CipherLab
User Guide

RFID Service API Programming

RK25
RS35

Version 1.07

CIPHERLAB CO., LTD.
Website: http://www.cipherlab.com

Release Notes

Version	Date	Notes
1.07	Sep. 10, 2021	▸ New: **1.1.6. Miscellaneous Configuration** – * GetTriggerSwitchMode * SetTriggerSwitchMode * GetSwitchMode * SetSwitchMode * GetModuleUniqueID * GetFilterDuplicate * SetFilterDuplicate * ClearFilterDuplicate ▸ New: Support RS35 mobile computer ▸ Modified: **1.1.6. Miscellaneous Configuration** – * GetRFIDSwitchStatus * SetRFIDSwitchStatus
1.06	June 23, 2021	▸ Modified: **1.1.6. Miscellaneous Configuration** – Add GetProtectTemperature & SetProtectTemperature functions ▸ Modified: **1.1.6. Miscellaneous Configuration** – Remove DeviceResponse.TagLock from RFIDDirectWriteTagByEPC, RFIDDirectWriteTagByTID, RFIDDirectKillTag, RFIDDirectUnlockTag, RFIDDirectLockTag, RFIDDirectPermanentLockTag, RFIDDirectUntraceableTag, & RFIDDirectAuthenticateTag functions ▸ Modified: **Chapter 2 Tag Access Demonstration** – Update the intro part
1.05	Nov. 24, 2020	▸ Modified: **1.1.6. Miscellaneous Configuration** – *RFIDWriteTagMassive: byte[] data (max. 32 bytes allowed) *RFIDDirectWriteTagByEPC:byte[] data (max. 32 bytes allowed) *RFIDDirectReadTagByEPC:int length (max. 32 bytes allowed) *RFIDDirectWriteTagByTID:byte[] data (max. 32 bytes allowed) *RFIDDirectReadTagByTID:"byte[] data" removed; int length (max. 32 bytes allowed) ▸ Modified: **1.1.6. Miscellaneous Configuration** – Remove WorkMode.SingleTagMode from SetWorkMode/GetWorkMode functions ▸ Modified: **1.1.6. Miscellaneous Configuration** – Remove WorkMode.SingleTagMode from SetAllGen2/SetAllQValue example code
1.04	Apr. 24, 2020	▸ Modified: **1.1.6. Miscellaneous Configuration** – append PR_ASK_Miller2_250KHz & PR_ASK_FM0_250KHz to GetRFLink /SetRFLink/GetAllRFLink/SetAllRFLink ▸ Modified: **1.1.6. Miscellaneous Configuration** – SetPowerMode, GetContinuousInventoryTime, SetContinuousInventoryTime, GetPowerMode appended

1.03	Aug. 23, 2019	▸ Modified: **Chapter 1 UHF RFID API** – RfidAPI_vx_x_xx.dll (for Xamarin) added to the Chapter 1 front page ▸ Modified: **1.1.6. Miscellaneous Configuration** – parameter names of GetModuleTemperature updated ▸ Modified: **1.1.6. Miscellaneous Configuration** – return value of DeviceTriggerStatus updated ▸ Modified: **1.1.6. Miscellaneous Configuration** – add KeyEventOutput & InterCharDelay to GetDataOutputSettings & SetDataOutputSettings functions ▸ Modified: **1.1.6. Miscellaneous Configuration** – GetJapanChannel & SetJapanChannel functions appended ▸ Modified: **1.1.6. Miscellaneous Configuration** – update GetRFLink, SetRFLink, GetAllRFLink, and SetAllRFLink functions with remarks ▸ Modified: **1.1.6. Miscellaneous Configuration** – update GetWorkMode & SetWorkMode functions with remarks
1.02	Jul. 05, 2019	▸ Modified: **1.1.6. Miscellaneous Configuration** – parameter of ScanMode.Alternate added to GetScanMode/SetScanMode ▸ Modified: **1.1.6. Miscellaneous Configuration** – added 6 APIs: GetRecognizedEPCEncoding/SetRecognizedEPCEncoding; GetDataOutputSettings/SetDataOutputSettings; RFIDDirectUntraceableTag/RFIDDirectAuthenticateTag ▸ Modified: **1.2. Intent** – add GeneralString.Intent_GUN_Attached & GeneralString.Intent_GUN_Unattached ▸ Modified: **1.2. Intent** – GeneralString.Intent_GUN_Power appended ▸ Modified: **1.2. Intent** – GeneralString.EXTRA_DATA_TYPE/GeneralString.EXTRA_RESPONSE parameters of GeneralString.Intent_RFIDSERVICE_TAG_DATA ▸ Modified: **1.2. Intent** – GeneralString.Intent_RFIDSERVICE_EVENT parameters ▸ Modified: **1.3. Class** - DeviceEvent (BatteryLose & Battery_Re_Plug appended) ▸ Modified: **1.3. Class** – FWUpdateErrorCode
1.01	May 10, 2019	▸ New: **Chapter 2 Tag Access Demonstration**
1.00	Mar. 13, 2019	First Release

Contents

Introduction

This Programming Guide contains necessary information for building Android applications controlling the UHF RFID device on RK25/ RS35 Mobile Computers.

We recommend that you read the documents thoroughly before use and keep them at hand for quick reference.

Thank you for choosing CipherLab products!

Development Tool

Before developing Android applications, programmers are supposed to make their machine ready with the requirements as follows:

- Java SE Development Kit (JDK, Java SE 7 or greater is recommended)
- Android SDK
- Android Studio, Eclipse IDE, or Xamarin for Visual Studio
- Visual Studio 2015 (a must while using Xamarin)

The software tools listed above are free and can be downloaded from their official websites respectively. Programmers are assumed to possess Android programming knowledge.

Chapter 1

UHF RFID API

Before developing your self-made application, the offered “***RfidAPI_vx_x_xx.jar***” or “***RfidAPI_vx_x_xx.dll***” library file has to be imported into your project.

Library Required	Location
RfidAPI_vx_x_xx.jar (for Android Studio or Eclipse) *RfidAPI_vx_x_xx.dll* (for Xamarin)	/sdcard/RfidService_Data

IN THIS CHAPTER

1.1. Configuration

1.1.1. Initialization

InitInstance	
Purpose	Creates an RfidManager instance before employing any APIs.
Syntax	**RfidManager InitInstance (Context context);**

Example	`private RfidManager mRfidManager;`
	`mRfidManager = RfidManager.InitInstance(this);`

Return Value	Gets an RfidManager instance if successful, else null.

Release	
Purpose	Releases resources.
Syntax	**void Release ();**
Example	`mRfidManager.Release();`

1.1.2. Service Version

GetServiceVersion	
Purpose	Obtains the RFID service version.
Syntax	**String GetServiceVersion ()**
Example	`String ver = mRfidManager.GetServiceVersion();`

1.1.3. API Version

GetAPIVersion	
Purpose	Obtains the RFID service api version.
Syntax	**String GetAPIVersion ()**
Example	`String ver = mRfidManager.GetAPIVersion();`

1.1.4. UHF RFID Device Information

GetDeviceInfo

Purpose	Gets the firmware information about the UHF RFID device.
Syntax	**public DeviceInfo GetDeviceInfo()**
Parameters	DeviceInfo *info*

SerialNumber	UHF RFID device serial number
Region	UHF RFID device region
KernelVersion	UHF RFID device kernel version
UserVersion	UHF RFID device firmware version
RFIDModuleVersion	UHF RFID device module firmware version

Example

```
DeviceInfo info = mRfidManager.GetDeviceInfo();
SerialNumber.setText(info.SerialNumber);
Region.setText(info.Region);
KernelVersion.setText(info.KernelVersion);
UserVersion.setText(info.UserVersion);
RFIDModuleVersion.setText(info.RFIDModuleVersion);
```

Return Value — If successful, it returns DeviceInfo.
Otherwise, it returns null.

GetDevicePowerSavingState

Purpose	Gets the power saving timeout of the UHF RFID device.
Syntax	**public int GetDevicePowerSavingState()**

Example

```
int time = mRfidManager.GetDevicePowerSavingState();
if(time==-1){
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m);}
```

Return Value	If successful, it returns the power saving timeout. Otherwise, it returns -1.
See Also	SetDevicePowerSavingState

SetDevicePowerSavingState

Purpose	Sets the power saving timeout of the UHF RFID device. By default, the idle power saving timeout is set to 2 minutes.
Syntax	**public int SetDevicePowerSavingState(int time)**
Parameters	The time delay before entering sleep mode (5 seconds for each unit ranging from 0~254). Default=24 (2 minutes). If Time=0, it never enters sleep mode.

Example

```
int re = mRfidManager.SetDevicePowerSavingState(100);
if(re!=ClResult.S_OK.ordinal()){
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m);}
```

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	GetDevicePowerSavingState

GetBatteryLifePercent

Purpose: Gets the battery information about the UHF RFID device.

Syntax: **public int GetBatteryLifePercent(DeviceVoltageInfo info)**

Parameters:

`Percentage`	The UHF RFID device battery life in percentage. Percentages values: 0, 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100.
`Voltage`	The UHF RFID device battery in voltage.
`ChargeStatus`	The UHF RFID device battery in charging status. 0 – not charged, 1 – charging, 2 – charge done, 3 – battery fault

Example:

```
DeviceVoltageInfo info = new DeviceVoltageInfo();
int re = mRfidManager.GetBatteryLifePercent(info);
if(re!=ClResult.S_OK.ordinal()){
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
else{
Log.i(TAG, "info Percentage = " + info.Percentage );
Log.i(TAG, "info Voltage = " + info.Voltage );
Log.i(TAG, "info ChargeStatus = " + info.ChargeStatus ); }
```

Return Value: If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

1.1.5. Reset to Default

ResetToDefault

Purpose: Resets the UHF RFID device to its factory default settings.

Syntax: **int ResetToDefault()**

Example:

```
int re = mRfidManager.ResetToDefault();
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value: If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

1.1.6. Miscellaneous Configuration

GetNotification

Purpose	Gets notification settings.
Syntax	**int GetNotification(NotificationParams settings)**
Parameters	A default value comes with an asterisk "*".

BeepType ReaderBeep
[in][out] A value that specifies the sound to play.

BeepType.Mute	
BeepType.Default	*
BeepType.Ringtone1	
BeepType.Ringtone2	
BeepType.Ringtone3	
BeepType.Ringtone4	

Enable_State *BatteryLED*
[in][out] Low battery LED light.

Enable_State.FALSE	Disables LED
Enable_State.TRUE	Enables LED *

Enable_State BatteryBeep
[in][out] Low battery beep.

Enable_State.FALSE	Disables beep *
Enable_State.TRUE	Enables beep

Enable_State *TemperatureWarning*
[in][out] High temperature warning.

Enable_State.FALSE	Disables beep
Enable_State.TRUE	Enables beep *

Example

```
NotificationParams settings = new NotificationParams();
int re = mRfidManager.GetNotification(settings);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value: If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also: SetNotification

SetNotification

Purpose	Configures notification settings.
Syntax	**int SetNotification(NotificationParams settings)**
Parameters	A default value comes with an asterisk "*". BeepType *ReaderBeep* [in][out] A value that specifies the sound to play.

BeepType.Mute	
BeepType.Default	*
BeepType.Ringtone1	
BeepType.Ringtone2	
BeepType.Ringtone3	
BeepType.Ringtone4	

Enable_State *BatteryLED*
[in][out] Low battery LED light.

Enable_State.FALSE	Disables LED
Enable_State.TRUE	Enables LED *

Enable_State *BatteryBeep*
[in][out] Low battery beep.

Enable_State.FALSE	Disables beep *
Enable_State.TRUE	Enables beep

Enable_State *TemperatureWarning*
[in][out] High temperature warning.

Enable_State.FALSE	Disables beep
Enable_State.TRUE	Enables beep *

Example

```
NotificationParams settings = new NotificationParams();
mRfidManager.GetNotification(settings);
settings.ReaderBeep = BeepType.Ringtone4;
settings.BatteryLED = Enable_State.TRUE;
settings.BatteryBeep = Enable_State.TRUE;
settings.ModuleTemperature = Enable_State.TRUE;
int re = mRfidManager.SetNotification(settings);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	GetNotification

GetConnectionStatus

Purpose	Get status with UHF RFID devices.
Syntax	**boolean GetConnectionStatus();**
Example	`Boolean Status = mRfidManager.GetConnectionStatus();`
Return Value	If Attached the UHF RFID devices, it returns true. Otherwise, it returns false.

FirmwareUpdate

Purpose: Sets firmware update or module firmware update of the UHF RFID device.

Syntax: **public int FirmwareUpdate(String path)**

Example:

```
String path = Environment.getExternalStorageDirectory().getPath();
path = path + "/PIS_S_v0.01o_DVT3.SHX";
int re = mRfidManager.FirmwareUpdate(path);
if(re!=ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value: If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also: GeneralString.Intent_FWUpdate_ErrorMessage
GeneralString.Intent_FWUpdate_Percent
GeneralString.Intent_FWUpdate_Finish

ShutdownDevice

Purpose: Shuts down the UHF RFID device.

