# MTI RFID ME™
# .Net Software Development Kit
# User's Manual
# Version 1.1.0


**June 2012**

# Contents

## Version Control

| Date & Version | Affects | Changes |
|---|---|---|
| 22 Oct, 2011 ver. 0.90 | All | Initial release of the SDK. |
| 28 Feb, 2012 Ver 1.0.0 | All | Changed communications dll to new version from MTI |
| 22 Mar 2012 Ver 1.1 | RU-824 | Added RU-824 and RU-861 functionality to SDK. |
| 20 Jun 2012 Ver 1.1.0.110 | All | Recompiled in VS2008 to allow those that have not upgraded to still use SDK. Also fixed write functionality for RU-888. |

## Introduction

This software development kit (SDK) has been developed in conjunction with MTI to allow .NET developers an easier time in deploying solutions that are designed to work with the MTI RFID ME™ family of devices. This SDK has been developed such that any programmer with a beginner to intermediate skill level should be able to create and deploy solutions that allow for the integration of the MTI hardware. The SDK will support any .Net programming language (C#, Visual Basic, etc.). Java is not currently supported. There are two code examples included with the installed SDK – one in Visual Basic and the other in C#. Finally, this version of the SDK must be used with Visual Studio 2008 or higher.

At this time, for the RFID ME, only standard EPCGlobal compliant functionality and necessary hardware controls have been implemented into this SDK. Advanced NXP functionality and additional hardware controls (radio controls, antenna port configuration, etc.) that have been published in the ***MTI RU-888 RFID Module USB API Reference Manual v3.1*** are not supported. Should there be a need, please contact us via email at autoid@ohio.edu and it will be determined at that time if the functionality is a necessary upgrade or if separate SDK will be developed.

For the RU-824 and the RU-861, only the standard EPCGlobal compliant functionality and necessary hardware controls have been implemented into this SDK. Advanced hardware controls (firmware communications, GPIO, etc.) that have been published in the ***MTI RU-824/861 RFID Reader/ Module Command Reference Manual Version 1.0*** are not supported. Should there be a need, please contact us via email at autoid@ohio.edu and it will be determined at that time if the functionality is a necessary upgrade or if separate SDK will be developed.

By purchasing this SDK, you are entitled to basic upgrades and are encouraged to report any problems or bugs that are found. If a major re-write of the SDK is undertaken, you will be alerted and provided with an opportunity to upgrade.

The sample programs included in the SDK distribution are fully functional and in theory one could use them “as is”. They are there to a) allow us to ensure that the SDK is working correctly, b) provide you with a framework from which to quickly develop an application and c) to show you how we had intended to have the SDK used. Feel free to use, modify and play with the programs as desired. If you have a need, feel free to contact us. We are happy to answer questions via email.

# Overview

The setup file will place all files in "c:\Program Files\BAIT\RFIDMeDevKit" unless another location is selected at setup. When adding the DLLs in your project, you will need to reference them in this location.

The general process for using the SDK and the hardware are as follows:

1. Plug the reader into the computer
2. In your project:
    a. Add RFIDMeDevKit.DLL as a project reference.
    b. Add Transfer.DLL as a project resource – make sure it is set to "always copy" under properties.
3. From your program:
    a. Connect to the reader (to ensure it is plugged in)
    b. Initiate a read (to see what tags exist)
    c. Use the results in a business process (to solve your specific problem)
4. When you deploy (via Setup file or manually):
    a. Make sure RFIDMeDevKit.DLL is included in package.
    b. Make sure Transfer.DLL is included in the package.
    c. Make lots of money!

At times during the documentation, references to the RFID tag's memory structure are made. Figure 1 (below) is from the Class 1, Gen 2 air interface protocol and shows how the tag's memory is laid out.

