone		Bool	FALSE	
7		"Interface".comBusy		Bool	TRUE	
8		"Interface".comError		Bool	FALSE	
9		"Interface".comStatus		Hex	16#7006	

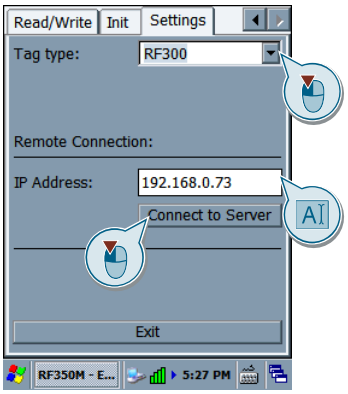
8. The tags diagnosing the S7 block can be found under "Diagnose for server FB and communication".

2.7 Operating the Application Example

The following chapters describe how to operate the applications of this sample.

2.7.1 Selecting the RFID protocol and connecting to the TCP server

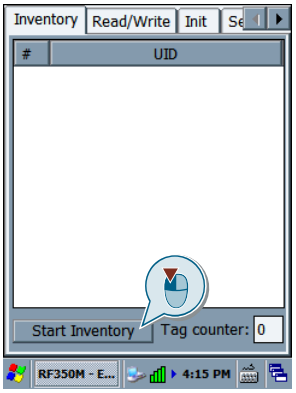
1. Once the application is started, go to the "Settings" task card.
2. Select in the "Tag type" drop-down list, if you want to read a RF300 or ISO transponder.
   Enter the IP address of the TCP server (server application or S7 server block) in the field "IP Address".
   Click on "Connect to Server" to establish a connection.



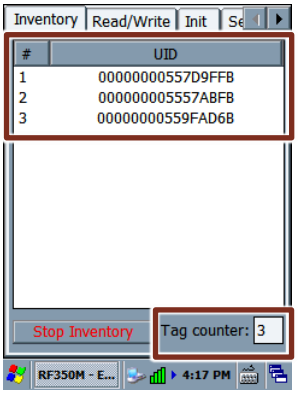
3. Once the connection is successfully established, the text on the button changes to "Disconnect".
4. Click "Disconnect" to disconnect the connection.

2.7.2 Performing inventories

1. Go to the "Inventory" task card.
2. Click the "Start Inventory" button.



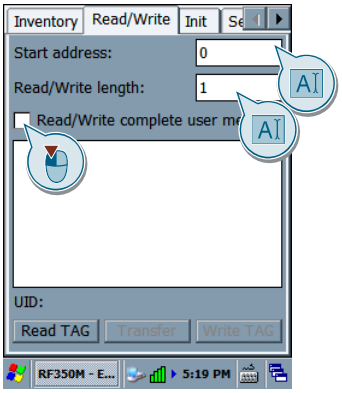
3. The button changes its font color to read and now shows the text "Stop inventory". Now hold several transponders before the read head of the hand-held terminal. The list now shows all found transponder IDs. In addition, the number of found transponders is shown in the "Tag counter" field.



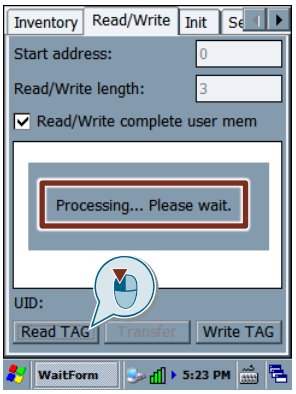
4. Click on "Stop Inventory" to stop the process.

2.7.3 Reading the transponder

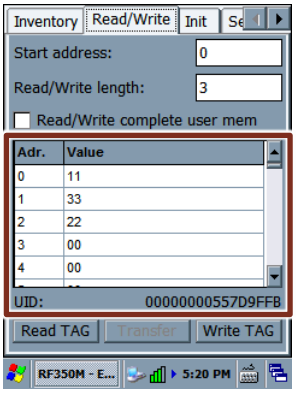
1. Go to the "Read/Write" task card.
2. In the field "Start address", enter the address of the user memory of the transponder you want to read. Then enter the length (in bytes) you want to read into the field "Read/Write length".
   As an alternative, you can select the "Read/Write complete user mem" control box to read the complete transponder.