Syntax: **public int ShutdownDevice()**

Example:

```
int re = mRfidManager.ShutdownDevice();
if(re!=ClResult.S_OK.ordinal()){
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value: If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

GetModuleTemperature

Purpose: Get module temperature.

Syntax: **public int GetModuleTemperature(ModuleTemperature temperature)**

Parameters:

GunModuleTemperature	The UHF RFID device module temperature.
GunProtectTemperature	The UHF RFID device over temperature protection.

Example	`ModuleTemperature t = new ModuleTemperature();`
	`int re = mRfidManager.GetModuleTemperature(t);`
	`if (re != ClResult.S_OK.ordinal()) {`
	`String m = mRfidManager.GetLastError();`
	`Log.e(TAG, "GetLastError = " + m);}`

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().

GetProtectTemperature

Purpose	Get current protect temperature. Ranges from 0~100. Default=65.
Syntax	**public int GetProtectTemperature()**

Example	`int PT = mRfidManager.GetProtectTemperature();`
	`if(PT==-1) {`
	`String m = mRfidManager.GetLastError();`
	`Log.e(TAG, "GetLastError = " + m); }`

Return Value	If successful, it returns the protect temperature. Otherwise, it returns -1
See Also	SetProtectTemperature

SetProtectTemperature

Purpose	Set new protect temperature. Ranges from 0~100. Default=65.
Syntax	**public int SetProtectTemperature(int pt)**

Example	`int re = mRfidManager.SetProtectTemperature(50);`
	`if(re!=ClResult.S_OK.ordinal()){`
	`String m = mRfidManager.GetLastError();`
	`Log.e(TAG, "GetLastError = " + m); }`

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	GetProtectTemperature

GetRFIDSwitchStatus	
Purpose	Gets the UHF RFID device reader switch position – RFID or pistol only mode.
Syntax	**public int GetRFIDSwitchStatus()** *If TriggerSwitchMode = false, SwitchMode (SetSwitchMode) cannot be set. *If TriggerSwitchMode = true, RFIDSwitchStatus (SetRFIDSwitchStatus) cannot be set.

Example	`int status = mRfidManager.GetRFIDSwitchStatus();`
	`if(status == -1) {`
	`String m = mRfidManager.GetLastError();`
	`Log.e(TAG, "GetLastError = " + m); }`

Return Value	If successful, it returns the Status. True: Switch is set to RFID False: Switch is set to pistol only. Otherwise, it returns -1.
See Also	SetRFIDSwitchStatus

SetRFIDSwitchStatus	
Purpose	Sets the UHF RFID device reader switch position – RFID or pistol only mode.
Syntax	**public int SetRFIDSwitchStatus()** *If TriggerSwitchMode = false, SwitchMode (SetSwitchMode) cannot be set. *If TriggerSwitchMode = true, RFIDSwitchStatus (SetRFIDSwitchStatus) cannot be set.

Example	`int re = mRfidManager.SetRFIDSwitchStatus(false);`
	`if(re!=ClResult.S_OK.ordinal()){`
	`String err = mRfidManager.GetLastError();`
	`Log.e(TAG, "SetRFIDSwitchStatus (err) = " + err); }`

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	GetRFIDSwitchStatus

GetScanMode

Purpose	Gets the current scan mode.
Syntax	**public ScanMode GetScanMode()**
Parameters	A default value comes with an asterisk "*". ScanMode *Mode*

ScanMode.Alternate	Press the trigger key once to have the reader keep reading tags. Press the trigger key again to stop reading.
ScanMode.Single	To read tag once per trigger. 1. Condition to start the operation: Press and hold the trigger key. 2. Conditions to stop the operation: (1) A tag is read (2) Trigger key is released. (3) "Scan Session Timeout" expires while no tag data is received (4) New scan mode is set. 3. Release trigger key and press it again to start a new scan session. Scan Session Timeout will be refreshed.
ScanMode.Test	Do not use
ScanMode.Continuous	* To Read tag continuously as trigger key is persistently pressed. 1. Condition to start the operation: Press and hold the trigger key. 2. Conditions to stop the operation: (1) Trigger key is released. (2) New scan mode is set.

Example

```
ScanMode mode =  mRfidManager.GetScanMode();
if(mode == ScanMode.Err) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value	If successful, it returns the ScanMode. Otherwise, it returns ScanMode.Err.
See Also	SetScanMode

SetScanMode

Purpose	Changes the current scan mode.
Syntax	**public int SetScanMode(ScanMode mode)**
Parameters	A default value comes with an asterisk "*". ScanMode *Mode*

ScanMode.Alternate	Press the trigger key once to have the reader keep reading tags. Press the trigger key again to stop reading.
ScanMode.Single	To read tag once per trigger. 1. Condition to start the operation: Press and hold the trigger key. 2. Conditions to stop the operation: (1) A tag is read (2) Trigger key is released. (3) "Scan Session Timeout" expires while no tag data is received (4) New scan mode is set. 3. Release trigger key and press it again to start a new scan session. Scan Session Timeout will be refreshed.
ScanMode.Test	Do not use
ScanMode.Continuous	* To Read tag continuously as trigger key is persistently pressed. 1. Condition to start the operation: Press and hold the trigger key. 2. Conditions to stop the operation: (1) Trigger key is released. (2) New scan mode is set.

Example

```
int re = mRfidManager.SetScanMode(ScanMode.Single);
if(re!=ClResult.S_OK.ordinal()){
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	GetScanMode

GetRFIDMode

Purpose	Gets the RFID scan mode of the UHF RFID device.
Syntax	**public RFIDMode GetRFIDMode()**
Parameters	A default value comes with an asterisk "*". RFIDMode *Mode*

RFIDMode.Inventory	*Inventory operation that reads RFID EPC data
RFIDMode.ReadTag	Operation that reads data from the selected memory bank
RFIDMode.WriteTag	Operation that writes data to the selected memory bank
RFIDMode.Inventory_EPC_TID	Inventory operation that reads RFID EPC and TID data

Example

```
RFIDMode mode = mRfidManager.GetRFIDMode();
if(mode==RFIDMode.Err) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value: If successful, it returns the RFIDMode.
Otherwise, it returns RFIDMode.Err.

Intent Data

Inventory	PC EPC , EPC length RSSI
ReadTag	PC EPC , EPC length ReadData , ReadData length
WriteTag	EPC , EPC length
Inventory_EPC_TID	PC EPC , EPC length TID , TID length RSSI

See Also: GeneralString.Intent_RFIDSERVICE_TAG_DATA
SetRFIDMode

SetRFIDMode	
Purpose	Sets the RFID scan mode of the UHF RFID device.
Syntax	**public int SetRFIDMode (RFIDMode mode)**
Parameters	A default value comes with an asterisk "*". RFIDMode *Mode*

RFIDMode.Inventory	* Inventory operation that reads RFID EPC data
RFIDMode.ReadTag	Operation that reads data from the selected memory bank
RFIDMode.WriteTag	Operation that writes data to the selected memory bank
RFIDMode.Inventory_EPC_TID	Inventory operation that reads RFID EPC and TID data

Example

```
int re = mRfidManager.SetRFIDMode(RFIDMode.Inventory_EPC_TID);
if(re!=ClResult.S_OK.ordinal()){
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value: If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

Intent Data

Inventory	PC EPC , EPC length RSSI
ReadTag	PC EPC , EPC length ReadData , ReadData length
WriteTag	EPC , EPC length
Inventory_EPC_TID	PC EPC , EPC length TID , TID length RSSI

See Also: GeneralString.Intent_RFIDSERVICE_TAG_DATA
GetRFIDMode

GetSelectedMemoryBank

Purpose	Gets the current selected memory bank.
Syntax	**public RFIDMemoryBank GetSelectedMemoryBank()**
Parameters	A default value comes with an asterisk "*". RFIDMemoryBank *MemoryBabk*

RFIDMemoryBank.Reserved	Reserved bank
RFIDMemoryBank.EPC	*EPC bank
RFIDMemoryBank.TID	TID bank
RFIDMemoryBank.User	User bank

Example

```
RFIDMemoryBank bank = mRfidManager.GetSelectedMemoryBank();
if (bank==RFIDMemoryBank.Err) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value	If successful, it returns the RFIDMemoryBank. Otherwise, it returns RFIDMemoryBank.Err.
See Also	SelectMemoryBank

SelectMemoryBank

Purpose	Select a memory bank to read or write.
Syntax	**public int SelectMemoryBank(RFIDMemoryBank bank)**
Parameters	A default value comes with an asterisk "*". RFIDMemoryBank *MemoryBabk*

RFIDMemoryBank.Reserved	Reserved bank
RFIDMemoryBank.EPC	*EPC bank
RFIDMemoryBank.TID	TID bank
RFIDMemoryBank.User	User bank

Example

```
int re = mRfidManager.SelectMemoryBank(RFIDMemoryBank.TID);
if(re!=ClResult.S_OK.ordinal()){
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	GetSelectedMemoryBank

GetTxPower

Purpose	Gets the Tx power. Ranges from 5~30. Default=27.
Syntax	**public int GetTxPower()**

Example

```
int tx = mRfidManager.GetTxPower();
if(tx==-1) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value	If successful, it returns the tx power. Otherwise, it returns -1
See Also	SetTxPower

SetTxPower

Purpose	Sets the Tx power. Ranges from 5~30. Default=27.
Syntax	**public int GetTxPower()**

Example

```
int re = mRfidManager.SetTxPower(3);
if(re!=ClResult.S_OK.ordinal()){
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	GetTxPower

RFIDDirectStartInventoryRound

Purpose	Automatically triggers a RFID inventory round on the UHF RFID device. Trigger-press is not required.
Syntax	**public int RFIDDirectStartInventoryRound(InventoryType type, int count)**

Parameters

A default value comes with an asterisk “*”.

InventoryType *type*

InventoryType.EPC	*EPC only
InventoryType.EPC_AND_TID	Both EPC and TID

int *count*

count	Number of reads in one inventory round. Ranges from 1~254

Example

```
int re =
mRfidManager.RFIDDirectStartInventoryRound(InventoryType.EPC_AND_TI
D ,100);
if(re!= ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
Intent Data	

EPC	PC EPC , EPC length RSSI
EPC_AND_TID	PC EPC , EPC length TID , TID length RSSI

See Also	GeneralString.Intent_RFIDSERVICE_TAG_DATA RFIDDirectCancelInventoryRound

RFIDDirectCancelInventoryRound

Purpose: Stops the UHF RFID device from tag scanning during an inventory round.

Syntax: **public int RFIDDirectCancelInventoryRound()**

Example:

```
int re = mRfidManager.RFIDDirectCancelInventoryRound();
if(re!=ClResult.S_OK.ordinal()){
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value: If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also: RFIDDirectStartInventoryRound

RFIDReadTagMassive

Purpose: Prepares the UHF RFID device for trigger-pressed read.

Syntax: **public int RFIDReadTagMassive(byte[] password, RFIDMemoryBank bank, int start, int length)**

Parameters:

byte[] *password*
Tag access password.
RFIDMemoryBank *bank*

RFIDMemoryBank.Reserved	Reserved bank
RFIDMemoryBank.EPC	EPC bank
RFIDMemoryBank.TID	TID bank
RFIDMemoryBank.User	User bank

int *Start*
Start byte position in bank memory.
int *length*
Number of bytes to read.

Example

```
int re = mRfidManager.RFIDReadTagMassive(null, RFIDMemoryBank.User,
0,0);
if(re!=ClResult.S_OK.ordinal()){
String err = mRfidManager.GetLastError();
Log.e(TAG, "RFIDReadTagMassive (re) = " + err); }
```

Return Value

If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also

GetRFIDMode, SetRFIDMode,RFIDWriteTagMassive

RFIDWriteTagMassive

Purpose

Prepares the UHF RFID device for trigger-pressed write.

Syntax

public int RFIDWriteTagMassive(byte[] password, RFIDMemoryBank bank, byte[] data, int start, int length)

Parameters

byte[] *password*
Tag access password.
RFIDMemoryBank *bank*

RFIDMemoryBank.Reserved	Reserved bank
RFIDMemoryBank.EPC	EPC bank
RFIDMemoryBank.TID	TID bank
RFIDMemoryBank.User	User bank

byte[] *data*
The data to write (max. 32 bytes allowed).
int *Start*
Start byte position in bank memory.
int *length*
Number of bytes to write.

Example

