AN3227
Application note

Developing an application with the STARTKIT-M24LR-A

Introduction

This application note explains how to develop a Visual Basic or C/C++ application code to drive STMicroelectronics STARTKIT-M24LR-A starter kit from a host computer. The STARTKIT-M24LR-A is a ISO 15693 reader designed to operate with the M24LR64-R dual interface EEPROM. It is connected to the host USB port and can be used either as a I^2C reader connected to the transponders through an I^2C bus cable or as an RF reader when using the integrated RF antenna.

Figure 1. STARTKIT-M24LR-A

Reference documents

- M24LR64-R datasheet
- *"M24LR64-R tool kit user guide"* (UM0853)
- *"Configuring your ISO 15693 reader to support the M24LR64-R"* application note (AN3163)

Contents

List of tables

List of figures

1 Description

1.1 M24LR64-R

The M24LR64-R is a Dual interface EEPROM which can be accessed either through an I^2C serial bus or a contactless interface using the ISO 15693 RFID protocol.

To easily control the M24LR64-R RF and I^2C channels, ST offers several tools among which is the STARTKIT-M24LR-A.

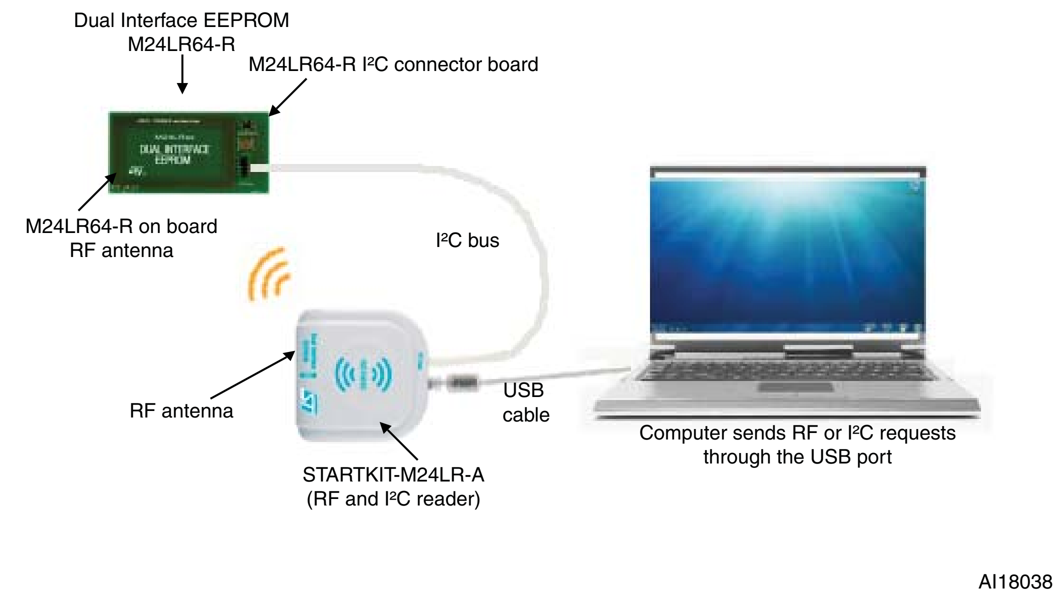
Refer to the product datasheet and to application note AN3163 “Configuring your ISO 15693 reader to support the M24LR64-R” for more in-depth information on the M24LR64-R and for explanations on the RF and I^2C communication protocols. Both documents are available on http://www.st.com.

Application note AN3227 helps software engineers using and including the software delivered with the STARTKIT-M24LR-A in their own application. Some examples are offered showing how to send an RF request in Visual Basic and C/C++. These examples also show how to manage the I^2C protocol.

1.2 STARTKIT-M24LR-A

The STARTKIT-M24LR-A reader is connected to the host-computer USB port. It manages RF ISO 15693 commands (High-level Inventory and Transparent commands) and I^2C commands between the reader and M24LR64-R-based transponders (see *Figure 2: STARTKIT-M24LR-A application schematics*).

Figure 2. STARTKIT-M24LR-A application schematics



- RF ISO 15693 commands

  RF ISO 15693 commands are sent by the host computer to the transponders via the reader RF interface. Two types of commands are available:

  – High-level Inventory command (see *Section 4.2: RF ISO 15693 High-level Inventory command*)

    The host sends an already formatted Inventory request to the transponders via the reader RF interface. This command launches an anticollision sequence to identify all the transponders present in the RF field and sends back the UID information to the host. Due to reader limitations, a maximum of two transponders can be detected by an Inventory request.

  – Transparent commands (see *Section 4.3: RF ISO 15693 Transparent commands*).

    The host can also send RF Transparent commands to transponders via the reader RF interface. The Transparent commands send single or multiple frames compliant with the ISO 15693 protocol. Refer to the M24LR64-R datasheet for a detailed description of the available Dual interface EEPROM command.

    The transponders answers are sent back to the computer.

- I^2C commands (see *Section 4.4: I2C commands*).

  I^2C commands are sent by the host computer to the M24LR64-R via the reader I^2C bus.

The following sections explain how to detect the STARTKIT-M24LR-A reader and open the communication prior to sending RF or I^2C commands to the M24LR64-R transponders. They then describe in details how to send RF commands to transponders and manage I^2C communication.

2 Installation requirements

Communication with the STARTKIT-M24LR-A is based on the HID USB class. To be able to communicate with the reader, the *HIDdll.dll* file must be installed in your computer. This is mandatory for sending any RF or I^2C request using the STARTKIT-M24LR-A.

Depending on whether you use Visual Basic or C/C++, other .bas or .h files must be used for correct function declaration. These installations are outlined below.

2.1 HIDdll.dll installation

The *HIDdll.dll* contains all the functions allowing to drive the STARTKIT-M24LR-A.

Copy the *HIDdll.dll* which is delivery with the AN3227.zip available at http://www.st.com/dualeeprom to the Windows system folder of your computer (C:/Windows/System32). If you have installed the *M24LRxx_Application_Software*, then the *HIDdll.dll* file is already present in your Windows system folder.