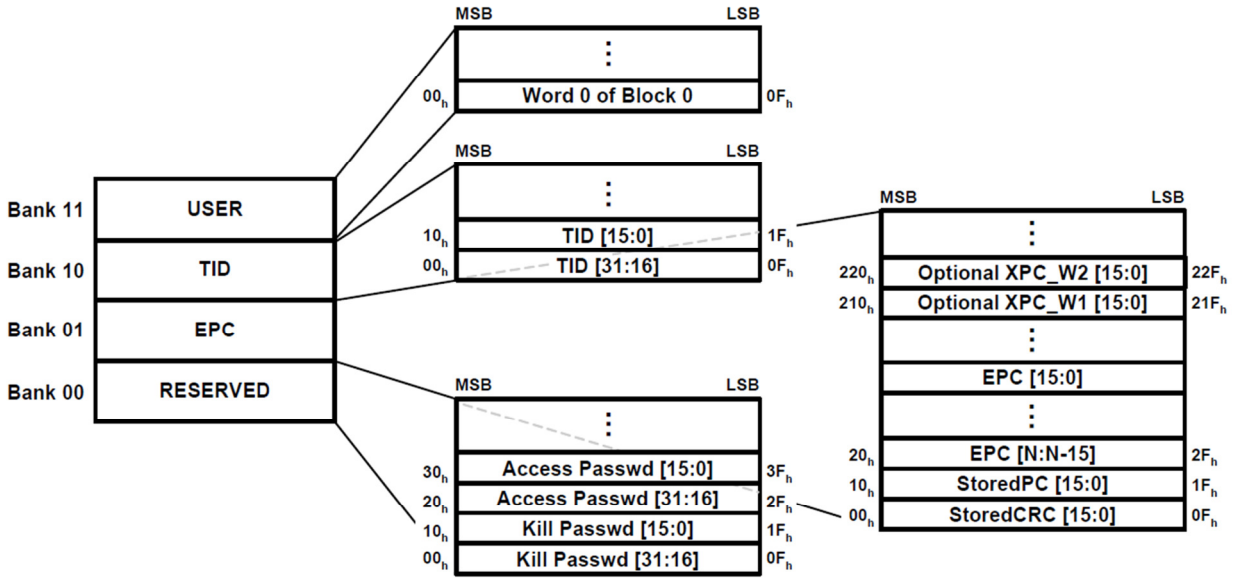



*Figure 1.* C1G2 logical memory map (uhfc1g2_1_2_0_standard-20080511.pdf – downloaded from http://www.gs1.org/gsmp/kc/epcglobal/uhfc1g2 on 22 Oct, 2011).

# Common Methods

## About

**Usage:** About()
**Description:** Reports current SDK version and identified reader.
**Parameters:** None
**Returns:** *(String)* "RFID ME™ SDK ver x.x, Current reader set to xxxx".
**Comments:** None.

## Connect

**Usage:** Connect(string myReader)
**Description:** Attempts to connect to the indicated RFID reader.
**Parameters:**
*myReader (string):* The type of reader that is to be connected to. Currently the only option is "RFIDME". As additional RFID readers are introduced, this list will increase.
**Returns:** *(String)* "Connected", "Reader not connected", or the specific error message if the connection attempt fails.
**Comments:** Once a connection has been made, there is a basic assumption that all subsequent commands are to be made against the type of reader initially selected. As such, the SDK will remember (during a given session), what reader was connected and will determine the applicability of the subsequent commands based on the reader initially selected. Should there be a need to send commands to a different type of reader, the SDK will have to be functionally disconnected and a new connection will need to be made.

## Disconnect

**Usage:** Disconnect()
**Description:** Attempts to disconnect from the RFID ME™ reader.
**Parameters:** None
**Returns:** *(String)* "Disconnected" or "Problem disconnecting – " and the specific error message received while trying to disconnect.
**Comments:** Disconnect is not actually required, but is suggested to help control potential memory leaks.

## getUSBHIDInfo

**Usage:** getUSBHIDInfo()
**Description:** Get USB HID device path for use with detecting device insertion and removal.
**Parameters:** None
**Returns:** *(String)* Device path name.
**Comments:** None.