```
String Data = "abcd";
int re = mRfidManager.RFIDWriteTagMassive(null, RFIDMemoryBank.User,
Data.getBytes(), 0, 4);
if(re!=ClResult.S_OK.ordinal()){
String err = mRfidManager.GetLastError();
Log.e(TAG, "RFIDWriteTagMassive (err) = " + err); }
```

Return Value

If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also

GetRFIDMode, SetRFIDMode,RFIDReadTagMassive

RFIDDirectReadTagByEPC

Purpose	Automatically triggers a RFID tag read on the UHF RFID device with a given EPC. Trigger-press is not required.
Syntax	**public int RFIDDirectReadTagByEPC(byte[] password, byte[] epc, RFIDMemoryBank bank, int start, int length, int retry)**

Parameters

byte[] *password*

Tag access password.

byte[] *epc*

EPC used to pick out the tag to read.

RFIDMemoryBank *bank*

RFIDMemoryBank.Reserved	Reserved bank
RFIDMemoryBank.EPC	EPC bank
RFIDMemoryBank.TID	TID bank
RFIDMemoryBank.User	User bank

int *Start*

Start byte position in bank memory.

int *length*

Number of bytes to read (max. 32 bytes allowed).

int *retry*

Number of retries if initial read fails.

Example

```
byte[] EPCByteArray = new byte[] {  (byte)0xe2, (byte)0x00, (byte) 0x30,
(byte) 0x98, (byte)0x06, (byte)0x02, (byte)0x01, (byte)0x98,
(byte)0x06, (byte)0x50, (byte)0xd7, (byte)0x4f };
int re = mRfidManager.RFIDDirectReadTagByEPC(null, EPCByteArray,
RFIDMemoryBank.User, 0, 0, 3);
if(re!=ClResult.S_OK.ordinal()){
String err = mRfidManager.GetLastError();
Log.e(TAG, "RFIDDirectReadTagByEPC (err) = " + err); }
```

Return Value

If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

Intent Data

EPC Data	ReadData , ReadData length

See Also

GeneralString.Intent_RFIDSERVICE_TAG_DATA
RFIDDirectWriteTagByEPC

RFIDDirectReadTagByTID

Purpose	Automatically triggers a RFID tag read on the UHF RFID device with a given TID. Trigger-press is not required.
Syntax	**public int RFIDDirectReadTagByTID(byte[] password, byte[] tid, RFIDMemoryBank bank, int start, int length, int retry)**

Parameters

byte[] *password*

Tag access password.

byte[] *tid*

TID used to pick out the tag to read.

RFIDMemoryBank *bank*

RFIDMemoryBank.Reserved	Reserved bank
RFIDMemoryBank.EPC	EPC bank
RFIDMemoryBank.TID	TID bank
RFIDMemoryBank.User	User bank

int *Start*

Start byte position in bank memory.

int *length*

Number of bytes to read (max. 32 bytes allowed).

int *retry*

Number of retries if initial read fails.

Example

```
byte[] TIDByteArray = new byte[] { (byte)0xe2, (byte)0x00, (byte)0x34,
(byte)0x12, (byte)0x01, (byte)0x72, (byte)0xfa, (byte)0x00,
(byte)0x02, (byte)0x34, (byte)0xd7, (byte)0x4f };
int re = mRfidManager.RFIDDirectReadTagByTID(null, TIDByteArray,
RFIDMemoryBank.User, 0, 4, 3);
if(re!=ClResult.S_OK.ordinal()){
String err = mRfidManager.GetLastError();
Log.e(TAG, "RFIDDirectReadTagByTID (err) = " + err); }
```

Return Value

If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

Intent Data

TID Data	ReadData , ReadData length

See Also

GeneralString.Intent_RFIDSERVICE_TAG_DATA
RFIDDirectWriteTagByTID

RFIDDirectWriteTagByEPC

Purpose	Automatically triggers an RFID tag write on the UHF RFID device with a given EPC.
Syntax	**public DeviceResponse RFIDDirectWriteTagByEPC(byte[] password, byte[] epc, RFIDMemoryBank bank, int start, int retry, byte[] data)**

Parameters

byte[] *password*

Tag access password.

byte[] *epc*

EPC used to pick out the tag to write.

RFIDMemoryBank *bank*

RFIDMemoryBank.Reserved	Reserved bank
RFIDMemoryBank.EPC	EPC bank
RFIDMemoryBank.TID	TID bank
RFIDMemoryBank.User	User bank

int *Start*

Start byte position in bank memory.

int re*try*

Number of retries if initial read fails.

byte[] *data*

The data to write (max. 32 bytes allowed).

DeviceResponse *Response*

DeviceResponse.OperationSuccess	Operation succeeded.
DeviceResponse.OperationFail	Operation failed.
DeviceResponse.OperationFinish	Operation finish.
DeviceResponse.DeviceTimeOut	UHF RFID device timed out.
DeviceResponse.DeviceBusy	UHF RFID device is busy.

Example

```
byte[] DataArray = new byte[] { (byte)0x61, (byte)0x61, (byte)0x61,
(byte)0x61 };
byte[] EPCByteArray = new byte[] {  (byte)0xe2, (byte)0x00, (byte) 0x30,
(byte) 0x98, (byte)0x06, (byte)0x02, (byte)0x01, (byte)0x98,
(byte)0x06, (byte)0x50, (byte)0xd7, (byte)0x4f };
DeviceResponse re = mRfidManager.RFIDDirectWriteTagByEPC(null,
EPCByteArray, RFIDMemoryBank.User, 0, 1, DataArray);
```

Return Value: It returns DeviceResponse.

See Also: RFIDDirectReadTagByTID

RFIDDirectWriteTagByTID

Purpose	Automatically triggers an RFID tag write on the UHF RFID device with a given TID.
Syntax	**public DeviceResponse RFIDDirectWriteTagByTID(byte[] password, byte[] tid, RFIDMemoryBank bank, int start, int retry, byte[] data)**

Parameters

byte[] *password*

Tag access password.

byte[] *tid*

TID used to pick out the tag to write.

RFIDMemoryBank *bank*

RFIDMemoryBank.Reserved	Reserved bank
RFIDMemoryBank.EPC	EPC bank
RFIDMemoryBank.TID	TID bank
RFIDMemoryBank.User	User bank

int *Start*

Start byte position in bank memory.

int *retry*

Number of retries if initial read fails.

byte[] *data*

The data to write (max. 32 bytes allowed).

DeviceResponse *Response*

DeviceResponse.OperationSuccess	Operation succeeded.
DeviceResponse.OperationFail	Operation failed.
DeviceResponse.OperationFinish	Operation finish.
DeviceResponse.DeviceTimeOut	UHF RFID device timed out.
DeviceResponse.DeviceBusy	UHF RFID device is busy.

Example

```
byte[] DataArray = new byte[] { (byte)0x31, (byte)0x32, (byte)0x33,
(byte)0x34 };

byte[] TIDByteArray = new byte[] { (byte)0xe2, (byte)0x00, (byte)0x34,
(byte)0x12, (byte)0x01, (byte)0x72, (byte)0xfa, (byte)0x00,
(byte)0x02, (byte)0x34, (byte)0xd7, (byte)0x4f };

DeviceResponse re = mRfidManager.RFIDDirectWriteTagByTID(null,
TIDByteArray, RFIDMemoryBank.User, 0, 3, DataArray);
```

Return Value	It returns DeviceResponse.
See Also	RFIDDirectReadTagByTID

RFIDDirectKillTag

Purpose	Automatically triggers the UHF RFID device to kill a RFID tag with a given EPC. Trigger-press is not required.
Syntax	**public DeviceResponse RFIDDirectKillTag(byte[] password, byte[] epc)**
Parameters	byte[] *password* Tag kill password. byte[] *epc* EPC used to pick out the tag to kill. DeviceResponse *Response*

DeviceResponse.OperationSuccess	Operation succeeded.
DeviceResponse.OperationFail	Operation failed.
DeviceResponse.OperationFinish	Operation finish.
DeviceResponse.DeviceTimeOut	UHF RFID device timed out.
DeviceResponse.DeviceBusy	UHF RFID device is busy.

Example

```
byte[] password = new byte[] { (byte)0x31, (byte)0x32, (byte)0x33,
(byte)0x34 };
byte[] EPCByteArray = new byte[] {  (byte)0xe2, (byte)0x00, (byte) 0x30,
(byte) 0x98, (byte)0x06, (byte)0x02, (byte)0x01, (byte)0x98,
(byte)0x06, (byte)0x50, (byte)0xd7, (byte)0x4f };
DeviceResponse re = mRfidManager.RFIDDirectKillTag(password,
EPCByteArray);
Log.i(TAG, "RFIDDirectKillTag(re) = " + re);
```

Return Value It returns DeviceResponse.

RFIDDirectUnlockTag	
Purpose	Automatically triggers the UHF RFID device to unlock a RFID tag memory bank with a given EPC. Trigger-press is not required.
Syntax	**public DeviceResponse RFIDDirectUnlockTag(byte[] password, byte[] epc, LockTarget targetBank)**
Parameters	byte[] *password* Tag access password. byte[] *epc* EPC used to pick out the tag to unlock. LockTarget *targetBank* Tag memory bank which will be unlocked

LockTarget.KillPassword	First 32 bit of Reserved bank
LockTarget.AccessPassword	Last 32 bit of Reserved bank
LockTarget.EPCBank	EPC bank
LockTarget.TIDBank	TID bank
LockTarget.UserBank	User bank

DeviceResponse *Response*

DeviceResponse.OperationSuccess	Operation succeeded.
DeviceResponse.OperationFail	Operation failed.
DeviceResponse.OperationFinish	Operation finish.
DeviceResponse.DeviceTimeOut	UHF RFID device timed out.
DeviceResponse.DeviceBusy	UHF RFID device is busy.

Example

```
byte[] password = new byte[] { (byte)0x31, (byte)0x32, (byte)0x33,
(byte)0x34 };
byte[] EPCByteArray = new byte[] {  (byte)0xe2, (byte)0x00, (byte) 0x30,
(byte) 0x98, (byte)0x06, (byte)0x02, (byte)0x01, (byte)0x98,
(byte)0x06, (byte)0x50, (byte)0xd7, (byte)0x4f };
DeviceResponse re = mRfidManager.RFIDDirectUnlockTag(password,
EPCByteArray , LockTarget.TIDBank);
Log.i(TAG, "RFIDDirectUnlockTag(re) = " + re);
```

Return Value	It returns DeviceResponse.
See Also	RFIDDirectLockTag

RFIDDirectLockTag	
Purpose	Automatically triggers the UHF RFID device to lock a RFID tag memory bank with a given EPC. Trigger-press is not required.
Syntax	**public DeviceResponse RFIDDirectLockTag(byte[] password, byte[] epc, LockTarget targetBank)**
Parameters	byte[] *password* Tag access password. byte[] *epc* EPC used to pick out the tag to lock. LockTarget *targetBank* Tag memory bank which will be locked

LockTarget.KillPassword	First 32 bit of Reserved bank
LockTarget.AccessPassword	Last 32 bit of Reserved bank
LockTarget.EPCBank	EPC bank
LockTarget.TIDBank	TID bank
LockTarget.UserBank	User bank

DeviceResponse *Response*

DeviceResponse.OperationSuccess	Operation succeeded.
DeviceResponse.OperationFail	Operation failed.
DeviceResponse.OperationFinish	Operation finish.
DeviceResponse.DeviceTimeOut	UHF RFID device timed out.
DeviceResponse.DeviceBusy	UHF RFID device is busy.

Example

```
byte[] password = new byte[] { (byte)0x31, (byte)0x32, (byte)0x33,
(byte)0x34 };
byte[] EPCByteArray = new byte[] {  (byte)0xe2, (byte)0x00, (byte) 0x30,
(byte) 0x98, (byte)0x06, (byte)0x02, (byte)0x01, (byte)0x98,
(byte)0x06, (byte)0x50, (byte)0xd7, (byte)0x4f };
DeviceResponse re = mRfidManager.RFIDDirectLockTag (password,
EPCByteArray , LockTarget.TIDBank);
Log.i(TAG, " RFIDDirectLockTag (re) = " + re);
```

Return Value	It returns DeviceResponse.
See Also	RFIDDirectUnlockTag

RFIDDirectPermanentLockTag

Purpose	Automatically triggers the UHF RFID device to permanently lock a RFID tag memory bank with a given EPC. Trigger-press is not required.
Syntax	**public DeviceResponse RFIDDirectPermanentLockTag(byte[] password, byte[] epc, LockTarget targetBank)**

Parameters

byte[] password

Tag access password.

byte[] *epc*

EPC used to pick out the tag to permanently lock.

LockTarget *targetBank*

Tag memory bank which will be permanently locked

LockTarget.KillPassword	First 32 bit of Reserved bank
LockTarget.AccessPassword	Last 32 bit of Reserved bank
LockTarget.EPCBank	EPC bank
LockTarget.TIDBank	TID bank
LockTarget.UserBank	User bank

DeviceResponse *Response*

DeviceResponse.OperationSuccess	Operation succeeded.
DeviceResponse.OperationFail	Operation failed.
DeviceResponse.OperationFinish	Operation finish.
DeviceResponse.DeviceTimeOut	UHF RFID device timed out.
DeviceResponse.DeviceBusy	UHF RFID device is busy.

Example