2.2 Visual Basic project requirements

To create a Visual Basic project, the *HIDdll.bas* header file must be inserted in your source code. It references all the STARTKIT-M24LR-A functions declared in the *HIDdll.dll*.

Insert the *HIDdll.bas* into your Visual Basic project. *HIDdll.bas* makes the link between *HIDdll.dll* and the source code. *HIDdll.bas* contains all the functions declarations and descriptions for whichever communication mode you choose: RF ISO 15693 or I^2C mode.

HIDdll.bas can *be* found in the AN3227.zip available at http://www.st.com/dualeeprom.

Below is an example of the command needing to be declared in your Visual Basic header file, when operating in RF ISO15693 mode or I^2C mode:

```
Public Declare Function API_USBAll Lib "HIDdll.DLL" (ByVal
commHandle As Long, ByVal cmdSize As Long,  ByRef cmd As Byte, ByRef
returnlen As Byte,  ByRef pBuffer As Byte)  As Long
```

2.3 C/C++ project requirements

To create a C/C++ project, the *HIDdll.h* header file and the *HIDdll.lib* library, are required. *HIDdll.h* and *HIDdll.lib* files allow linking your source code to the *HIDdll.dll* file. *HIDdll.h* can be found in the source code example AN3227.zip available at http://www.st.com/dualeeprom.

In the source code, declare the header files as follows:

```
#include "HIDdll.h"
```

Below is an example of the command needing to be declared in your C/C++ header file, when operating in RF ISO15693 mode or I^2C mode:

```
'extern "C" __declspec(dllexport) int __stdcall API_USBAll(HANDLE
commHandle,int cmdSize, unsigned char *cmd, unsigned char
*returnlen,unsigned char *pbuffer);
```

3 STARTKIT-M24LR-A detection

To send an I^2C or an RF command to the STARTKIT-M24LR-A reader, the computer must first detect the reader. Once it is detected, a handle is randomly assigned to the reader.

This section presents all the available functions for performing reader detection. Visual Basic and C/C++ source code examples are also provided.

3.1 Reader detection functions

The following functions are included in the *HIDdll.dll* file and must be called to be able to use the reader:

- **`API_USBGetConnectedDeviceNum`**
  This function detects any connected STARTKIT-M24LR-A reader, and sends back the number of readers connected to the USB ports of your computer.
- **`API_USBOpenWithNum`**
  This function returns the handle of the STARTKIT-M24LR-A identified by the index number. This reader must have been previously detected by the **`API_USBGetConnectedDeviceNum`** function.
- **`API_USBCloseComm`**
  This function closes USB communication for the STARTKIT-M24LR-A identified by its handle.

3.1.1 API_USBGetConnectedDeviceNum

API_USBGetConnectedDeviceNum Visual Basic prototype

Table 7 illustrates the `API_USBGetConnectedDeviceNum` Visual Basic function.

Table 1. `API_USBGetConnectedDeviceNum` Visual Basic function

Function description	
Prototype	`Public Declare Function API_USBGetConnectedDeviceNum Lib "HIDdll.DLL" () As Long`
Parameters	None
Returned value	Number of connected USB devices minus 1

API_USBGetConnectedDeviceNum C/C++ prototype

Table 2 illustrates the `API_USBGetConnectedDeviceNum` C/C++ function.

Table 2. `API_USBGetConnectedDeviceNum` C/C++ function

Function description	
Prototype	`extern "C" __declspec(dllexport) int __stdcall API_USBGetConnectedDeviceNum(void);`
Parameters	None
Returned value	Number of connected USB devices minus 1

3.1.2 API_USBOpenWithNum

API_USBOpenWithNum Visual Basic prototype

Table 3 illustrates the `API_USBOpenWithNum` Visual Basic function.

Table 3. `API_USBOpenWithNum` Visual Basic function

Function description	
Prototype	`Public Declare Function API_USBOpenWithNum Lib "HIDdll.DLL" (ByRef hcomm As Long, ByVal deviceIndex As Long, ByVal numInputBuffers As Byte) As Long`
Parameters	**hcomm**: STARTKIT-M24LR-A handle returned by the function **deviceIndex**: device index (0x00 for one connected device) **numinputBuffers**: 0x40
Returned value	Error code

API_USBOpenWithNum C/C++ prototype

Table 4 illustrates the `API_USBOpenWithNum` C/C++ function.

Table 4. `API_USBOpenWithNum` C/C++ function

Function description	
Prototype	`extern "C" __declspec(dllexport) int __stdcall API_USBOpenWithNum(HANDLE * hcomm,int deviceIndex,WORD numInputBuffers);`
Parameters	**hcomm**: STARTKIT-M24LR-A handle returned by the function **deviceIndex**: device index (0x00 for one connected device) **numinputBuffers**: 0x40
Returned value	Error code

3.1.3 API_USBCloseComm

API_USBCloseComm Visual Basic prototype

Table 5 illustrates the `API_USBCloseComm` Visual Basic function.

Table 5. `API_USBCloseComm` Visual Basic function

Function description	
Prototype	`Public Declare Function API_USBCloseComm Lib "HIDdll.DLL" (ByVal commHandle As Long) As Boolean`
Parameters	**commHandle**: handle of the device to be closed
Returned value	Error code

API_USBCloseComm C/C++ prototype

Table 5 illustrates the `API_USBCloseComm` C/C++ function.