## killTag

**Usage:** killTag(string myPassword, string myTagID)
**Description:** Attempts to permanently kill (disable) the tag. This command assumes that only one tag will be in the reader's field. If more than one tag is found, an error will be returned. ***Once done, this tag will not respond to any reader again!***
**Parameters:**
*myPassword (string):* The tag's current Kill password, formatted as a single hex string without any spaces.
*myTagID (string):* The EPC value of the tag that is to be killed.
**Returns:** *(String)* "Success" or the specific reader error encountered. If an invalid password is defined, an error indicating such is returned.
**Comments:** This is a permanent action. It is suggested that a) this command be used with extreme care and b) that execution of the killTag be done inside of a loop that requires the user to positively confirm that the action is desired. As of this version, only the RFID ME™ is supported. Support for the RU-824 and RU-861 should be available by early May, 2012.

**>>> Remember, once a tag is killed it is basically scrap! <<<**

## lockTag

**Usage:** lockTag(string myTagID, string myPassWord, string myLockAction, string myLockItem)
**Description:** Attempts to lock/unlock the indicated portion of the tag. This command assumes that only one tag will be in the reader's field. If more than one tag is found, an error will be returned.
**Parameters:**
*myTagID (string):* The EPC value of the tag that is to be read.
*myPassword (string):* The tag's current Access password, formatted as a single hex string without any spaces.
*myLockAction (string)*: The type of action to be taken. The valid options are "Unlock", "PermanentUnlock", "Lock" and "PermanentLock". "Lock" and "Unlock" will allow the user to change the setting in the future. "PermanentLock" and "PermanentUnlock" will permanently set the state of the tag, and future attempts at locking or unlocking the tag will fail.
*myLockItem (string)*: The section of the tag to be locked. The valid options are "Kill", "Access", "EPC", "TID" and "User".
**Returns:** *(String)* "Success" or the specific reader error encountered. If an invalid password field is defined, an error indicating such is returned.
**Comments:** Since the permanent lock/unlock actions cannot be undone, it is suggested that "Lock" and "Unlock" be used unless it is known that the settings will never need to be changed again. As of this version, only the RFID ME™ is supported. Support for the RU-824 and RU-861 should be available by early May, 2012.

## readEPC

**Format:** readEPC(String ShowSpaces, String myDelimiter)
**Description:** Reads the EPC memory bank (MB01) from any tags that are in range.
**Parameters:**

*Show Spaces (Boolean)* – Show a space between each byte of the EPC value. This function allows the SDK to display data.

*Delimiter (string)* – The character(s) used to separate the returned EPC tag values if there is more than one tag in the field.

**Returns:** *(String)* "No tags found" or a list of hex formatted EPC tag values, separated by the user selected delimiter. If there is a problem, the reader specific error that was encountered will be returned.

**Comments:** It is suggested that a space NOT be used as a delimiter. Additionally, non-printable ASCII characters are not currently supported. If the programmer is using C#, it might be possible to use "/r/n" to force a new line, but this has not been tested and it is possible that doing so might cause problems within the DLL.

## readMemory

**Usage:** readMemory(string myEPC, string myMemoryBank, string myPassword, string myStartAddress, string myDataLength)

**Description:** Attempts to read the indicated section of an RFID tag. This is a function that should be reserved for someone who understands how a tag's memory is configured. This command assumes that only one tag will be in the reader's field. If more than one tag is found, an error will be returned.

**Parameters:**

*myEPC (string):* The EPC value of the tag that is to be read.

*myPassword (string):* The tag's current Access password, formatted as a single hex string without any spaces.

*myStartAddress (string)*: The word location where the read is to begin.

*myMemoryBank (string)*: The memory bank to be read, formatted as a two digit string. Valid options are "00", "01", "02" and "03". See Figure 1 above or refer to the UHF Class 1 Gen 2 Air Interface Protocol Specification for further information on the structure of the tag and where to look for information.

*myDataLength (string):* The number of words (four digit hex values) to be read from user memory. See Figure 1 above or refer to the UHF Class 1 Gen 2 Air Interface Protocol Specification for further information on the structure of the tag and where to look for information.

**Returns:** *(String)* The tag's indicated memory block in a single hex string that is of the indicated length. If there is a problem, the reader specific error that was encountered will be returned.