```
byte[] password = new byte[] { (byte)0x31, (byte)0x32, (byte)0x33, (byte)0x34 };
byte[] EPCByteArray = new byte[] {  (byte)0xe2, (byte)0x00, (byte) 0x30, (byte) 0x98, (byte)0x06, (byte)0x02, (byte)0x01, (byte)0x98, (byte)0x06, (byte)0x50, (byte)0xd7, (byte)0x4f };
DeviceResponse re = mRfidManager.RFIDDirectPermanentLockTag (password, EPCByteArray , LockTarget.TIDBank);
Log.i(TAG, " RFIDDirectPermanentLockTag (re) = " + re);
```

Return Value	It returns DeviceResponse.
See Also	RFIDDirectLockTag

GetIncludedEPCFilter

Purpose	Gets the EPC included filter settings of the GHF RFID device.
Syntax	**public int GetIncludedEPCFilter(RfidEpcFilter epcfilter)**
Parameters	**RfidEpcFilter** *epcfilter*

Enable	byte A value indicating whether the UHF RFID device's RFIDWriteTagMassive function is enabled. 0 – disable (default), 1 – enable, 2 – enable filter rangeing from EPCPattern1 ~ EPCPattern2.
Startbit_MSB	byte Start bit of EPC
Startbit_LSB	byte Start bit of EPC
PatternLength_MSB	byte The pattern length in bit
PatternLength_LSB	byte The pattern length in bit
Scheme	byte EPC encoding scheme to filter
EPCPattern1	String The value of the EPC first partition to be filtered.
EPCPattern2	String The start value of the EPC second partition to be filtered.

* Startbit + PatternLength must be less than or equal to 256.

* EPCPatternX: Included EPC pattern, max=32 Byte. Bits of 0b will be padded to the right (LSB) of pattern if PatternLength is not multiple of 8.

*If [ENABLE] is set to 2, EPCPattern2 is needed. It indicates only the tag which EPC is in the range of EPCPattern1 ~ EPCPattern2 will be accepted (EPC Pattern1 ≤ Tag EPC ≤ EPC Pattern2). The length of EPCPattern2 is equal to EPCPattern1.

Example

```
RfidEpcFilter f = new RfidEpcFilter();
int re = mRfidManager.GetIncludedEPCFilter(f);
if(re!=ClResult.S_OK.ordinal()){
String err = mRfidManager.GetLastError();
Log.e(TAG, "GetIncludedEPCFilter (err) = " + err); }
```

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	SetIncludedEPCFilter

SetIncludedEPCFilter	
Purpose	Sets the EPC included filter settings of the GHF RFID device.
Syntax	**public int SetIncludedEPCFilter(RfidEpcFilter epcfilter)**
Parameters	RfidEpcFilter *epcfilter*

Enable	byte A value indicating whether the UHF RFID device's EPC filter function is enabled. 0 – disable (default), 1 – enable, 2 – enable filter rangeing from EPCPattern1 ~ EPCPattern2.
Startbit_MSB	byte Start bit of EPC
Startbit_LSB	byte Start bit of EPC
PatternLength_MSB	byte The pattern length in bit
PatternLength_LSB	byte The pattern length in bit
Scheme	byte EPC encoding scheme to filter
EPCPattern1	String The value of the EPC first partition to be filtered.
EPCPattern2	String The start value of the EPC second partition to be filtered.

* Startbit + PatternLength must be less than or equal to 256

* EPCPatternX: Included EPC pattern, max=32 Bytes. Bits of 0b will be padded to the right (LSB) of pattern if PatternLength is not multiple of 8.

*If [ENABLE] is set to 2, EPCPattern2 is needed. It indicates only the tag which EPC is in the range of EPCPattern1~ EPCPattern2 will be accepted (EPC Pattern1 ≤ Tag EPC ≤ EPC Pattern2). The length of EPCPattern2 is equal to EPCPattern1.

Example	`RfidEpcFilter f = new RfidEpcFilter();`
	`f.Enable = 1;`
	`f.Startbit_LSB = ((byte)(0x08));`
	`f.Startbit_MSB = ((byte)(0));`
	`f.EPCPattern1 = "00";`
	`f.EPCPattern2 ="";`
	`f.PatternLength_LSB = ((byte)(0x03));`
	`f.PatternLength_MSB = ((byte)(0));`
	`f.Scheme = (byte) 0x30;`
	`int re = mRfidManager.SetIncludedEPCFilter(f);`
	`if(re != ClResult.S_OK.ordinal()){`
	`Log.e(TAG, "SetIncludedEPCFilter(err) = " + mRfidManager.GetLastError());}`

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	GetIncludedEPCFilter

GetExcludedEPCFilter	
Purpose	Gets the EPC excluded filter settings of the GHF RFID device.
Syntax	**public int GetExcludedEPCFilter (RfidEpcFilter epcfilter)**
Parameters	RfidEpcFilter *epcfilter*

Enable	byte A value indicating whether the UHF RFID device's EPC filter function is enabled. 0 – disable (default), 1 – enable, 2 – enable filter rangeing from EPCPattern1 ~ EPCPattern2.
Startbit_MSB	byte Start bit of EPC
Startbit_LSB	byte Start bit of EPC
PatternLength_MSB	byte The pattern length in bit
PatternLength_LSB	byte The pattern length in bit
Scheme	byte EPC encoding scheme to filter
EPCPattern1	String The value of the EPC first partition to be filtered.
EPCPattern2	String The start value of the EPC second partition to be filtered.

* Startbit + PatternLength must be less than or equal to 256

* EPCPatternX: Included EPC pattern, max=32 Bytes. Bits of 0b will be padded to the right (LSB) of pattern if PatternLength is not multiple of 8.

*If [ENABLE] is set to 2, EPCPattern2 is needed. It indicates only the tag which EPC is in the range of EPCPattern1 ~ EPCPattern2 will be accepted (EPC Pattern1 ≤ Tag EPC ≤ EPC Pattern2). The length of EPCPattern2 is equal to EPCPattern1.

Example

```
RfidEpcFilter f = new RfidEpcFilter();
int re = mRfidManager.GetExcludedEPCFilter(f);
if(re!=ClResult.S_OK.ordinal()){
String err = mRfidManager.GetLastError();
Log.e(TAG, "GetExcludedEPCFilter (err) = " + err); }
```

Return Value — If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also — SetExcludedEPCFilter

SetExcludedEPCFilter	
Purpose	Sets the EPC excluded filter settings of the GHF RFID device.
Syntax	**public int SetExcludedEPCFilter(RfidEpcFilter epcfilter)**
Parameters	RfidEpcFilter *epcfilter*

Enable	byte A value indicating whether the UHF RFID device's EPC filter function is enabled. 0 – disable (default), 1 – enable, 2 – enable filter rangeing from EPCPattern1~ EPCPattern2.
Startbit_MSB	byte Start bit of EPC
Startbit_LSB	byte Start bit of EPC
PatternLength_MSB	byte The pattern length in bit
PatternLength_LSB	byte The pattern length in bit
Scheme	byte EPC encoding scheme to filter
EPCPattern1	String The value of the EPC first partition to be filtered.
EPCPattern2	String The start value of the EPC second partition to be filtered.

* Startbit + PatternLength must be less than or equal to 256

* EPCPatternX: Included EPC pattern, max=32 Bytes. Bits of 0b will be padded to the right (LSB) of pattern if PatternLength is not multiple of 8.

*If [ENABLE] is set to 2, EPCPattern2is needed. It indicates only the tag which EPC is in the range of EPCPattern1~ EPCPattern2 will be accepted (EPC Pattern1 ≤ Tag EPC ≤ EPC Pattern2). The length of EPCPattern2 is equal to EPCPattern1.

Example	`RfidEpcFilter f = new RfidEpcFilter();`

```
RfidEpcFilter f = new RfidEpcFilter();
f.Enable =1;
f.Startbit_LSB = ((byte)(0x08));
f.Startbit_MSB = ((byte)(0));
f.EPCPattern1 = "00000050";
f.EPCPattern2 = "";
f.PatternLength_LSB = ((byte)(0x1c));
f.PatternLength_MSB = ((byte)(0));
f.Scheme = (byte) 0x35;
int re = mRfidManager.SetExcludedEPCFilter(f);
if(re!=ClResult.S_OK.ordinal()){
String err = mRfidManager.GetLastError();
Log.e(TAG, "SetExcludedEPCFilter (err) = " + err); }
```

Return Value If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also GetExcludedEPCFilter

GetWorkMode

Purpose Gets the current work mode.

Syntax **public WorkMode GetWorkMode()**

Parameters A default value comes with an asterisk “*”.
WorkMode *Mode*

WorkMode.ComprehensiveMode	*Comprehensive mode
WorkMode.MultiTagMode	Multi-tag mode
WorkMode.UserDefine1	user define1
WorkMode.UserDefine2	user define2
WorkMode.UserDefine3	user define3
WorkMode.UserDefine4	user define4
WorkMode.UserDefine5	user define5

Remarks The WorkMode.MultiTagMode parameter is not available when the RFID device region is set to Japan.

Example

```
WorkMode mode =  mRfidManager.GetWorkMode();
if(mode == WorkMode.Err) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value If successful, it returns the WorkMode.
Otherwise, it returns WorkMode.Err.

See Also SetWorkMode,

SetWorkMode

Purpose	Sets the current work mode.
Syntax	**public int SetWorkMode (WorkMode mode)**
Parameters	A default value comes with an asterisk "*". WorkMode *Mode*

WorkMode.ComprehensiveMode	*Comprehensive mode
WorkMode.MultiTagMode	Multi-tag mode
WorkMode.UserDefine1	user define1
WorkMode.UserDefine2	user define2
WorkMode.UserDefine3	user define3
WorkMode.UserDefine4	user define4
WorkMode.UserDefine5	user define5

Remarks: The WorkMode.MultiTagMode parameter is not available when the RFID device region is set to Japan.

Example

```
int re = mRfidManager.SetWorkMode(WorkMode.UserDefine3);
if(re!=ClResult.S_OK.ordinal()){
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value: If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also: GetWorkMode,

GetQValue

Purpose	Gets the Q value.
Syntax	**public QValue GetQValue()**
Parameters	QValue *qvalue*

Dynamic	False=fixed; true=dynamic.
value	The Q value. Ranges from 0~15.
Min	Minimum Q value. Ranges from 0~15.
Max	Maximum Q value. Ranges from 0~15.

Note: Q value is dependent to work mode, each work mode has its own Q value parameter

Example

```
QValue q = mRfidManager.GetQValue();
if(q==null) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value: If successful, it returns the Qvalue.
Otherwise, it returns `null`.

See Also: SetQValue,

SetQValue

Purpose	Sets the Q value.
Syntax	**public int SetQValue(QValue q)**
Parameters	QValue *qvalue*

Dynamic	False=fixed; true=dynamic.
value	The Q value. Ranges from 0~15.
Min	Minimum Q value. Ranges from 0~15.
Max	Maximum Q value. Ranges from 0~15.

Note: Q value is dependent to work mode, each work mode has its own Q value parameter

Example

```
QValue q =new QValue();
q.Dynamic=false;
q.value=5;
q.Min = 4;
q.Max = 15;
int re = mRfidManager.SetQValue(q);
if(re!=ClResult.S_OK.ordinal()){
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value: If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also: GetQValue,

GetRFLink

Purpose	Gets the current RF Link.
Syntax	**public RFLink GetRFLink()**
Parameters	A default value comes with an asterisk "*". RFLink *rflink*

RFLink.DSB_ASK_FM0_40KHz	DSB_ASK /FM0/ 40 KHz
RFLink.PR_ASK_Miller4_250KHz	PR_ASK /Miller4/ 250KHz
RFLink.PR_ASK_Miller4_300KHz	PR_ASK /Miller4/ 300KHz
RFLink.DSB_ASK_FM0_400KHz	* DSB_ASK /FM0/ 400KHz
RFLink.PR_ASK_Miller2_250KHz	PR_ASK /Miller2/ 250KHz
RFLink.PR_ASK_FM0_250KHz	PR_ASK /FM0/ 250KHz

Remarks: RF Link is dependent on work modes; each work mode has its own RF Link parameter.

RFLink.DSB_ASK_FM0_40KHz and *RFLink.DSB_ASK_FM0_400KHz* parameters are not available when the RFID device region is set to Japan.

Example

```
RFLink link =  mRfidManager.GetRFLink();
if(link == RFLink.Err) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value — If successful, it returns the RFLink.
Otherwise, it returns `RFLink.Err`.

See Also — SetRFLink,

SetRFLink

Purpose — Sets the current RF Link.