Table 6. `API_USBCloseComm` C/C++ function

Function description	
Prototype	`extern "C" __declspec(dllexport) BOOL __stdcall API_USBCloseComm(HANDLE commHandle);`
Parameters	**commHandle**: handle of the device to be closed
Returned value	Error code

3.2 Visual Basic source code example

Below is an example of reader detection Visual Basic code:

```
Public Function Detect_STARTKIT_M24LR_A() As Boolean
Dim lngNbStartKit As Long
Dim lngStatus As Long
Dim i As Long
Dim deviceIndex As Integer
Dim numInputBuffersInt As Integer
Dim strDataMsg As String
'STARTKIT-M24LR-A
Dim lngReqDataLen As Long
Dim abytReqData (0 To 63) As Byte
Dim abytAnswerDataLen (0 To 63) As Byte
Dim abytAnswerData (0 To 63) As Byte
Dim strRequestData As String
Dim strAnswerData As String
'RETURN THE NUMBER OF STARTKIT-M24LR-A DEVICES CONNECTED
lngNbStartKit = API_USBGetConnectedDeviceNum()
'ONE STARTKIT-M24LR-A CONNECTED
If (lngNbStartKit > 0) Then
 deviceIndex = 0
    numInputBuffersInt = 64
'------> OPEN CONNECTION WITH STARTKIT-M24LR-A
    '------> GET USB HANDLE (hcomm_public)
    lngStatus = API_USBOpenWithNum(hcomm_public, _
                               deviceIndex, _
                               numInputBuffersInt)
'------> GET FIRMWARE VERSION OF STARTKIT-M24LR-A
    strReqData = "86"
    lngReqDataLen = Len(strReqData) / 2 'lenght of request
'format request as Array of bytes for API_USBALL use
   For i = 0 To lngReqDataLen - 1
        abytReqData (i) = CByte("&h" & Mid(strReqData, _
                               (i * 2) + 1, 2))
   Next i
 lngStatus = API_USBAll(hcomm_public, _
                          lngReqDataLen,
```

```
                          abytReqData(0), _
                          abytAnswerDataLen(0),
                          abytAnswerData(0))
'Analyse STARTKIT-M24LR-A answer
   For i = 1 To abytSTARTKITanswerSize(0)
        strAnswerData = strAnswerData & _
                        Chr(abytSTARTKITanswer(i - 1))
   Next i
'DISPLAY STARTKIT-M24LR-A ANSWER = GET INFO
   txtDetectResult.Text = strAnswerData
Else
   'No STARTKIT-M24LR-A CONNECTED
End If
End Function

'STARTKIT-M24LR-A KIT CLOSE USB COMMUNICATION
Private Sub Form_terminate()
Dim booAnswer As Boolean
Dim lngCloseComm As Long
booAnswer = API_USBCloseComm(hcomm_public)
End Sub
```

3.3 C/C++ source code example

Below is an example of reader detection C/C++ code:

```
int detect_STARTKIT_M24LR_A ()
{
  int iNbStartKit;
  int deviceIndex;
  unsigned short numInputBuffersInt;
  int istatus;
  int entry3;

  /* RETURN THE NUMBER OF STARTKIT-M24LR-A DEVICES CONNECTED   */
  iNbStartKit = API_USBGetConnectedDeviceNum();

  /* ONE STARTKIT-M24LR-A CONNECTED */
  if (iNbStartKit > 0)
  {
     deviceIndex = 0;
     numInputBuffersInt = 64;
/* ------> OPEN CONNECTION AND GET USB HANDLE */
     istatus = API_USBOpenWithNum(&hcomm_public,
                                  deviceIndex,
                                  numInputBuffersInt);

     if (istatus == 0)
     {
    /* USB connection OK */
    /* hcomm_public is handle for STARTKIT-M24LR-A functions */
    return 1;
```

```
        }
        else
        {
        /* USB connection HS */
       return 0;
        }
   }
   /* NO STARTKIT-M24LR-A READER CONNECTED */
   else
   {
        return 0;
   }
}

/* CLOSE STARTKIT-M24LR-A COMMUNICATION */
int Close_STARTKIT_M24LR_A_communication (void)
{

    API_USBCloseComm(hcomm_public);
    hcomm_public=NULL;
    return 1;
}
```

4 USB API_USBALL function

The API_USBALL is an high-level function sent by the host to the STARTKIT-M24LR-A through the USB interface. It allows sending any RF ISO 15693 or I^2C commands and retrieving the M24LR64-R answers.

Note: The RF ISO 15693 commands (High-level Inventory or Transparent commands) are sent via the reader integrated RF antenna, while the I^2C commands are sent via the I^2C cable (SCL and SDA).

4.1 API_USBALL generic description

Below are the generic description of `API_USBAll` function in Visual Basic and C/C++.

4.1.1 Visual Basic API_USBALL prototype

Table 7 illustrates the `API_USBAll` Visual Basic function.

Table 7. `API_USBAll` Visual Basic function

Function description	
Prototype	`Public Declare Function API_USBAll Lib "HIDdll.DLL" (ByVal commHandle As Long, ByVal cmdSize As Long, ByRef cmd As Byte, ByRef returnlen As Byte, ByRef pBuffer As Byte) As Long`
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: size of the **cmd** expressed in bytes **cmd**: communication method between the reader and the M24LR24-R (0xB0 for RF ISO 15693 and 0xB8 for I^2C), plus the request sent to the M24LR64-R **returnlen**: pbuffer size **pbuffer**: M24LR64-R answer, if any. *Note: The **returnlen** and **pbuffer** parameters are filled in by the reader after USB request management.*
Returned value	Error code

4.1.2 C/C++ source API_USBALL prototype

Table 8 described the `API_USBAll` C/C++ function.

Table 8. API_USBAll C/C++ function

Function description	
Prototype	`extern "C" __declspec(dllexport) int __stdcall API_USBAll (HANDLE commHandle, int cmdSize, unsigned char *cmd, unsigned char *returnlen, unsigned char *pbuffer);`
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: size of the **cmd** expressed in bytes **cmd**: communication method between the reader and the M24LR24-R (0xB0 for RF ISO 15693 and 0xB8 for I^2C), plus the request sent to the M24LR64-R **returnlen**: pbuffer size **pbuffer**: M24LR64-R answer, if any. *Note: The **returnlen** and **pbuffer** parameters are filled in by the reader after USB request management.*
Returned value	Error code

4.2 RF ISO 15693 High-level Inventory command

To issue a RF ISO 15693 High-level Inventory command, send the `API_USBAll` function with the cmd parameter containing the ISO15693 mode header ‘0xB0’, followed by the integrated Inventory request command ‘0x01 06 00 00’. Refer to *Table 9* for a detailed description of the corresponding API_USBAll function, and to *Section 4.2.1* and *Section 4.2.2* for code examples in Visual Basic and C/C++.

Note: Due to reader limitations, a maximum of only two transponders can be detected with one Inventory request.