3. Hold a transponder to the read head and then click on "Read TAG" to read the transponder according to your predefinitions. While the data are read and processed, the note "Processing… Please wait" is displayed.

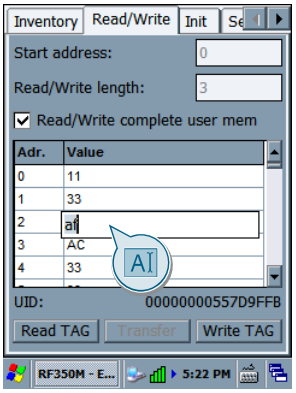


4. After the user memory of the transponder has been read, you can see the transponder addresses and the related user data (bytes in HEX format) in the list. You can also see the transponder ID in the field "UID".

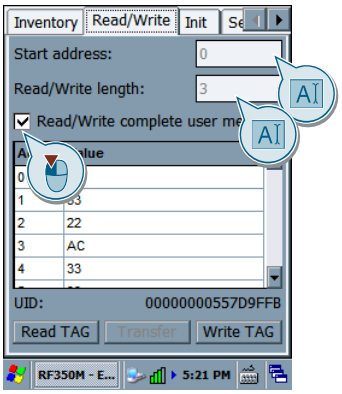


2.7.4 Writing on the transponder

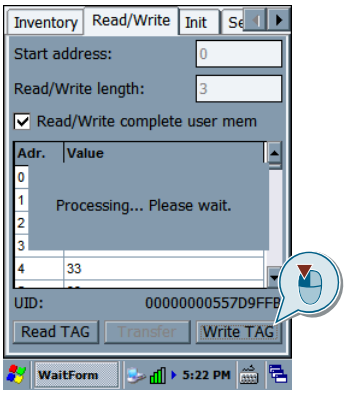
1. Go to the "Read/Write" task card.
2. Read the transponder before you write on it.
3. Click in any data field of the list to change the value of the byte. Enter the desired value in the hand-held terminal with the key fields. Only HEX values can be used: "00" to "FF".



4. In the field "Start address", enter the address of the user memory of the transponder you want to write on. Then enter the length (in bytes) you want to write from the specified address into the field "Read/Write length".
   As an alternative, you can select the "Read/Write complete user mem" control box to write on the complete transponder.

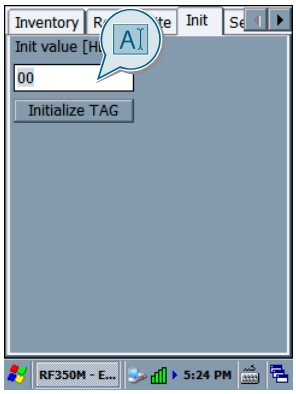


5. Hold a transponder to the read head and then click on "Write TAG" to write on the transponder according to your predefinitions. While the data written, the note "Processing… Please wait" is displayed.

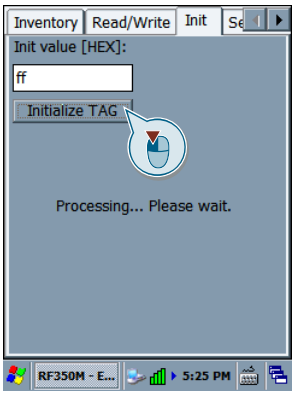


2.7.5 Initializing the transponder

1. Go to the "Init" task card.
2. In the field "Init value", enter the byte value for initializing the entire transponder.

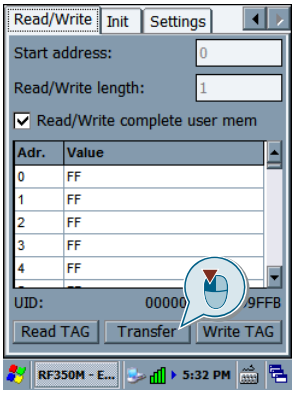


3. The click "Initialize TAG" to start the process. While the data written, the note "Processing… Please wait" is displayed.