**Comments:** This function is included for those that are looking to do more controlled reading from the RFID tag. For the majority of the applications, use of this method is unnecessary as the other methods in this SDK will provide a simpler interface for obtaining the desired results. For example, to read the EPC value of a tag, using readEPC provides a simple way to get the information. To use readMemory, the programmer will need to ensure that they are a) addressing the correct memory bank (MB01), b) start reading from the correct memory address in the memory bank (word 02), and c) know to read the correct number of words (which is obtained from the first 5 bits of word 01. As of this version, only the RFID ME™ is supported. Support for the RU-824 and RU-861 should be available by early May, 2012.

## readUserMemory

**Usage:** readUserMemory(string myTagID, string myPassword, string myDataLength)
**Description:** Attempts to read the tag's user memory (memory bank 11) from the selected tag. This command assumes that only one tag will be in the reader's field. If more than one tag is found, an error will be returned.
**Parameters:**
*myTagID (string):* The EPC value of the tag that is to be read.
*myPassword (string):* The tag's current Access password, formatted as a single hex string without any spaces.
*myDataLength (string):* The number of words (four digit hex values) to be read from user memory.
**Returns:** *(String)* The tag's User Memory in a single hex string that is of the indicated length. If there is a problem, the reader specific error that was encountered will be returned.
**Comments:** If the tag does not have user memory enabled, a message to that effect will be returned. See the EPCGlobal Gen2 Air Interface Protocol or the tag's datasheet for information on the available memory size.

## setPassWord

**Usage:** setPassWord(string myTagID, string myPassWordField, string myCurrentPwd, string myNewPassWord)
**Description:** Attempts to set/reset the tag's passwords (Access and Kill) in the reserved (memory bank 00). This command assumes that only one tag will be in the reader's field. If more than one tag is found, an error will be returned.
**Parameters:**
*myTagID (string)*: The EPC value of the tag that is to be programmed.
*myPassWordField (string)*: The password field to be programmed. The two options are "Access" and "Kill".
*myCurrentPassWord (string)*: The tag's current Access password, formatted as a single hex string without any spaces.
*myNewPassWord (string)*: The new password, formatted as a single hex string without any spaces.
**Returns:** *(String)* "Success" or the specific reader error encountered. If an invalid password field is defined, an error indicating such is returned.
**Comments:** This method will select the desired tag, read both current passwords from the reserved memory block (MB00), generate a new overall reserved MB00 value and write that back. As of this version, only the RFID ME™ is supported. Support for the RU-824 and RU-861 should be available by early May, 2012.

## setPower

**Usage:** setPower(int Power)
**Description:** Set the power for the reader. The RFID ME™ has a value range of 5-18. The RUI-824 has an implemented range of 5-24.
**Parameters:**
*Power (integer)*: The power level that is desired.

**Returns:** *(String)* The set power level or the specific error message returned from the reader. If a power level that is outside of the range of the reader is attempted, an error message indicating that the parameter is invalid will be returned.

### writeEPC

**Usage:** writeEPC(string myTagID, string myPassword, string myNewData)
**Description:** Attempts to write the EPC value (MB01) to a tag. This function assumes only one tag exists in the field. This command assumes that only one tag will be in the reader's field. If more than one tag is found, an error will be returned.
**Parameters:**
*myTagID (string)***:** The tag to be programmed.
*myPassword (string)*: The tag's current Access password, formatted as a single hex string without any spaces.
*myNewData (string)*: The new data to be programmed, formatted as a single hex string without any spaces.
**Returns:** *(String)* "Success" or the error that was encountered while trying to write the value.
**Comments:** Due to the nature of RFID, it is suggested that the writeEPC method be invoked within a loop that will iteratively attempt to program the tag for a finite number of times until it succeeds. In addition to writing the EPC value, the writeEPC method will reset the EPC length value, leaving **ALL** other PC word data as it is currently encoded.