Table 9. RF ISO15693 high-level Inventory command

API_USBAll parameters	
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: 0x05 **cmd**: 0xB0 (RF mode) + 0x01 06 00 00 (high-level Inventory request) **returnlen**: pbuffer size **pbuffer**: The reader sends back the UID of the detected transponders. If no transponder is detected, the ‘0x83’ error code is returned.
Returned value	Error code

4.2.1 Visual Basic source code example

Below is an example of Visual Basic Inventory code:

```
Private Function Cmd_Inventory_STARTKIT_M24LR_A () As Boolean
Dim i, j As Integer
Dim lngStatus As Long
'request
Dim strReqData As String
Dim abytReqData(0 To 63) As Byte
Dim lngReqDataLen As Long
'STARTKIT-M24LR-A answer variables
Dim strAnswerData As String
Dim abytAnswerData(0 To 63) As Byte
Dim abytAnswerDataLen(0 To 63) As Byte
Dim lngTranspNumber As Long
Dim strtransponder As String
Dim strDataMsg As String
 'STARTKIT-M24LR-A Inventory ISO15693 request:
  `"B0 01 06 00 00"
strReqData = "B001060000"
  lngReqDataLen = Len(strReqData) \ 2
 For i = 0 To lngReqDataLen - 1
      abytReqData(i) = CByte("&h" & Mid(strReqData, _
                       (i * 2) + 1, 2))
  Next i
' STARTKIT-M24LR-A INVENTORY request in Host mode
  lngStatus = API_USBAll(hcomm_public, _
                         lngReqDataLen,
                         abytReqData(0), _
                         abytAnswerDataLen(0),
                         abytAnswerData(0))
'Format Answer as tring
  strAnswerData = ""
  For i = 0 To abytAnswerDataLen(0) - 1
     strAnswerData = strAnswerData & i2hhh(CLng(abytAnswerData(i)),
2)
  Next i
' RF INVENTORY REQUEST RESULT
  ' if(lngStatus = 0) then PASS else FAIL
  ' if (abytAnswerDataLen(0) = 0) then No transponder answer
  ' else abytAnswerData() contains the transponder(s) answer(s)
End Function
```

4.2.2 C/C++ source code example

Below is an example of C/C++ Inventory code:

```
/* INVENTORY COMMAND + ANTI COLLISION PROCESS */
int Cmd_Inventory_STARTKIT_M24LR_A (void)
{
byte sReqData[1024];
  byte sRspData[1024];
  int   iReqLen;
  unsigned char iRspLen;
  int iResult;
  int iResult2;
  int entry3;
  int i;
 /* Inventory request ISO15693 command : */
  /* B0 + 01060000 */
 sReqData[0] = 0x05; /* Nb of bytes of all bytes request for
API_USBAll function */
  sReqData[1] = 0xB0;
  sReqData[2] = 0x01;
  sReqData[3] = 0x06;
  sReqData[4] = 0x00;
  sReqData[5] = 0x00;
/* STARTKIT-M24LR-A Write Single Block request */
  iResult = API_USBAll(hcomm_public,
                       sReqData[0],sReqData+1,
                       &iRspLen,sRspData);
printf("\n    --> answer  : ");
  if (iResult == 1)
     printf("No tag answer received");
  else
     for (i=0; i<iRspLen; i++)
            printf("%.2X",sRspData[i]);
/* RF INVENTORY REQUEST RESULT */
  /* if(iResult == 0) PASS else FAIL */
  /* if (iRspLen == 0) No transponder answer */
  /* else sRspData contains the transponder(s) answer(s) */
if (iResult != 0)
  {
    /* No Tag detected in the Antenna Field */
  }
  else
  {
    /* 1 or more transponders are in Antenna Field */
  }
return iResult;
}
```

4.3 RF ISO 15693 Transparent commands

To issue RF ISO 15693 Transparent commands, send the `API_USBAll` function with the cmd parameter containing the ISO15693 mode header ‘0xB0’ and ‘0xFF’ to select the Transparent mode, followed by the M24LR64-R request frame.

All the requests described in the M24LR64-R datasheet can be issued by using this method. *Section 4.3.1* and *Section 4.3.2* illustrate two examples of requests, the Read single block and write single block request, which allow to read and write a single block of Dual interface memory.

4.3.1 Read single block command

Table 10 describes the parameters that must be passed to the API_USBAll function, in order to read a block of 4 bytes from the M24LR64-R memory.

Following are the corresponding code examples in Visual Basic and C/C++.

Table 10. Read single block

API_USBAll parameters	
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: 0x07 **cmd** 0xB0FF: RF Transparent command 0x04: Number of bytes of the request. 0x0A: RF protocol request flag 0x20: Read single block request Address (2 bytes): Block number where the reader reads the data. The target block number is obtained by inverting the 2-byte code. **returnlen**: pbuffer size **pbuffer**: The reader sends back either the read data in pbuffer or the error flag if the read operation has failed (refer to the datasheet for the list of all possible error flags).
Returned value	Error code
Example	cmd = B0 FF 04 0A 20 FA 01 reads a 4-byte data block from address 0x01FA.

Visual Basic source code example

Read 4 bytes from address

```
Private Function RFReadsingleBlock() As Boolean
Dim i As Integer
Dim lngStatus As Long
'request
Dim strReqData As String
Dim abytReqData(0 To 63) As Byte
Dim lngReqDataLen As Long
'anwer
```

```
Dim strAnswerData As String
Dim abytAnswerData(0 To 63) As Byte
Dim abytAnswerDataLen(0 To 63) As Byte
'----RF READ SINGLE BLOCK request
' STARTKIT-M24LR-A reader parameters :    0xB0FF
' Number of bytes of request :      0x04
' Flag                              0x0A
' RF Read command :                 0x20
' Address (send FA01 = LSB BYTE first): 0x01FA
strReqData = "B0FF040A20FA01"
lngReqDataLen = Len(strReqData) / 2
For i = 0 To lngReqDataLen - 1
     abytReqData(i) = CByte("&h" & Mid(strReqData, (i * 2) + 1, 2))
Next i
lngStatus = API_USBAll(hcomm_public, _
                       lngReqDataLen,
                       abytReqData(0), _
                       abytAnswerDataLen(0),
                       abytAnswerData(0))
For i = 0 To abytAnswerDataLen(0) - 1
  strAnswerData = strAnswerData & _
                  i2hhh(CLng(abytAnswerData(i)), 2)
Next i
' RF READ REQUEST RESULT
' if(lngStatus = 0) then PASS else FAIL
' if (lngRespDataLen = 0) then No transponder answer
' else strRespData contains the transponder answer
End Function
```