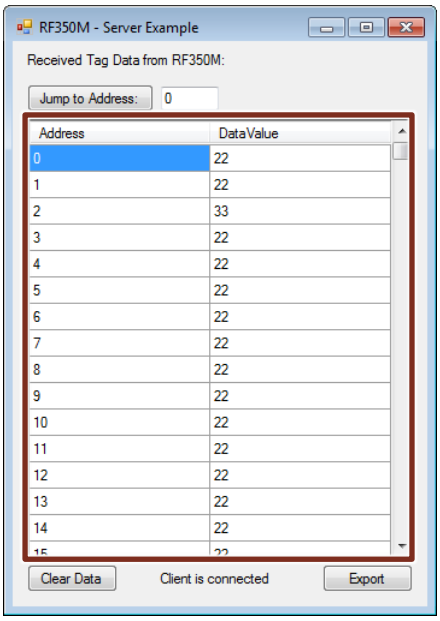
2.7.6 Sending transponder data to the TCP server

1. To start the TCP server of the server application for Windows, follow the instructions given in chapter 2.6.3 Commissioning the server application for Windows.
   As an alternative, to start the TCP server of the S7-PLC, follow the instructions given in chapter 2.6.4 Commissioning the S7 server block.
2. Connect to a TCP server as described in chapter 2.7.1 Selecting the RFID protocol and connecting to the TCP server.
3. To read a transponder, follow the instructions given in chapter 2.7.3 Reading the transponder.
4. Click the "Transfer" button to send the read transponder data to the TCP server (S7-PLC or Windows PC/PG). The button is enabled after a connection to a server has been established.

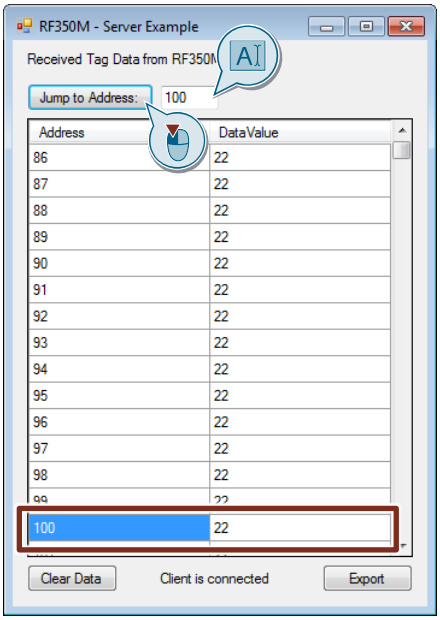


Monitoring the received transponder data with the server application for Windows

1. The server application automatically receives transponder data from the mobile RFID application of the SIMATIC RF350M. The received data are shown in a list.



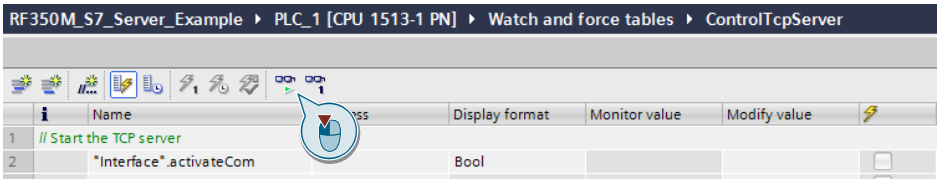
2. Enter any transponder address in the input field beside the button "Jump to Address" and then click on the button to jump to the desired address on the memory.



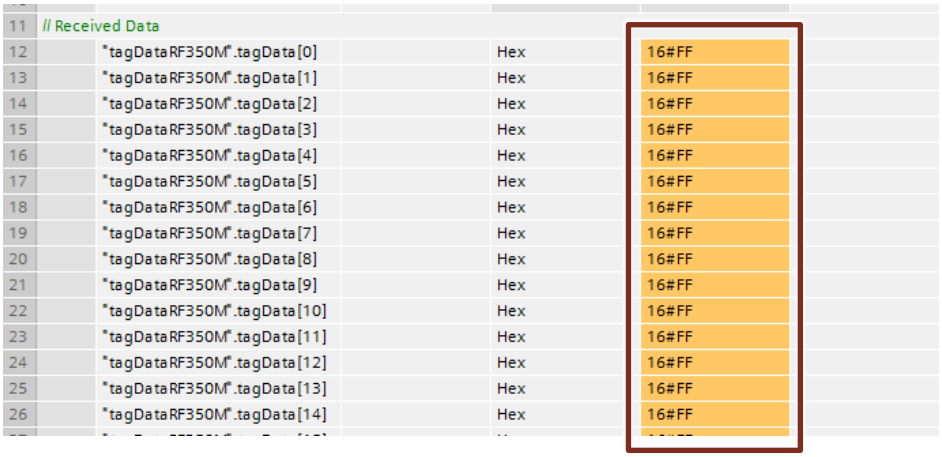
3. Click on the "Export" button to export the received data in CSV format. Follow the instructions in the pop-up dialog.
4. Click on the "Clear Data" button to delete the received data.