### writeMultipleEPC

**Usage:** writeMultipleEPC(string myNewEPC, string myPassword)
**Description:** Attempts to program **ALL** tags within range.
**Parameters:**
*myNewEPC (string)*: The new data to be programmed, formatted as a single hex string without any spaces.
*myPassword (string)*: The tag's current Access password, formatted as a single hex string without any spaces.
**Returns:** *(String)* "All tags programmed" or the tag id(s) that failed to program. The specific error message that was encountered will not be passed back in this method. All tags that were not explicitly listed can be assumed to have successfully programmed.
**Comments:** Due to the nature of RFID, it is suggested that the writeUserMemory method be invoked within a loop that will iteratively attempt to program the tag for a finite number of times until it succeeds. If a failure occurs, it is suggested that the offending tag be removed, which is why it's EPC value is reported back. In addition to writing the EPC value, the writeMultipleEPC method will reset the EPC length value, leaving **ALL** other PC word data as it is currently encoded. As of this version, only the RFID ME™ is supported. Support for the RU-824 and RU-861 should be available by early May, 2012.

### writeUserMemory

**Usage:** writeUserMemory(string myTagID, string myPassword, string myNewData)
**Description:** Attempts to write the user memory value (MB11) to a tag. This command assumes that only one tag will be in the reader's field. If more than one tag is found, an error will be returned.

**Parameters:**
*myTagID (string)***:** The tag to be programmed.
*myPassword (string)*: The tag's current Access password, formatted as a single hex string without any spaces.
*myNewData (string)*: The new data to be programmed, formatted as a single hex string without any spaces.
**Returns:** *(String)* "Success" or the error that was encountered while trying to write the value.
**Comments:** Due to the nature of RFID, it is suggested that the writeUserMemory method be invoked within a loop that will iteratively attempt to program the tag for a finite number of times until it succeeds. As of this version, only the RFID ME™ is supported. Support for the RU-824 and RU-861 should be available by early May, 2012.

## RU-824/RU-861 Specific Methods

The following commands can only be used with the RU-824 or the RU-861 as the RFID ME™ chipset does not support them. This is a limitation of the RFID ME™ and is not

### getAntConfig

**Usage:** getAntConfig()
**Description:** Returns the current antenna configuration from the reader.
**Parameters:** None.
**Returns:** *(String)* Antenna configuration as comma delimited string (Power, Dwell, Inventory Cycles), specific reader error message or "Reader does not support this function".
**Comments:** None.

### getSingulationAlgorithm

**Usage:** getSingulationAlgorithm()
**Description:** Reports the current singulation algorithm.
**Parameters:** None.
**Returns:** *(String)* The singulation algorithm value (0=fixed Q or 1=dynamic Q), the specific reader error message or "Reader does not support this function".
**Comments:** None.

### getSingulationAlgorithmConfig

**Usage:** getSingulationAlgorithmConfig()
**Description:** Returns the current singulation algorithm configuration
**Parameters:** None.
**Returns:** *(String)* The singulation algorithm configuration as a comma separated string (see the low level Command Reference Manual from MTI for the specific information), the specific reader error message or "Reader does not support this function".
**Comments:** None.

### setDwellTime

**Usage:** setDwellTime(int DwellTime)
**Description:** Sets the antenna dwell time.
**Parameters:**
*DwellTime (int)***:** The amount of time to dwell on an antenna during the inventory process.
**Returns:** *(String)* "Success", the specific reader error message or "Reader does not support this function".
**Comments:** None.

### setNumInvCyc

**Usage:** setNumInvCycle(int InvCycles)
**Description:** Sets the antenna's number of inventory cycles.
**Parameters:**
*InvCycles (int)***:** The number of inventory cycles to perform during the inventory process.
**Returns:** *(String)* "Success", the specific reader error message or "Reader does not support this function".
**Comments:** None.

### setSingulationAlgorithm

**Usage:** setSingulationAlgorithm(bool DynamicQ)
**Description:** Sets the type of inventory Q cycle to perform during the inventory process.
**Parameters:**
*DynamicQ (bool)***:** Whether or not to use a dynamic Q singulation algorithm.
True = use a dynamic Q
False = do not use a dynamic Q
**Returns:** *(String)* "Success", the specific reader error message or "Reader does not support this function".
**Comments:** None.