C/C++ source code example

```
/* READ SINGLE BLOCK COMMAND SENT IN Transparent mode */
/* TRANSPONDER's ANSWER IS READ LIKE ANSWER */
int RFReadsingleBlock (void)
 byte sReqData[1024];
  byte sRspData[1024];
  int   iReqLen;
  unsigned char iRspLen;
  int iResult;
  int entry3;
  int i;
/* RF READ SINGLE BLOCK request format */
  /* STARTKIT-M24LR-A reader param : B0FF        */
  /* STARTKIT-M24LR-A reader param Nb Bytes : 04 */
  /* Flag RF request : 0A */
  /* RF Read single block command : 20  */
  /* Address : 01FA  (send FA01 = LSB BYTE first) */
/* request = B0FF + 04 + 0A20FA01        */
sReqData[0] = 0x07;/* Number of bytes of all bytes request for
API_USBAll function */
  sReqData[1] = 0xB0;
  sReqData[2] = 0xFF;
  sReqData[3] = 0x04;
```

```
  sReqData[4] = 0x0A;
  sReqData[5] = 0x20;
  sReqData[6] = 0xFA;
  sReqData[7] = 0x01;
/* STARTKIT-M24LR-A Write Single Block request */
  iResult = API_USBAll(hcomm_public,
                      sReqData[0],sReqData+1,
                      &iRspLen,sRspData);
printf("\n    --> answer  : ");
  if (iResult == 1)
   printf("No tag answer received");
  else
   for (i=0; i<iRspLen; i++)
           printf("%.2X",sRspData[i]);
/* RF REQUEST RESULT */
  /* if(iResult == 0) PASS else FAIL */
  /* if (iRspLen == 0) No transponder answer */
  /* else sRspData contains the transponder answer */
return iResult;
}
```

4.3.2 Write single block command

Table 11 describes the parameters that must be passed to the API_USBAll function, in order to write a 4-byte data block to the M24LR64-R memory starting.

Following are code examples in Visual Basic and C/C++.

Table 11. Write single block

API_USBAll parameters	
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: 0x0B **cmd** 0xB0FF: RF Transparent command 0x08: Number of bytes of the request. 0x0A: RF protocol request flag 0x21: Write single block request Address (2 bytes): Block number where the reader reads the data. The target block number is obtained by inverting the 2-byte code. 4 bytes: bytes to be programmed in the M24LR64-R memory. **returnlen**: pbuffer size **pbuffer**: The reader sends back the request status (refer to the datasheet for the list of all possible values).
Returned value	Error code
Example	cmd = B0 FF 08 0A 21 FA 01 AA BB CC DD writes a block containing ‘0xAA BB CC DD’ to address 0x01FA.

Visual Basic source code example

```
Private Function WriteSingleBlockRF() As Boolean
Dim i As Integer
Dim lngStatus As Long
'request
Dim strReqData As String
Dim abytReqData(0 To 63) As Byte
Dim lngReqDataLen As Long
'STARTKIT-M24LR-A answer
Dim strAnswerData As String
Dim abytAnswerData(0 To 63) As Byte
Dim abytAnswerDataLen(0 To 63) As Byte
'----RF WRITE SINGLE BLOCK request
' STARTKIT-M24LR-A reader parameters :    0xB0FF
' Number of bytes of request :      0x08
' Flag                              0x0A
' RF Write command :                0x21
' Address (send FA01 = LSB BYTE first): 0x01FA
' Data 0xAABBCCDD :                 AABBCCDD
strReqData = "B0FF080A21FA01AABBCCDD"
lngReqDataLen = Len(strReqData) / 2
For i = 0 To lngReqDataLen - 1
  abytReqData(i) = CByte("&h" & Mid(strReqData, (i * 2) + 1, 2))
Next i
lngStatus = API_USBAll(hcomm_public, _
                       lngReqDataLen,
                       abytReqData(0), _
                       abytAnswerDataLen(0),
                       abytAnswerData(0))
For i = 0 To abytAnswerDataLen(0) - 1
    strAnswerData = strAnswerData & _
                    i2hhh(CLng(abytAnswerData(i)), 2)
Next i
' RF WRITE REQUEST RESULT
  ' if(lngStatus = 0) then PASS else FAIL
  ' if (lngRespDataLen = 0) then No transponder answer
  ' else strRespData contains the transponder answer
End Function
```

C/C++ source code example

```
/* WRITE SINGLE BLOCK COMMAND SENT IN Transparent mode */
/* TRANSPONDER's ANSWER IS WRITE LIKE ANSWER (ANSWER AFTER TPROG
TIME) */
int WriteSingleBlockRF (void)
{
 byte sReqData[1024];
  byte sRspData[1024];
  int   iReqLen;
  unsigned char iRspLen;
  int iResult;
  int entry3;
```

```
  int i;
/* RF WRITE SINGLE BLOCK request format */
  /* STARTKIT-M24LR-A reader param : B0FF        */
  /* STARTKIT-M24LR-A reader param Nb Bytes : 08 */
  /* Flag RF request : 0A */
  /* RF Write single block command : 21  */
  /* Address : 01FA  (send FA01 = LSB BYTE first) */
  /* Data : AABBCCDD                */
/* request = B0FF + 08 + 0A21FA01AABBCCDD */
sReqData[0] = 0x0B;/* Number of bytes of all bytes request for
API_USBAll function */
  sReqData[1] = 0xB0;
  sReqData[2] = 0xFF;
  sReqData[3] = 0x08;
  sReqData[4] = 0x0A;
  sReqData[5] = 0x21;
  sReqData[6] = 0xFA; /* address inverted */
  sReqData[7] = 0x01;
  sReqData[8] = 0xAA; /* data */
  sReqData[9] = 0xBB;
  sReqData[10] = 0xCC;
  sReqData[11] = 0xDD;

  /* STARTKIT-M24LR-A Write Single Block request */
  iResult = API_USBAll(hcomm_public,
                       sReqData[0],sReqData+1,
                       &iRspLen,sRspData);

  printf("\n    --> answer  : ");
  if (iResult != 0)
   printf("No tag answer received");
  else
   for (i=0; i<iRspLen; i++)
           printf("%.2X",sRspData[i]);
 /* RF REQUEST RESULT */
  /* if(iResult == 0) PASS else FAIL */
  /* if (iRspLen == 0) No transponder answer */
  /* else sRspData contains the transponder answer */

  return iResult;
}
```