Syntax — **public int SetRFLink (RFLink link)**

Parameters — A default value comes with an asterisk "*".
RFLink *rflink*

RFLink.DSB_ASK_FM0_40KHz	DSB_ASK /FM0/ 40 KHz
RFLink.PR_ASK_Miller4_250KHz	PR_ASK /Miller4/ 250KHz
RFLink.PR_ASK_Miller4_300KHz	PR_ASK /Miller4/ 300KHz
RFLink.DSB_ASK_FM0_400KHz	* DSB_ASK /FM0/ 400KHz
RFLink.PR_ASK_Miller2_250KHz	PR_ASK /Miller2/ 250KHz
RFLink.PR_ASK_FM0_250KHz	PR_ASK /FM0/ 250KHz

Remarks — RF Link is dependent on work modes; each work mode has its own RF Link parameter.
RFLink.DSB_ASK_FM0_40KHz and *RFLink.DSB_ASK_FM0_400KHz* parameters are not available when the RFID device region is set to Japan.

Example

```
int re = mRfidManager.SetRFLink(RFLink.PR_ASK_Miller4_300KHz);
if(re!=ClResult.S_OK.ordinal()){
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value — If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also — GetRFLink,

GetGen2	
Purpose	Gets the current Gen2.
Syntax	**public int GetGen2(Gen2Settings settings)**
Parameters	A default value comes with an asterisk "*". Gen2Settings *settings*

SessionSettings	Session data SessionSettings.S0 *SessionSettings.S1 SessionSettings.S2 SessionSettings.S3
InventoryStatusSettings	Inventory status *InventoryStatusSettings.STATE_A InventoryStatusSettings.STATE_B InventoryStatusSettingsAB_FLIP
SLFlagSettings	SL Flag *SLFlagSettings.All SLFlagSettings.Deasserted SLFlagSettings.Asserted

Note: RFID Gen2 is dependent to work mode, each work mode has its own RFID Gen2 parameter.

Example

```
Gen2Settings settings = new Gen2Settings();
int re = mRfidManager.GetGen2(settings);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value: If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also: SetGen2,

SetGen2

Purpose	Sets the current Gen2.
Syntax	**public int SetGen2(Gen2Settings settings)**
Parameters	A default value comes with an asterisk "*". Gen2Settings *settings*

SessionSettings	Session data SessionSettings.S0 * SessionSettings.S1 SessionSettings.S2 SessionSettings.S3
InventoryStatusSettings	Inventory status *InventoryStatusSettings.STATE_A InventoryStatusSettings.STATE_B InventoryStatusSettingsAB_FLIP
SLFlagSettings	SL Flag *SLFlagSettings.All SLFlagSettings.Deasserted SLFlagSettings.Asserted

Note: RFID Gen2 is dependent to work mode, each work mode has its own RFID Gen2 parameter.

Example

```
Gen2Settings settings = new Gen2Settings();
settings.Session = SessionSettings.S1;
settings.InventoryStatus_Action = InventoryStatusSettings.AB_FLIP;
settings.SL_Flag =SLFlagSettings.Asserted;
int re = mRfidManager.SetGen2(settings);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	GetGen2

GetAllQValue

Purpose	Gets the all Q value.
Syntax	**public int GetAllQValue(AllQValue allq)**
Parameters	AllQValue ArrayList<QValue> *allqvalue*

Dynamic	False=fixed; true=dynamic.
value	The Q value. Ranges from 0~15.
Min	Minimum Q value. Ranges from 0~15.
Max	Maximum Q value. Ranges from 0~15.

Note: Q value is dependent to work mode, each work mode has its own Q value parameter.

Example

```
AllQValue q = new AllQValue();
int re = mRfidManager.GetAllQValue(q);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
for(int i=0; i<8 ;i++)
Log.i(TAG, "GetAllQValue [value] = " + q.Q_all.get(i).value);
```

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	SetAllQValue, GetQValue, SetQValue

SetAllQValue

Purpose

Sets the all Q value.

Syntax

public int SetAllQValue(AllQValue allq)

Parameters

AllQValue ArrayList<QValue> *allqvalue*

Dynamic	False=fixed; true=dynamic.
value	The Q value. Ranges from 0~15.
Min	Minimum Q value. Ranges from 0~15.
Max	Maximum Q value. Ranges from 0~15.

Note: Q value is dependent on work mode; each work mode has its own Q value parameter.

Example

```
AllQValue q = new AllQValue();
int re = mRfidManager.GetAllQValue(q);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();Log.e(TAG, "GetLastError = " +
m);
return; }

QValue MultiTag = new QValue();
MultiTag.Dynamic =
q.Q_all.get(WorkMode.MultiTagMode.ordinal()).Dynamic;
MultiTag.value = 9;
MultiTag.Max = q.Q_all.get(WorkMode.MultiTagMode.ordinal()).Max;
MultiTag.Min = q.Q_all.get(WorkMode.MultiTagMode.ordinal()).Min;
q.Q_all.set(WorkMode.MultiTagMode.ordinal(),MultiTag);
QValue user1 = new QValue();
user1.Dynamic = q.Q_all.get(WorkMode.UserDefine1.ordinal()).Dynamic;
user1.value = 10;
user1.Max = q.Q_all.get(WorkMode.UserDefine1.ordinal()).Max;
user1.Min = q.Q_all.get(WorkMode.UserDefine1.ordinal()).Min;
q.Q_all.set(WorkMode.UserDefine1.ordinal(),user1);

re = mRfidManager.SetAllQValue(q);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();Log.e(TAG, "GetLastError = " +
m);}
```

Return Value

If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also

GetAllQValue, GetQValue, SetQValue

GetAllRFLink

Purpose	Gets the all RF Link.
Syntax	**public int GetAllRFLink(AllRFLink allrf)**
Parameters	A default value comes with an asterisk "*". AllRFLink ArrayList <RFLink> *allrflink*

RFLink.DSB_ASK_FM0_40KHz	DSB_ASK /FM0/ 40 KHz
RFLink.PR_ASK_Miller4_250KHz	PR_ASK /Miller4/ 250KHz
RFLink.PR_ASK_Miller4_300KHz	PR_ASK /Miller4/ 300KHz
RFLink.DSB_ASK_FM0_400KHz	* DSB_ASK /FM0/ 400KHz
RFLink.PR_ASK_Miller2_250KHz	PR_ASK /Miller2/ 250KHz
RFLink.PR_ASK_FM0_250KHz	PR_ASK /FM0/ 250KHz

Remarks

RF Link is dependent on work modes; each work mode has its own RF Link parameter.

RFLink.DSB_ASK_FM0_40KHz and RFLink.DSB_ASK_FM0_400KHz parameters are not available when the RFID device region is set to Japan.

Example

```
AllRFLink rf = new AllRFLink();
int re = mRfidManager.GetAllRFLink(rf);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();Log.e(TAG, "GetLastError = " +
m);}
for (int i = 0; i < 8; i++)
Log.i(TAG, "GetAllRFLink [RFLink] = "+ rf.RFLink_all.get(i));
```

Return Value

If successful, it returns ClResult.S_OK.ordinal().

Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also

SetAllRFLink, GetRFLink, SetRFLink

SetAllRFLink

Purpose	Sets the all RF Link.
Syntax	**public int SetAllRFLink(AllRFLink allrf)**
Parameters	A default value comes with an asterisk "*". AllRFLink ArrayList <RFLink> *allrflink*

RFLink.DSB_ASK_FM0_40KHz	DSB_ASK /FM0/ 40 KHz
RFLink.PR_ASK_Miller4_250KHz	PR_ASK /Miller4/ 250KHz
RFLink.PR_ASK_Miller4_300KHz	PR_ASK /Miller4/ 300KHz
RFLink.DSB_ASK_FM0_400KHz	* DSB_ASK /FM0/ 400KHz
RFLink.PR_ASK_Miller2_250KHz	PR_ASK /Miller2/ 250KHz
RFLink.PR_ASK_FM0_250KHz	PR_ASK /FM0/ 250KHz

Remarks

RF Link is dependent on work modes; each work mode has its own RF Link parameter.

RFLink.DSB_ASK_FM0_40KHz and RFLink.DSB_ASK_FM0_400KHz parameters are not available when the RFID device region is set to Japan.

Example

```
AllRFLink rf = new AllRFLink();
int re = mRfidManager.GetAllRFLink(rf);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();Log.e(TAG, "GetLastError = " +
m);
return; }
rf.RFLink_all.set(WorkMode.UserDefine1.ordinal(),
RFLink.PR_ASK_Miller4_300KHz.ordinal());
rf.RFLink_all.set(WorkMode.UserDefine2.ordinal(),
RFLink.DSB_ASK_FM0_40KHz.ordinal());
re = mRfidManager.SetAllRFLink(rf);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();Log.e(TAG, "GetLastError = " +
m);}
```

Return Value

If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also

GetAllRFLink, GetRFLink, SetRFLink

GetAllGen2

Purpose	Gets the all Gen2.
Syntax	**public int GetAllGen2(AllGen2Settings allsettings)**
Parameters	A default value comes with an asterisk "*". AllGen2Settings ArrayList <Gen2Settings> *allsettings*

SessionSettings	Session data SessionSettings.S0 * SessionSettings.S1 SessionSettings.S2 SessionSettings.S3
InventoryStatusSettings	Inventory status *InventoryStatusSettings.STATE_A InventoryStatusSettings.STATE_B InventoryStatusSettingsAB_FLIP
SLFlagSettings	SL Flag *SLFlagSettings.All SLFlagSettings.Deasserted SLFlagSettings.Asserted

Note: RFID Gen2 is dependent to work mode, each work mode has its own RFID Gen2 parameter.

Example

```
AllGen2Settings allsettings = new AllGen2Settings();
int re = mRfidManager.GetAllGen2(allsettings);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();Log.e(TAG, "GetLastError = " +
m);}
for (int i = 0; i < 8; i++) {
Log.i(TAG,"GetAllGen2 [Session]  ("+ i + ") = " +
allsettings.Gen2_all.get(i).Session);
Log.i(TAG,"GetAllGen2 [SL_Flag] ("+ i + ") = " +
allsettings.Gen2_all.get(i).SL_Flag);
Log.i(TAG,"GetAllGen2 [InventoryStatus_Action] ("+ i + ") = " +
allsettings.Gen2_all.get(i).InventoryStatus_Action); }
```

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	SetAllGen2, GetGen2, SetGen2

SetAllGen2	
Purpose	Sets the all Gen2.
Syntax	**public int SetAllGen2(AllGen2Settings allsettings)**
Parameters	A default value comes with an asterisk "*". AllGen2Settings ArrayList <Gen2Settings> *allsettings*

SessionSettings	Session data SessionSettings.S0 *SessionSettings.S1 SessionSettings.S2 SessionSettings.S3
InventoryStatusSettings	Inventory status *InventoryStatusSettings.STATE_A InventoryStatusSettings.STATE_B InventoryStatusSettingsAB_FLIP
SLFlagSettings	SL Flag *SLFlagSettings.All SLFlagSettings.Deasserted SLFlagSettings.Asserted

Note: RFID Gen2 is dependent on work mode; each work mode has its own RFID Gen2 parameter.

Example