Monitoring the received transponder data in the S7-PLC

1. In the TIA Portal project, go to "PLC_1 > Watch and force tables" and open the watch table "ControlTcpServer" with a double-click.
2. Click on the "Monitor all" button.



3. The S7 block automatically receives transponder data from the mobile RFID application of the SIMATIC RF350M. Part of the received data is shown in the watch table under "Received Data".



3 Valuable Information

This chapter describes how to set up the SDK and a connection from Visual Studio to the hand-held terminal.

Installing SDK

The SDK contains the debugging environment for the SIMATIC RF350M. Install the SDK as follows:

1. Load the RF350M SDK from the entry page (\2\) of this application example and unzip the file to a directory of your choice.
2. Open the file "Windows CE 6.0 SDK" and double-click on "Merlin_SDK_2011.msi".
3. Follow the instructions of the installer to install the SDK.

Establishing a connection to the SIMATIC RF350M

The steps required to connect Visual Studio or the PC/PG and the hand-held terminal are explained below. This connection is required for remote debugging of your application.

1. Connect the power supply unit (included in delivery) to the docking station.
2. Plug the hand-held terminal into the docking station. The terminal starts up automatically.
3. Wait until Windows CE has loaded on the terminal.
4. Connect the USB cable (included in delivery) with a USB port of your PC/PG.
5. Start Windows Mobile Device Center (active sync).

Note Windows Mobile Device Center opens automatically when you connect the SIMATIC RF350M to your PC/PG using USB. When connected for the first time, Windows Mobile Device Center will be installed. Follow the instructions on your screen.

6. The docking station with plugged in hand-held terminal will actively establish an active-sync connection to the Windows Mobile device center. Once the device has been connected, the device status "Connected" is shown in the device center:

7. If the SDK is installed and the active-sync connection has been established, Visual Studio will also be connected with the hand-held terminal.

4 Annex

4.1 Service and support

Industry Online Support

Do you have any questions or need support?

Siemens Industry Online Support offers access to our entire service and support know-how as well as to our services.

Siemens Industry Online Support is the central address for information on our products, solutions and services.

Product information, manuals, downloads, FAQs and application examples – all information is accessible with just a few mouse clicks at:
https://support.industry.siemens.com

Technical Support

Siemens Industry's Technical Support offers quick and competent support regarding all technical queries with numerous tailor-made offers
– from basic support right up to individual support contracts.

Please address your requests to the Technical Support via the web form:
www.siemens.com/industry/supportrequest

Service offer

Our service offer comprises, among other things, the following services:

- Product Training
- Plant Data Services
- Spare Parts Services
- Repair Services
- On Site and Maintenance Services
- Retrofit & Modernization Services
- Service Programs and Agreements

Detailed information on our service offer is available in the Service Catalog:
https://support.industry.siemens.com/cs/sc

Industry Online Support app

Thanks to the "Siemens Industry Online Support" app, you will get optimum support even when you are on the move. The app is available for Apple iOS, Android and Windows Phone:
https://support.industry.siemens.com/cs/ww/en/sc/2067

4.2 Links and literature

Table 4-1

No.	Topic
\1\	Siemens Industry Online Support https://support.industry.siemens.com
\2\	Link to the entry page of the application example https://support.industry.siemens.com/cs/ww/en/view/109747584

4.3 Change documentation

Table 4-2

Version	Date	Modifications
V1.0	09/2017	First version
V1.1	08/2019	Update to TIA Portal V15.1