4.4 I²C commands

To issue I²C commands, send the `API_USBAll` function with the cmd parameter containing the I²C mode header ‘0xB8’, followed by the M24LR64-R request frame. All the I²C requests described in the M24LR64-R datasheet can be issued by using this method.

The following commands are available:

- I²C start (see *Section 4.4.1*)
- I²C send byte (see *Section 4.4.2*)
- I²C read byte (see *Section 4.4.3*)
- I²C send Ack (see *Section 4.4.4*)
- I²C send NoAck (see *Section 4.4.5*)
- I²C read ACK (see *Section 4.4.6*)
- I²C stop (see *Section 4.4.7*)
- I²C read bytes (see *Section 4.4.8*)
- I²C send bytes (see *Section 4.4.9*)
- I²C read data (see *Section 4.4.10*)
- I²C write data (see *Section 4.4.11*)

4.4.1 I²C start command

Table 12 describes the parameters that must be passed to the API_USBAll function, to issue an I²C start command.

Table 12. I²C start

API_USBAll parameters	
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: 0x02 **cmd** 0xB8: I²C command 0x01: Sends I²C start on SDA/SCL lines. **returnlen**: don’t care **pbuffer**: don’t care
Returned value	Error code

4.4.2 I²C send byte command

Table 13 describes the parameters that must be passed to the API_USBAll function, to issue an I²C send byte command.

Table 13. I²C send byte

API_USBAll parameters	
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: 0x03 **cmd** 0xB8: I²C command 0x02: Sends a byte on the SDA line. 0xXX: Byte to be sent **returnlen**: don't care **pbuffer**: don't care
Returned value	Error code
Example	cmd = B8 02 AA, sends data byte '0xAA' on the I²C bus.

4.4.3 I²C read byte command

Table 14 describes the parameters that must be passed to the API_USBAll function, to issue an I²C read byte command.

Table 14. I²C read byte

API_USBAll parameters	
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: 0x02 **cmd** 0xB8: I²C command 0x03: Sends back the byte value read on the SDA line. **returnlen**: pbuffer size **pbuffer**: read byte
Returned value	Error code

4.4.4 I^{2}C send ACK command

Table 15 describes the parameters that must be passed to the API_USBAll function, to issue an I^{2}C send an acknowledge (Ack) command.

Table 15. I^{2}C send Ack

API_USBAll parameters	
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: 0x02 **cmd** 0xB8: I^{2}C command 0x04: Sends I^{2}C Acknowledge on SDA/SCL I^{2}C bus. **returnlen**: don’t care **pbuffer**: don’t care
Returned value	Error code

4.4.5 I^{2}C send NoAck command

Table 16 describes the parameters that must be passed to the API_USBAll function, to issue an I^{2}C send a non-acknowledge (NoAck) command.

Table 16. I^{2}C send NoAck

API_USBAll parameters	
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: 0x02 **cmd** 0xB8: I^{2}C command 0x05: Sends I^{2}C NoAck on SDA/SCL I^{2}C bus. **returnlen**: don’t care **pbuffer**: don’t care
Returned value	Error code

4.4.6 I^2C read Ack command

Table 17 describes the parameters that must be passed to the API_USBAll function, to issue an I^2C read Ack command. The reader sends an Ack on the I^2C bus (one clock cycle followed by the read SDA value).

Table 17. I^2C read Ack

API_USBAll parameters	
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: 0x02 **cmd** 0xB8: I^2C command 0x09: Reads ACK value on the SDA line. **returnlen**: pbuffer size **pbuffer**: 0x80: if Ack (SDA=0) 0x81: if NoAck (SDA=1).
Returned value	Error code

4.4.7 I^2C stop command

Table 18 describes the parameters that must be passed to the API_USBAll function, to issue an I^2C send stop command.

Table 18. I^2C stop

API_USBAll parameters	
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: 0x02 **cmd** 0xB8: I^2C command 0x06: Sends I^2C stop on SDA/SCL I^2C bus. **returnlen**: don't care **pbuffer**: don't care
Returned value	Error code

4.4.8 I^2C read X bytes command

Table 19 describes the parameters that must be passed to the API_USBAll function to issue an I^2C read X bytes command. The reader reads the bytes from the I^2C bus and sends an Ack after each byte read.

Table 19. I^2C read X bytes

API_USBAll parameters	
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: 0x03 **cmd** 0xB8: I^2C command 0x0A: Read X bytes on SDA/SCL I^2C bus, and sends Ack between bytes 0x0X: Number of bytes to be read **returnlen**: pbuffer size **pbuffer**: read bytes
Returned value	Error code
Example	cmd= B8 0A 04 reads 4 bytes on the ^2C bus.

4.4.9 I^2C send X bytes command

Table 20 describes the parameters that must be passed to the API_USBAll function to issue an I^2C send X bytes command. The reader sends the bytes on the I^2C bus and reads an Ack after each byte sent.

Table 20. I^2C send X bytes

API_USBAll parameters	
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: size of the command (depends on the number of bytes to be written) **cmd** 0xB8: I^2C command 0x0B: Sends X bytes on SDA/SCL I^2C bus, and reads Ack between bytes 0x0X: Number of bytes to be read X Bytes: bytes to be sent **returnlen**: pbuffer size **pbuffer**: 0x80: if Ack (SDA=0) received for all the transmitted bytes 0x81: if at least one NoAck (SDA=1) received
Returned value	Error code
Example	cmd= B8 0B 04 AA 55 AA 55 sends the 4 bytes ‘0xAA 55 AA 55’ on the ^2C bus.