```
AllGen2Settings allsettings = new AllGen2Settings();
int re = mRfidManager.GetAllGen2(allsettings);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();Log.e(TAG, "GetLastError = " +
m);
return; }
Gen2Settings settings_multi = new Gen2Settings();
settings_multi.Session = SessionSettings.S1;
settings_multi.SL_Flag =  SLFlagSettings.Asserted;
settings_multi.InventoryStatus_Action =
InventoryStatusSettings.STATE_B;
allsettings.Gen2_all.set(WorkMode.MultiTagMode.ordinal(),
settings_multi);
Gen2Settings settings_ComprehensiveMode = new Gen2Settings();
settings_ComprehensiveMode.Session = SessionSettings.S2;
settings_ComprehensiveMode.SL_Flag =  SLFlagSettings.Asserted;
settings_ComprehensiveMode.InventoryStatus_Action =
InventoryStatusSettings.AB_FLIP;
allsettings.Gen2_all.set(WorkMode.ComprehensiveMode.ordinal(),
settings_ComprehensiveMode);
Gen2Settings settings_df1 = new Gen2Settings();
settings_df1.Session = SessionSettings.S3;
settings_df1.SL_Flag =  SLFlagSettings.Deasserted;
settings_df1.InventoryStatus_Action =
InventoryStatusSettings.STATE_B;
allsettings.Gen2_all.set(WorkMode.UserDefine1.ordinal(),
settings_df1);

re = mRfidManager.SetAllGen2(allsettings);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();Log.e(TAG, "GetLastError = " +
m);}
```

Return Value
If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also
GetAllGen2, GetGen2, SetGen2

DeviceTriggerStatus

Purpose	Get the UHF RFID device's scan trigger key status
Syntax	**public int DeviceTriggerStatus()**

Example	`int Status = mRfidManager.DeviceTriggerStatus();`
	`if(Status == -1) {`
	`String m = mRfidManager.GetLastError();Log.e(TAG, "GetLastError = " + m);}`

Return Value	If successful, it returns the scan trigger key status: 0: RFID scanning disabled 1: RFID scanning enabled. Otherwise, it returns -1.
See Also	EnableDeviceTrigger

EnableDeviceTrigger

Purpose	Enables or disables the UHF RFID device's scan trigger key.
Syntax	**public int EnableDeviceTrigger(boolean value)**

Example	`int re = mRfidManager.EnableDeviceTrigger(false);`
	`if (re != ClResult.S_OK.ordinal()) {`
	`String m = mRfidManager.GetLastError();Log.e(TAG, "GetLastError = " + m);}`

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	DeviceTriggerStatus

SoftScanTrigger

Purpose	Emulates the behaviour of physical trigger key. The following steps have to be done beforehand: 1. Register for the Cipherlab-specific string – GeneralString.Intent_RFIDSERVICE_TAG_DATA – by calling the android.content.ContextWrapper.registerReceiver function. 2. Receive the registered string by calling the Android BroadcastReceiver() function. 3. Fetch the data from the received Intent.
Syntax	**public int SoftScanTrigger(boolean Status)**

Example	`int re = mRfidManager.SoftScanTrigger(true); //Scan on`
	`//int re = mRfidManager.SoftScanTrigger(false); //Scan off`
	`if (re != ClResult.S_OK.ordinal()) {`
	`String m = mRfidManager.GetLastError();Log.e(TAG, "GetLastError = " + m);}`

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	GeneralString.Intent_RFIDSERVICE_TAG_DATA

GetRecognizedEPCEncoding	
Purpose	Gets the EPC encoding schemes that are recognized by the UHF RFID Reader.
Syntax	**public int GetRecognizedEPCEncoding(EPCEncodingScheme encoding)**
Parameters	EPCEncodingScheme *encoding*

GDTI96	96-bit "Global Document Type Identifier" encoding scheme
GSRN96	96-bit "Global Service Relation Number" encoding scheme
GSRNP	"Global Service Relation Number– Provider" encoding scheme
USDoD96	96-bit "US Department of Defense Identifier" encoding scheme
SGTIN96	96-bit "Serialized Global Trade Item Number" encoding scheme
SSCC96	96-bit "Serial Shipping Container Code" encoding scheme
SGLN96	96-bit "Global Location Number With or Without Extension" encoding scheme
GRAI96	96-bit "Global Returnable Asset Identifier" encoding scheme
GIAI96	96-bit "Global Individual Asset Identifier" encoding scheme
GID96	96-bit "General Identifier" encoding scheme
SGTIN198	198-bit "Serialized Global Trade Item Number" encoding scheme
GRAI170	170-bit "Global Returnable Asset Identifier" encoding scheme
GIAI202	202-bit "Global Individual Asset Identifier" encoding scheme
SGLN195	195-bit "Global Location Number With or Without Extension" encoding scheme
GDTI113	113-bit "Global Document Type Identifier" encoding scheme
ADI	"Aerospace and Defense Identifier" encoding scheme
CPI96	96-bit "Component and Part Identifier" encoding scheme
CPI	"Component and Part Identifier" encoding scheme
GDTI174	174-bit "Global Document Type Identifier" encoding scheme
SGCN96	96-bit "Serialised Global Coupon Number" encoding scheme

Note: RFID Gen2 is dependent to work mode, each work mode has its own RFID Gen2 parameter.

Example	(see code below)

```
EPCEncodingScheme encode = new EPCEncodingScheme();
int re = mRfidManager.GetRecognizedEPCEncoding(encode);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
else{
Log.i(TAG, "GDTI96 = " + encode.GDTI96);
Log.i(TAG, "GSRN96 = " + encode.GSRN96);
Log.i(TAG, "GSRNP = " + encode.GSRNP);
Log.i(TAG, "USDoD96 = " + encode.USDoD96);
Log.i(TAG, "SGTIN96 = " + encode.SGTIN96);
Log.i(TAG, "SSCC96 = " + encode.SSCC96);
Log.i(TAG, "SGLN96 = " + encode.SGLN96);
Log.i(TAG, "GRAI96 = " + encode.GRAI96);
Log.i(TAG, "GIAI96 = " + encode.GIAI96);
Log.i(TAG, "GID96 = " + encode.GID96);
Log.i(TAG, "SGTIN198 = " + encode.SGTIN198);
Log.i(TAG, "GRAI170 = " + encode.GRAI170);
Log.i(TAG, "GIAI202 = " + encode.GIAI202);
Log.i(TAG, "SGLN195 = " + encode.SGLN195);
Log.i(TAG, "GDTI113 = " + encode.GDTI113);
Log.i(TAG, "ADI = " + encode.ADI);
Log.i(TAG, "CPI96 = " + encode.CPI96);
Log.i(TAG, "CPI = " + encode.CPI);
Log.i(TAG, "GDTI174 = " + encode.GDTI174);
Log.i(TAG, "SGCN96 = " + encode.SGCN96); }
```

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	SetRecognizedEPCEncoding

SetRecognizedEPCEncoding	
Purpose	Sets the EPC encoding schemes that are recognized by the UHF RFID Reader.
Syntax	**public int SetRecognizedEPCEncoding(EPCEncodingScheme encoding)**
Parameters	EPCEncodingScheme *encoding*

GDTI96	96-bit "Global Document Type Identifier" encoding scheme
GSRN96	96-bit "Global Service Relation Number" encoding scheme
GSRNP	"Global Service Relation Number– Provider" encoding scheme
USDoD96	96-bit "US Department of Defense Identifier" encoding scheme
SGTIN96	96-bit "Serialized Global Trade Item Number" encoding scheme
SSCC96	96-bit "Serial Shipping Container Code" encoding scheme
SGLN96	96-bit "Global Location Number With or Without Extension" encoding scheme
GRAI96	96-bit "Global Returnable Asset Identifier" encoding scheme
GIAI96	96-bit "Global Individual Asset Identifier" encoding scheme
GID96	96-bit "General Identifier" encoding scheme
SGTIN198	198-bit "Serialized Global Trade Item Number" encoding scheme
GRAI170	170-bit "Global Returnable Asset Identifier" encoding scheme
GIAI202	202-bit "Global Individual Asset Identifier" encoding scheme
SGLN195	195-bit "Global Location Number With or Without Extension" encoding scheme
GDTI113	113-bit "Global Document Type Identifier" encoding scheme
ADI	"Aerospace and Defense Identifier" encoding scheme
CPI96	96-bit "Component and Part Identifier" encoding scheme
CPI	"Component and Part Identifier" encoding scheme
GDTI174	174-bit "Global Document Type Identifier" encoding scheme
SGCN96	96-bit "Serialised Global Coupon Number" encoding scheme

Note: RFID Gen2 is dependent to work mode, each work mode has its own RFID Gen2 parameter.

Example

```
EPCEncodingScheme encode = new EPCEncodingScheme();
int re = mRfidManager.GetRecognizedEPCEncoding(encode);
encode.SGLN96 = false;
re = mRfidManager.SetRecognizedEPCEncoding(encode);
if (re != ClResult.S_OK.ordinal()){
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m); }
```

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	GetRecognizedEPCEncoding

GetDataOutputSettings

Purpose	Gets the current output data format.
Syntax	**int GetDataOutputSettings(RfidOutputConfiguration settings)**
Parameters	RfidOutputConfiguration *Settings* [in][out] A value that specifies whether to auto-affix a character after EPC.

RfidOutputConfiguration. szEPCPrefixCode	Prefixes the EPC data (= Enter-character + EPC data)
RfidOutputConfiguration. szEPCSuffixCode	Suffixes the EPC data (= EPC data + Enter-character)
RfidOutputConfiguration. KeyEventOutput	Enables keyboard emulation (works only when the ScanMode is set to Single).
RfidOutputConfiguration. InterCharDelay	Gets or sets the inter-character delay of the key event (works only when the ScanMode is set to Single).

Example

```
RfidOutputConfiguration Settings = new RfidOutputConfiguration();
int re = mRfidManager.GetDataOutputSettings(Settings);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m);}
else {
Log.i(TAG, "szEPCPrefixCode = " + Settings.szEPCPrefixCode);
Log.i(TAG, "szEPCSuffixCode = " + Settings.szEPCSuffixCode);
Log.i(TAG, "KeyboardOutput = " + Settings.KeyEventOutput);
Log.i(TAG, "InterCharDelay = " + Settings.InterCharDelay);}
```

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	SetDataOutputSettings

SetDataOutputSettings

Purpose	Sets the current output data format.
Syntax	**int SetDataOutputSettings(RfidOutputConfiguration settings)**

Parameters

RfidOutputConfiguration *Settings*

[in][out] A value that specifies whether to auto-affix a character after EPC.

RfidOutputConfiguration. szEPCPrefixCode	Prefixes the EPC data (= Enter-character + EPC data)
RfidOutputConfiguration. szEPCSuffixCode	Suffixes the EPC data (= EPC data + Enter-character)
RfidOutputConfiguration. KeyEventOutput	Enables keyboard emulation (works only when the ScanMode is set to Single).
RfidOutputConfiguration. InterCharDelay	Gets or sets the inter-character delay of the key event (works only when the ScanMode is set to Single).

Example

```
RfidOutputConfiguration Settings = new RfidOutputConfiguration();
Settings.szEPCPrefixCode ="AAA";
Settings.szEPCSuffixCode ="BBB";
Settings.KeyEventOutput = true;
Settings.InterCharDelay = 100; // 100ms
int re = mRfidManager.SetDataOutputSettings(Settings);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m);}
```

Return Value

If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also

GetDataOutputSettings

RFIDDirectUntraceableTag

Purpose	Gen2V2 Untraceable operations. Untraceable lets the user decide which memory bank to show and what length of the memory bank to show.
Syntax	**DeviceResponse RFIDDirectUntraceableTag(byte[] password, RFIDMemoryBank bank, int start, byte[] filterdata, UntraceableU u, int untraceable_epc, UntraceableTID tid, UntraceableUser user, UntraceableRange range, int retry)**

Parameters

byte[] *password*

Tag access password.

RFIDMemoryBank *bank*

RFIDMemoryBank.EPC	EPC bank
RFIDMemoryBank.TID	TID bank
RFIDMemoryBank.User	User bank

int *Start*

Start byte position in Filter data.

byte[] *filterdata*

Content of filter data.

UntraceableU *u*

A value for the U bit in XPC_W1.

UntraceableU.DeassertU	Deassert U in XPC_W1
UntraceableU.AssertU	Assert U in XPC_W1

int *untraceable_epc*

A value for the epc length (Words).

UntraceableTID *tid*

TID specifies the TID memory that a Tag untraceably hides.

UntraceableTID.HideNone	The tag exposes TID memory.
UntraceableTID.HideSome	The tag hides some TID memory.
UntraceableTID.HideAll	The tag untraceably hides all of TID memory.

UntraceableUser *user*

User specifies whether a Tag untraceably hides User memory.

UntraceableUser.View	The Tag exposes User memory.
UntraceableUser.Hide	The tag untraceably hides User memory.

Parameters

UntraceableRange *range*

Range specifies a Tag’s operating range.

UntraceableRange.Normal	The tag persistently enables normal operating range.
UntraceableRange.ToggleTemporarily	The tag temporarily toggles its operating range (if normal then to reduced; if reduced then to normal) but reverts to its prior persistent operating range when the Tag loses power.
UntraceableRange.Reduced	The tag persistently enables reduced operating range.

int *retry*

Number of retries if initial read fails.

DeviceResponse *Response*

DeviceResponse.OperationSuccess	Operation succeeded.
DeviceResponse.OperationFail	Operation failed.
DeviceResponse.OperationFinish	Operation finish.
DeviceResponse.DeviceTimeOut	UHF RFID device timed out.
DeviceResponse.DeviceBusy	UHF RFID device is busy.

Example