### setSingulationAlgorithmConfig

**Usage:** setSingulationAlgorithmConfig(string myNewConfig, bool myDynamicQ)
**Description:** Sets the singulation algorithm configuration.
**Parameters:**
*myNewConfi(string)***:** The singulation algorithm configuration values to be used as a comma delimited hexadecimal string. See the low level Command Reference Manual from MTI for the specific information on what the values need to be.
*myDynamicQ(bool):* Whether or not a dynamic Q is currently set.
True = use a dynamic Q
False = do not use a dynamic Q
**Returns:** *(String)* "Success", the specific reader error message or "Reader does not support this function".
**Comments:** None.

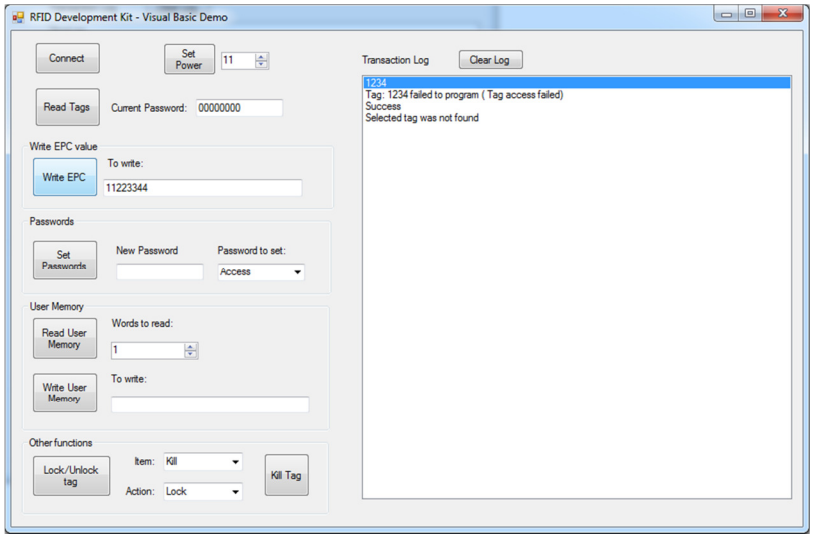
# Trouble Shooting

| | |
|---|---|
| "A problem occurred while communicating with the reader" occurs despite the reader being plugged in. | There is a known issue with some USB devices that results in an inability to correctly connect to the RFID reader. Try switching out any USB mice being used (Dell mice seem to cause a problem) for a different brand.<br><br>This problem is being worked on and you will be notified when it is fixed. |
| Intermittent reading of tags | Confirm that the RFID tag is not sitting directly on top of the reader. It has been noticed that if a small gap (1 inch or so) is maintained between the reader and the tag, the system works much smoother. |
| A lot of errors occur when programming a tag – the tag programs eventually, but it takes multiple tries. | • Ensure that the power is not too low **OR** high<br>• Ensure the tag is not sitting directly on the reader (see above).<br>• Ensure that only one tag is in range. If other tags are at the edge of the read envelope, move them away. |

# Code Examples

The following code assumes a simple form that has had the RFIDMeDevKit added as a reference and that the “transfer.dll” file is included in the same directory from which the compiled application will run.

The sample programs include (one is C# and the other in Visual Basic) were both developed in Microsoft Visual Studio 2010. If you need to open it in a previous version of VS, you will probably have to create a new project and manually add the form into that project. Both programs are the same, with the programming language as the only difference.




## Code Example (Visual Basic.NET)

To include the RFIDMeDevKit.DLL, you will need the following code added to your program…

```
    Public Class Form1
        'Declare RFID ME™ SDK
        Public myReader As New RFIDMEDevKit.reader
```

## Code Example (C#)

To include the RFIDMeDevKit.DLL, you will need the following code added to your program…

```
    public partial class Form1 : Form
    {
        //Declare RFID ME™ SDK
        public RFIDMEDevKit.reader myReader = new RFIDMEDevKit.reader();
```