4.4.10 I²C read data command

Table 21 describes the parameters that must be passed to the API_USBAll function to issue an I²C read data command. The reader sends a complete I²C read data sequence.

Following are code examples in Visual Basic and C/C++.

Table 21. I²C read data

API_USBAll parameters	
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: 0x07 **cmd** 0xB8: I²C command 0x08: complete I²C read data sequence 0x0X: Number of bytes to be read 0x01: STARTKIT-M24LR-A mandatory parameter 0x0A: Device select code Address (2 bytes): Address of the data to be read. The I²C protocol address bytes are not inverted. **returnlen**: pbuffer size **pbuffer**: read bytes if command successful, ‘0x81’ otherwise
Returned value	Error code
Example	cmd= B8 08 04 01 A0 00 08 read 4 bytes from address 0x00 08

Visual Basic source code example

```
Public Function I2C_Read() As Long
Dim i As Long
Dim lngReqDataLen As Long
Dim abytReqData (0 To 63) As Byte
Dim abytAnswerDataLen (0 To 63) As Byte
Dim abytAnswerData (0 To 63) As Byte
Dim strReqData As String
Dim lngstatus As Long
'----I2C READ command
  ' STARTKIT-M24LR-A reader parameters :    0xB8
  ' I2C read command                   :    0x08
  ' Number of bytes to read            :    0x04
  ' Parameter                          :    0x01
  ' Device ID                          :    0xA0
  ' Address     :    0x0008
strReqData = "B8080401A0008"
  lngReqDataLen = Len(strReqData) \ 2
For i = 0 To lngReqDataLen - 1
      abytReqData(i) = CByte("&h" & Mid(strReqData, _
                       (i * 2) + 1, 2))
  Next i
' STARTKIT-M24LR-A I2C READ command
```

```
  lngStatus = API_USBAll(hcomm_public, _
                         lngReqDataLen,
                         abytReqData(0), _
                         abytAnswerDataLen(0),
                         abytAnswerData(0))
For i = 0 To abytAnswerDataLen(0) - 1
    strAnswerData = strAnswerData & _
                 i2hhh(CLng(abytAnswerData(i)), 2)
  Next i
 ' I2C READ REQUEST RESULT
  ' if(lngStatus = 0) then PASS else FAIL
  ' if (abytAnswerDataLen = 0) then No transponder answer
  ' else abytAnswerData contains the transponder answer

End Function
```

C/C++ source code example

```
/* I2C READ COMMAND [0xB8][0x08] COMMANDS */
int I2C_read (void)
{
byte sReqData[1024];
  byte sRspData[1024];
  int   iReqLen;
  unsigned char iRspLen;
  int iResult;
  int entry3;
  int i;
/* I2C READ request format                          */
  /* STARTKIT-M24LR-A reader param          : B8       */
  /* I2C read command                   : 08       */
  /* Number of bytes to be written      : 04    */
  /* parameter                          : 01    */
  /* Device ID                          : A0       */
  /* Address                            : 0008 */
/* request = B8 07 04 01 A0 0008 */
sReqData[0] = 0x07;/* Number of bytes of all bytes request for
API_USBAll function */
  sReqData[1] = 0xB8;
  sReqData[2] = 0x08;
  sReqData[3] = 0x04;
  sReqData[4] = 0x01;
  sReqData[5] = 0xA0;
  sReqData[6] = 0X00;
  sReqData[7] = 0X08;
  iReqLen = 8;
/* STARTKIT-M24LR-A I2C READ command */
  iResult = API_USBAll(hcomm_public,
                      sReqData[0],sReqData+1,
                      &iRspLen,sRspData);
printf("\n    --> answer  : ");
  if (iResult == 1)
   printf("No I2C Acknowledge received");
```

```
  else
  for (i=0; i<iRspLen; ai++)
          printf("%.2X",sRspData[i]);
/* I2C READ REQUEST RESULT */
  /* if(iResult == 0) PASS else FAIL */
  /* if (iRspLen == 0) No transponder answer */
  /* else sRspData contains the transponder answer */
return iResult;
}
```

4.4.11 I²C Write data command

Table 21 describes the parameters that must be passed to the API_USBAll function to issue an I²C write data command. The reader sends a complete I²C write data sequence.

Following are code examples in Visual Basic and C/C++.

Table 22. I²C write data

API_USBAll parameters	
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: size of the command (depends on the number of bytes to be written) **cmd** 0xB8: I²C command 0x07: I²C write data sequence 0x0X: Number of bytes to be written 0x01: STARTKIT-M24LR-A mandatory parameter 0x0A: Device select code Address (2 bytes): Address where the data will be written. The I²C protocol address bytes are not inverted. AA BB CC DD: Data bytes to be written **returnlen**: pbuffer size **pbuffer**: 0x80: if Ack (SDA=0) received for all the transmitted bytes 0x81: if at least one NoAck (SDA=1) received
Returned value	Error code
Example	cmd= B8 07 04 01 A0 00 08 AA 55 AA 55 writes 4 bytes (0xAA 55 AA 55’ at address 0x00 08.

Visual Basic source code example

```
Public Function I2C_Write() As Long
Dim i As Long
Dim lngReqDataLen As Long
Dim abytReqData (0 To 63) As Byte
Dim abytAnswerDataLen (0 To 63) As Byte
Dim abytAnswerData (0 To 63) As Byte
```

```
Dim strReqData As String
Dim lngstatus As Long
'----I2C WRITE command
  ' STARTKIT-M24LR-A reader parameters  :    0xB8
  ' I2C write command                  :    0x07
  ' Number of bytes to read            :    0x04
  ' Parameter                          :    0x01
  ' Device ID                          :    0xA0
  ' Address                            :    0x0008
  ' Data                               :    0xAABBCCDD
strReqData = "B8070401A00008AABBCCDD"
  lngReqDataLen = Len(strReqData) \ 2
 For i = 0 To lngReqDataLen - 1
      abytReqData(i) = CByte("&h" & Mid(strReqData, _
                       (i * 2) + 1, 2))
  Next i
' STARTKIT-M24LR-A USB I2C WRITE command
  lngStatus = API_USBAll(hcomm_public, _
                         lngReqDataLen,
                         abytReqData(0), _
                         abytAnswerDataLen(0),
                         abytAnswerData(0))
For i = 0 To abytAnswerDataLen(0) - 1
    strAnswerData = strAnswerData & _
                 i2hhh(CLng(abytAnswerData(i)), 2)
  Next i
' I2C WRITE REQUEST RESULT
  ' if(lngStatus = 0) then PASS else FAIL
  ' if (abytAnswerDataLen = 0) then No transponder answer
  ' else abytAnswerData contains the transponder answer
End Function
```