```
byte[] password = new byte[] { (byte)0x11, (byte)0x11, (byte)0x11,
(byte)0x11};
byte[] EPCByteArray1 = new byte[] {  (byte)0xe2, (byte)0xc0, (byte)
0x68, (byte) 0x92, (byte)0x00, (byte)0x00, (byte)0x00, (byte)0x3a,
(byte)0x1e, (byte)0x33, (byte)0xe1, (byte)0x2a };
DeviceResponse re = mRfidManager.RFIDDirectUntraceableTag(password,
RFIDMemoryBank.EPC, 4 , EPCByteArray1, UntraceableU.DeassertU, 5 ,
UntraceableTID.HideNone ,
UntraceableUser.View ,UntraceableRange.Normal , 5);
Log.i(TAG, "RFIDDirectUntraceableTag(re) = " + re);
```

Return Value

It returns DeviceResponse.

Intent Data

RESPONSE Data	Type, Response

See Also

GeneralString.Intent_RFIDSERVICE_TAG_DATA

RFIDDirectAuthenticateTag	
Purpose	Gen2V2 Authenticate operations. Automatically triggers a RFID tag read on the UHF RFID device with a given filterdata. Trigger-press is not required.
Syntax	**DeviceResponse RFIDDirectAuthenticateTag(byte[] password, RFIDMemoryBank bank, int start, byte[] filterdata, AuthenticateSenRep senrep, AuthenticateIncRepLen increplen, byte[] message, int retry)**
Parameters	byte[] *password*

Tag access password.

RFIDMemoryBank *bank*

RFIDMemoryBank.EPC	EPC bank
RFIDMemoryBank.TID	TID bank
RFIDMemoryBank.User	User bank

int *Start*

Start byte position in Filter data.

byte[] *filterdata*

Content of filter data.

AuthenticateSenRep *senrep*

SenRep specifies whether a Tag backscatters its response or stores the response in its ResponseBuffer.

AuthenticateSenRep.Store	Store
AuthenticateSenRep.Send	Send

AuthenticateIncRepLen *increplen*

IncRepLen specifies whether a Tag omits or includes length in its reply.

AuthenticateIncRepLen.Omit_Length_From_Reply	The tag omits length from its reply
AuthenticateIncRepLen.Included_Length_From_Reply	The tag includes length in its reply.

byte[] *message*

Content of message data. (10 bytes)

int *retry*

Number of retries if initial read fails.

DeviceResponse *Response*

DeviceResponse.OperationSuccess	Operation succeeded.
DeviceResponse.OperationFail	Operation failed.
DeviceResponse.OperationFinish	Operation finish.
DeviceResponse.DeviceTimeOut	UHF RFID device timed out.
DeviceResponse.DeviceBusy	UHF RFID device is busy.

Example	`byte[] password = new byte[] { (byte)0x00, (byte)0x00, (byte)0x00, (byte)0x00};` `byte[] MessageByteArray = new byte[] { (byte)0x8e, (byte)0x02, (byte) 0x49, (byte) 0x9d, (byte)0x2d, (byte)0x26, (byte)0x03, (byte)0xf9, (byte)0x8f, (byte)0x5c};` `byte[] EPCByteArray1 = new byte[] {  (byte)0xe2, (byte)0xc0, (byte) 0x68, (byte) 0x92, (byte)0x00, (byte)0x00, (byte)0x00, (byte)0x3a, (byte)0x1e, (byte)0x33, (byte)0xe1, (byte)0x2a };` `DeviceResponse re = mRfidManager.RFIDDirectAuthenticateTag(password, RFIDMemoryBank.EPC, 4 , EPCByteArray1, AuthenticateSenRep.Send, AuthenticateIncRepLen.Included_Length_From_Reply , MessageByteArray  , 5);` `Log.i(TAG, "RFIDDirectAuthenticateTag(re) = " + re);`
Return Value	It returns DeviceResponse.
Intent Data	**RESPONSE Data** | Type, Response, ReadData, ReadData length
See Also	GeneralString.Intent_RFIDSERVICE_TAG_DATA

Authenticate process

Step1. Set Key0 for Authentication
Write Key0 to Memory bank3 (User), Start address 192 (0xC0), Length 128bits (16 bytes).
For example, set Key 0 to 0x11111111111111111111111111111111

```
public void For_Set_Authenticate_Key0_Test(){
byte[] password = new byte[] { (byte)0x00, (byte)0x00, (byte)0x00,
(byte)0x00};
byte[] WriteDataArray = new byte[] { (byte)0x11, (byte)0x11,
(byte)0x11, (byte)0x11, (byte)0x11, (byte)0x11, (byte)0x11,
(byte)0x11, (byte)0x11, (byte)0x11, (byte)0x11, (byte)0x11,
(byte)0x11, (byte)0x11, (byte)0x11, (byte)0x11 };
byte[] EPCByteArray = new byte[] {  (byte)0xe2, (byte)0xc0, (byte) 0x68,
(byte) 0x92, (byte)0x00, (byte)0x00, (byte)0x00, (byte)0x3a,
(byte)0x1e, (byte)0x33, (byte)0xe1, (byte)0x2a };
DeviceResponse re = mRfidManager.RFIDDirectWriteTagByEPC(password,
EPCByteArray, RFIDMemoryBank.User, 384, 3, WriteDataArray);
Log.i(TAG, "RFIDDirectWriteTagByEPC Set Key0 (re) = " + re);}
```

Step2. Activate Key0 for Authentication
Write 0xE200 to Memory bank3 (User), Start address 200 (0xC8), Length 16bits (2 bytes). This step can activate Key0 authentication. After activating Key0 with success, contents of Key0 mentioned in Step 1 cannot be read.

```
public void For_Activate_Authenticate_Key0_Test(){
byte[] password = new byte[] { (byte)0x00, (byte)0x00, (byte)0x00,
(byte)0x00};
byte[] WriteDataArray = new byte[] { (byte)0xe2, (byte)0x00};
byte[] EPCByteArray = new byte[] {  (byte)0xe2, (byte)0xc0, (byte) 0x68,
(byte) 0x92, (byte)0x00, (byte)0x00, (byte)0x00, (byte)0x3a,
(byte)0x1e, (byte)0x33, (byte)0xe1, (byte)0x2a };
DeviceResponse re = mRfidManager.RFIDDirectWriteTagByEPC(password,
EPCByteArray, RFIDMemoryBank.User, 400, 3, WriteDataArray);
Log.i(TAG, "RFIDDirectWriteTagByEPC Activate Key0 (re) = " + re);}
```

Authenticate process

Step3. Generate 80-bit (10bytes) Random value for Authentication message.

For example: randomly sends "0x8E 02 49 9D 2D 26 03 F9 8F 5C" to tags.

```
byte[] MessageByteArray = new byte[] { (byte)0x8e, (byte)0x02, (byte)
0x49, (byte) 0x9d, (byte)0x2d, (byte)0x26, (byte)0x03, (byte)0xf9,
(byte)0x8f, (byte)0x5c};
```

Step4. Tag will respond the message with encryption. (tag response to Reader)

```
public void ForRFIDDirectAuthenticateTagTest(){
byte[] password = new byte[] { (byte)0x00, (byte)0x00, (byte)0x00,
(byte)0x00};
byte[] MessageByteArray = new byte[] { (byte)0x8e, (byte)0x02, (byte)
0x49, (byte) 0x9d, (byte)0x2d, (byte)0x26, (byte)0x03, (byte)0xf9,
(byte)0x8f, (byte)0x5c};
byte[] EPCByteArray1 = new byte[] {  (byte)0xe2, (byte)0xc0, (byte)
0x68, (byte) 0x92, (byte)0x00, (byte)0x00, (byte)0x00, (byte)0x3a,
(byte)0x1e, (byte)0x33, (byte)0xe1, (byte)0x2a };
DeviceResponse re = mRfidManager.RFIDDirectAuthenticateTag(password,
RFIDMemoryBank.EPC, 4 , EPCByteArray1, AuthenticateSenRep.Send,
AuthenticateIncRepLen.Included_Length_From_Reply ,
MessageByteArray  , 5);
Log.i(TAG, "RFIDDirectAuthenticateTag(re) = " + re);}
```

Authentication Response: If type=8, authentication response data in ReadData.

```
If
(intent.getAction().equals(GeneralString.Intent_RFIDSERVICE_TAG_DAT
A)){
/* * type : 0=Normal scan (Press Trigger Key to receive the data) ;
1=Inventory EPC ; 2=Inventory ECP TID ; 3=Reader tag ; 5=Write tag ;
6=Lock tag ; 7=Kill tag ; 8=Authenticate tag ; 9=Untraceable tag
 * response : 0=RESPONSE_OPERATION_SUCCESS ;
1=RESPONSE_OPERATION_FINISH ; 2=RESPONSE_OPERATION_TIMEOUT_FAIL ;
6=RESPONSE_PASSWORD_FAIL ;
7=RESPONSE_OPERATION_FAIL ;251=DEVICE_BUSY * */
int type = intent.getIntExtra(GeneralString.EXTRA_DATA_TYPE, -1);
int response = intent.getIntExtra(GeneralString.EXTRA_RESPONSE, -1);
String ReadData =
intent.getStringExtra(GeneralString.EXTRA_ReadData);
int ReadData_length =
intent.getIntExtra(GeneralString.EXTRA_ReadData_LENGTH, 0);}
```

For example: the reader will receive the tag response message as below: "140001ca01014ac833a180697f7ce43a3089e42a8ab0"

[Intent_RFIDSERVICE_TAG_DATA] ReadData=140001ca01014ac833a180697f7ce43a3089e42a8ab

[Intent_RFIDSERVICE_TAG_DATA] ReadData_length=22

Decrypt the last 16 bytes (0x4ac833a180697f7ce43a3089e42a8ab0) of ReadData.

Step5. Use AES tool to decrypt the received message.

For example, decrypting "0x4ac833a180697f7ce43a3089e42a8ab0" will get the "0x96 C5 A3 2D 1D 6E 8E 02 49 9D 2D 26 03 F9 8F 5C" message in which the last 80-bit (10 bytes) value ("8E 02 49 9D 2D 26 03 F9 8F 5C") is the same as the one mentioned in Step 3 (the reader sent to tags).

GetJapanChannel

Purpose: Gets the Japan channel on the UHF RFID Reader.

Syntax: **public int GetJapanChannel(JapanChannel channel)**

Parameters: JapanChannel *channel*

JP_916_8Mhz	916.8 MHz
JP_918_0Mhz	918.0 MHz
JP_919_2Mhz	919.2 MHz
JP_920_4Mhz	920.4 MHz
JP_920_6Mhz	920.6 MHz
JP_920_8Mhz	920.8 MHz
JP_916_8Mhz	916.8 MHz

Example:

```
JapanChannel channel = new JapanChannel();
int re = mRfidManager.GetJapanChannel(channel);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m);}
else{
Log.i(TAG, "GetJapanChannel (JP_916_8Mhz) = " + channel.JP_916_8Mhz);
Log.i(TAG, "GetJapanChannel (JP_918_0Mhz) = " + channel.JP_918_0Mhz);
Log.i(TAG, "GetJapanChannel (JP_919_2Mhz) = " + channel.JP_919_2Mhz);
Log.i(TAG, "GetJapanChannel (JP_920_4Mhz) = " + channel.JP_920_4Mhz);
Log.i(TAG, "GetJapanChannel (JP_920_6Mhz) = " + channel.JP_920_6Mhz);
Log.i(TAG, "GetJapanChannel (JP_920_8Mhz) = " + channel.JP_920_8Mhz);}
```

Return Value: If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also: SetJapanChannel

SetJapanChannel

Purpose	Sets the Japan channel on the UHF RFID Reader.
Syntax	**public int SetJapanChannel(JapanChannel channel)**
Parameters	JapanChannel *channel*

JP_916_8Mhz	916.8 MHz
JP_918_0Mhz	918.0 MHz
JP_919_2Mhz	919.2 MHz
JP_920_4Mhz	920.4 MHz
JP_920_6Mhz	920.6 MHz
JP_920_8Mhz	920.8 MHz
JP_916_8Mhz	916.8 MHz

Note: This function works only when the RFID device region is set to Japan.

Example

```
int re = 0;
JapanChannel M_channel = new JapanChannel();
M_channel.JP_916_8Mhz = true;
M_channel.JP_920_8Mhz = true;
re = mRfidManager.SetJapanChannel(M_channel);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m);}
```

Return Value: If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also: GetJapanChannel

GetContinuousInventoryTime

Purpose	Gets the Continuous inventory time & delay time on the UHF RFID Reader.
Syntax	**public int GetContinuousInventoryTime(ContinuousInventoryTime time)**
Parameters	**ContinuousInventoryTime** *time*

InventoryTime	The continuous inventory time setting value ranges from 0 to 1000.
DelayTime	The continuous inventory delay time setting value ranges from 0 to 1000.

Example

```
ContinuousInventoryTime time = new ContinuousInventoryTime();
int re = mRfidManager.GetContinuousInventoryTime(time);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m);}
else{
Log.i(TAG, "GetContinuousInventoryTime (InventoryTime) = " +
time.InventoryTime);
Log.i(TAG, "GetContinuousInventoryTime (DelayTime) = " +
time.DelayTime);}
```

Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	SetContinuousInventoryTime, GetPowerMode, SetPowerMode

SetContinuousInventoryTime

Purpose: Sets the Continuous inventory time & delay time on the UHF RFID Reader.

Syntax: **public SetContinuousInventoryTime(in ContinuousInventoryTime time)**

Parameters: **ContinuousInventoryTime** *time*

InventoryTime	The continuous inventory time setting value ranges from 0 to 1000.
DelayTime	The continuous inventory delay time setting value ranges from 0 to 1000.

Example