C/C++ source code example

```
/* I2C WRITE COMMAND [0xB8][0x07] COMMANDS */
int I2C_write (void)
{
 byte sReqData[1024];
  byte sRspData[1024];
  int   iReqLen;
  unsigned char iRspLen;
  int iResult;
  int entry3;
  int i;
/* I2C WRITE request format                        */
  /* STARTKIT-M24LR-A reader param         : B8        */
  /* I2C write command                   : 07        */
  /* Number of bytes to be written       : 04    */
  /* parameter                           : 01    */
  /* Device ID                           : A0        */
  /* Address                             : 0008 */
  /* Data                                : AABBCCDD */
/* request = B8 07 04 01 A0 0008 AABBCCDD     */
```

```
sReqData[0] = 0x0B;/* Number of bytes of all bytes request for
API_USBAll function */
  sReqData[1] = 0xB8;
  sReqData[2] = 0x07;
  sReqData[3] = 0x04;
  sReqData[4] = 0x01;
  sReqData[5] = 0xA0;
  sReqData[6] = 0x00;
  sReqData[7] = 0x08;
  sReqData[8] = 0xAA;
  sReqData[9] = 0xBB;
  sReqData[10] = 0xCC;
  sReqData[11] = 0xDD;
  iReqLen = 12;
 /* STARTKIT-M24LR-A I2C Write command */
  iResult = API_USBAll(hcomm_public,
                       sReqData[0],sReqData+1,
                       &iRspLen,sRspData);
printf("\n    --> answer  : ");
  if (iResult == 1)
  printf("No I2C Acknowledge received");
  else
  for (i=0; i<iRspLen; ai++)
          printf("%.2X",sRspData[i]);
/* I2C READ REQUEST RESULT */
  /* if(iResult == 0) PASS else FAIL */
  /* if (iRspLen == 0) No transponder answer */
  /* else sRspData contains the transponder answer */
return iResult;
}
```

4.5 STARTKIT-M24LR-A commands

The STARTKIT-M24LR-A can be controlled by the host computer using a set of USB dedicated commands:

- LED command
- Buzzer command
- Get version command

4.5.1 LED command

The LED command activates the STARTKIT-M24LR-A on-board LED. This command can be used by programmers to follow up application execution by making the LED blink when the application is running.

Refer to *Table 23* for a detailed description of the LED command.

Table 23. LED command

API_USBAll parameters	
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: 0x03 **cmd** 0x88: LED blinking command 0xXX: LED blinking duration where 0x01 is the shortest and 0xFF is the longest duration. 0xYY: Number of times the LED blinks **returnlen**: don’t care **pbuffer**: don’t care
Returned value	Error code
Example	cmd= 88 18 0A makes the LED blink 10 times (0x0A) during 0x18

4.5.2 Buzzer command

The Buzzer command drives the STARTKIT-M24LR-A on-board buzzer. This command can be used to signal the end of application execution or an application failure by generating one or multiple buzzes.

Refer to *Table 24* for a detailed description of the Buzzer command.

Table 24. Buzzer command

API_USBAll parameters	
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: 0x03 **cmd** 0x89: Buzzer management command 0xXX: Buzz duration where 01 is the shortest and 0xFF is the longest duration. 0xYY: Number of buzzes ranging from 1 (0x01) to 255 (0xFF) **returnlen**: don't care **pbuffer**: don't care
Returned value	Error code
Example	cmd= 89 18 0A makes the LED blink 10 times (0x0A) during 0x18

4.5.3 Get version command

The Get version command can be used to retrieve the version of the STARTKIT-M24LR-A used in the application.

Refer to *Table 25* for a detailed description of the Get version command.

Table 25. Get version command

API_USBAll parameters	
Parameters	**commHandle**: STARTKIT-M24LR-A handle assigned during reader detection process **cmdSize**: 0x01 **cmd** 0x86: Retrieves the version of the STARTKIT-M24LR-A. **returnlen**: don't care **pbuffer**: don't care
Returned value	Error code

Appendix A Useful source code zip files

The AN3227.zip package contains two simple projects to test the RF ISO 15693 and I^2C commands:

- AN3227_VB_sourcecode folder contains the Visual Basic project
- AN3227_C_sourcecode folder contains the C/C++ project,
- AN3227_software folder contains a PC software allowing to send I^2C and RF commands to the STARTKIT-M24LR-A through a simple user interface

These projects help users understand how to develop an application to communicate with the STARTKIT-M24LR-A.

The AN3227.zip package can be downloaded from http://www.st.com/dualeeprom.

Appendix B List of error codes

The error codes which are returned by the RF ISO 15693 and I^{2}C commands are the following:

```
#define HID_DEVICE_SUCCESS                 0x00
#define HID_DEVICE_NOT_FOUND               0x01
#define HID_DEVICE_NOT_OPENED              0x02
#define HID_DEVICE_ALREADY_OPENED          0x03
#define HID_DEVICE_TRANSFER_TIMEOUT        0x04
#define HID_DEVICE_TRANSFER_FAILED         0x05
#define HID_DEVICE_CANNOT_GET_HID_INFO     0x06
#define HID_DEVICE_HANDLE_ERROR            0x07
#define HID_DEVICE_INVALID_BUFFER_SIZE     0x08
#define HID_DEVICE_SYSTEM_CODE             0x09
#define HID_DEVICE_UNKNOWN_ERROR           0xFF
```

Revision history

Table 26. Document revision history

Date	Revision	Changes
30-Jul-2010	1	Initial release.