```
int re = 0;
ContinuousInventoryTime time = new ContinuousInventoryTime();
time.InventoryTime = 170;
time.DelayTime = 30;
re = mRfidManager.SetContinuousInventoryTime(time);
if (re != ClResult.S_OK.ordinal()) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m);}
```

Return Value: If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also: GetContinuousInventoryTime, GetPowerMode, SetPowerMode

GetPowerMode

Purpose: Gets the power mode on the UHF RFID Reader.

Syntax: **public PowerMode GetPowerMode()**

Parameters: A default value comes with an asterisk "*".
PowerMode *mode*

PowerMode.PowerSave	* Power Save mode; Continuous inventory time=110, Delay time=90.
PowerMode.Normal	Normal mode; Continuous inventory time=170, Delay time=30.
PowerMode.Boost	Boost mode; Continuous inventory time=0, Delay time=0.
PowerMode.Other	Other

Example

```
PowerMode mode = mRfidManager.GetPowerMode();
if(mode == PowerMode.Err){
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m);}
```

Return Value: If successful, it returns the PowerMode.
Otherwise, it returns `PowerMode.Err`.

See Also: SetPowerMode, GetContinuousInventoryTime, SetContinuousInventoryTime

SetPowerMode

Purpose	Sets the power mode on the UHF RFID Reader.
Syntax	**public int SetPowerMode(PowerMode mode)**
Parameters	A default value comes with an asterisk "*". **PowerMode** *mode*

PowerMode.PowerSave	* Power Save mode; Continuous inventory time=110, Delay time=90.
PowerMode.Normal	Normal mode; Continuous inventory time=170, Delay time=30.
PowerMode.Boost	Boost mode; Continuous inventory time=0, Delay time=0.

Example

```
int re = mRfidManager.SetPowerMode(PowerMode.Normal);
if(re!=ClResult.S_OK.ordinal()){
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m);}
```

Return Value: If successful, it returns ClResult.S_OK.ordinal().
Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also: GetPowerMode, GetContinuousInventoryTime, SetContinuousInventoryTime

GetTriggerSwitchMode

Purpose	Gets the new trigger change switch mode by the UHF RFID Reader. Default by false. Trigger 5 times in 1sec to switch mode.
Syntax	**public TriggerSwitchMode GetTriggerSwitchMode()**
Parameters	A default value comes with an asterisk "*". **TriggerSwitchMode** *mode*

TriggerSwitchStatus	*False=Disable. True=Enable. Trigger 5 times in 1sec to switch mode.
CurrentSwitchMode	UHF RFID device reader switch position. *SwitchMode.UHFRFIDReader SwitchMode.BarcodeReader SwitchMode.UHFRFIDBarcodeReader

*If TriggerSwitchMode = false, SwitchMode (SetSwitchMode) cannot be set.

*If TriggerSwitchMode = true, RFIDSwitchStatus (SetRFIDSwitchStatus) cannot be set.

Example

```
TriggerSwitchMode TSM = new TriggerSwitchMode();
TSM = mRfidManager.GetTriggerSwitchMode();
if(TSM==null) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m);}
else{
Log.i(TAG, "GetTriggerSwitchMode (TriggerSwitchStatus)= " +
TSM.TriggerSwitchStatus );
Log.i(TAG, "GetTriggerSwitchMode (CurrentSwitchMode)= " +
TSM.CurrentSwitchMode );}
```

Return Value	If successful, it returns the TriggerSwitchMode. Otherwise, it returns null.
See Also	SetTriggerSwitchMode, GetSwitchMode, SetSwitchMode

SetTriggerSwitchMode

Purpose: Sets the new trigger change switch mode by the UHF RFID Reader. Default by false. Trigger 5 times in 1sec to switch mode.

Syntax: **int SetTriggerSwitchMode(boolean Status)**

*If TriggerSwitchMode = false, SwitchMode (SetSwitchMode) cannot be set.

*If TriggerSwitchMode = true, RFIDSwitchStatus (SetRFIDSwitchStatus) cannot be set.

Example:

```
int re = mRfidManager.SetTriggerSwitchMode(true);
if(re!=ClResult.S_OK.ordinal()){
String err = mRfidManager.GetLastError();
Log.e(TAG, "SetChangeSwitchMode (err) = " + err);}
```

Return Value: If successful, it returns ClResult.S_OK.ordinal().

Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also: GetTriggerSwitchMode, GetSwitchMode, SetSwitchMode

GetSwitchMode

Purpose: Gets the UHF RFID device reader switch position – RFID only or pistol only or RFID and pistol mode. (The trigger change switch mode is enable.)

Syntax: **public SwitchMode GetSwitchMode()**

Parameters: A default value comes with an asterisk "*".

SwitchMode *mode*

SwitchMode.UHFRFIDReader	* RFID only
SwitchMode.BarcodeReader	Pistol only
SwitchMode.UHFRFIDBarcodeReader	RFID and pistol mode

*If TriggerSwitchMode = false, SwitchMode (SetSwitchMode) cannot be set.

*If TriggerSwitchMode = true, RFIDSwitchStatus (SetRFIDSwitchStatus) cannot be set.

Example:

```
SwitchMode mode = mRfidManager.GetSwitchMode();
if(mode==SwitchMode.Err) {
String m = mRfidManager.GetLastError();
 Log.e(TAG, "GetLastError = " + m);}
```

Return Value: If successful, it returns the SwitchMode.
Otherwise, it returns `SwitchMode.Err`.

See Also: GetTriggerSwitchMode, SetTriggerSwitchMode, SetSwitchMode

SetSwitchMode

Purpose	Sets the UHF RFID device reader switch position – RFID only or pistol only or RFID and pistol mode. (The trigger change switch mode is enable.)
Syntax	**public int SetSwitchMode(SwitchMode mode)**
Parameters	A default value comes with an asterisk "*".

SwitchMode *mode*

SwitchMode.UHFRFIDReader	* RFID only
SwitchMode.BarcodeReader	Pistol only
SwitchMode.UHFRFIDBarcodeReader	RFID and pistol mode

*If TriggerSwitchMode = false, SwitchMode (SetSwitchMode) cannot be set.

*If TriggerSwitchMode = true, RFIDSwitchStatus (SetRFIDSwitchStatus) cannot be set.

Example

```
int re = mRfidManager.SetSwitchMode(SwitchMode.BarcodeReader);
if(re!=ClResult.S_OK.ordinal()){
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m);}
```

Return Value: If successful, it returns ClResult.S_OK.ordinal().

Otherwise, it returns ClResult.S_ERR.Ordinal().

See Also: GetTriggerSwitchMode, SetTriggerSwitchMode, GetSwitchMode

GetModuleUniqueID

Purpose: Gets the Module Unique ID.

Syntax: **public int GetModuleUniqueID()**

Example

```
int ID = mRfidManager.GetModuleUniqueID();
if(ID==-1){
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m);}
```

Return Value: If successful, it returns the Module Unique ID.
Otherwise, it returns -1.

GetFilterDuplicate

Purpose: Gets the EPC filter duplicate status.

Syntax: **public int GetFilterDuplicate()**

Example

```
int status = mRfidManager.GetFilterDuplicate();
if(status==-1) {
String m = mRfidManager.GetLastError();
Log.e(TAG, "GetLastError = " + m);}
```

Return Value: If successful, it returns the Status.

1: Enable; 0: Disable.

Otherwise, it returns -1.

See Also: SetFilterDuplicate, ClearFilterDuplicate

SetFilterDuplicate

Purpose	Sets the EPC filter duplicate status.
Syntax	**int SetFilterDuplicate(int iMode)**
Example	`int status = mRfidManager.SetFilterDuplicate(1);` `if(status!=ClResult.S_OK.getValue()){` `String m = mRfidManager.GetLastError();` `Log.e(TAG, "GetLastError = " + m);}`
Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	GetFilterDuplicate, ClearFilterDuplicate

ClearFilterDuplicate

Purpose	Clear the EPC filter duplicate data.
Syntax	**public int ClearFilterDuplicate()**
Example	`int status = mRfidManager.ClearFilterDuplicate();` `if(status!=ClResult.S_OK.getValue()){` `String m = mRfidManager.GetLastError();` `Log.e(TAG, "GetLastError = " + m);}`
Return Value	If successful, it returns ClResult.S_OK.ordinal(). Otherwise, it returns ClResult.S_ERR.Ordinal().
See Also	GetFilterDuplicate, SetFilterDuplicate

1.2. Intent

GeneralString.Intent_RFIDSERVICE_CONNECTED	
Purpose	After running InitInstance, the system makes connection between the application and the reader service. With success in making connection, this intent is sent.

GeneralString.Intent_GUN_Attached	
Purpose	When attaching the UHF RFID devices, this intent is sent.

GeneralString.Intent_GUN_Unattached	
Purpose	When removing the UHF RFID devices, this intent is sent.

GeneralString.Intent_GUN_Power		
Purpose	When the charging status of changes, this intent is sent.	

Parameters	**GeneralString.Data_GUN_ACPower**	The device AC charging mode status.
		Boolean
		True = AC charging
		False = Non-AC charging

	GeneralString.Data_GUN_Connect	The device charging mode status.
		Boolean
		True = Charging
		False = No charging

Syntax	**boolean AC = intent.getBooleanExtra(GeneralString.Data_GUN_ACPower, false);**
	boolean Connect = intent.getBooleanExtra(GeneralString.Data_GUN_Connect, false);	

GeneralString.Intent_RFIDSERVICE_TAG_DATA	
Purpose	By calling SoftScanTrigger() to scan the tag with success or press the trigger key to read/write the tag, this intent is used to inform the application. Parameters supported are as follows. The following steps have to be done beforehand: 1. Register for the Cipherlab-specific string – GeneralString.Intent_RFIDSERVICE_TAG_DATA – by calling the android.content.ContextWrapper.registerReceiver function. 2. Receive the registered string by calling the Android BroadcastReceiver() function. 3. Fetch the data from the received Intent.
Parameters	see below

GeneralString.EXTRA_DATA_TYPE	int
	0=Normal scan; (Press trigger key to receive the data)
	1=Inventory EPC;
	2=Inventory ECP TID;
	3=Reader tag;
	5=Write tag;
	6=Lock tag;
	7=Kill tag;
	8=Authenticate tag;
	9=Untraceable tag

GeneralString.EXTRA_RESPONSE	int
	0=RESPONSE_OPERATION_SUCCESS;
	1=RESPONSE_OPERATION_FINISH;
	2=RESPONSE_OPERATION_TIMEOUT_FAIL;
	6=RESPONSE_PASSWORD_FAIL;
	7=RESPONSE_OPERATION_FAIL;
	251=DEVICE_BUSY

GeneralString.EXTRA_DATA_RSSI	Double RSSI
GeneralString.EXTRA_PC	String PC
GeneralString.EXTRA_EPC	String EPC
GeneralString.EXTRA_TID	String TID
GeneralString.EXTRA_ReadData	String ReadData
GeneralString.EXTRA_EPC_LENGTH	int EPC length

Parameters

GeneralString.EXTRA_TID_LENGTH	int TID length
GeneralString.EXTRA_ReadData_LENGTH	int ReadData length

Syntax

int type = intent.getIntExtra(GeneralString.EXTRA_DATA_TYPE, -1);

int response = intent.getIntExtra(GeneralString.EXTRA_RESPONSE, -1);

double data_rssi = intent.getDoubleExtra(GeneralString.EXTRA_DATA_RSSI, 0);

String PC = intent.getStringExtra(GeneralString.EXTRA_PC);

String EPC = intent.getStringExtra(GeneralString.EXTRA_EPC);

String TID = intent.getStringExtra(GeneralString.EXTRA_TID);

String ReadData = intent.getStringExtra(GeneralString.EXTRA_ReadData);

int EPC_length = intent.getIntExtra(GeneralString.EXTRA_EPC_LENGTH, 0);

int TID_length = intent.getIntExtra(GeneralString.EXTRA_TID_LENGTH, 0);

int ReadData_length = intent.getIntExtra(GeneralString.EXTRA_ReadData_LENGTH, 0);

GeneralString.Intent_RFIDSERVICE_EVENT

Purpose

The events raised by the UHF RFID device.

Parameters

GeneralString.EXTRA_EVENT_MASK	int
	1=PowerSavingMode;
	16=LowBattery;
	64=ScannerFailure;
	8192=BatteryLose;
	16384=OverTemperature;
	32768=Battery_Re_Plug

Syntax

int event = intent.getIntExtra(GeneralString.EXTRA_EVENT_MASK, -1);

Example

```
int event  = intent.getIntExtra(GeneralString.EXTRA_EVENT_MASK, -1);
Log.d(TAG, "[Intent_RFIDSERVICE_EVENT] DeviceEvent=" + event );
if(event == DeviceEvent.LowBattery.getValue())
Log.i(GeneralString.TAG, "LowBattery " );
else if(event == DeviceEvent.PowerSavingMode.getValue() )
Log.i(GeneralString.TAG, "PowerSavingMode " );
else if(event == DeviceEvent.OverTemperature.getValue())
Log.i(GeneralString.TAG, "OverTemperature " );
else if(event == DeviceEvent.ScannerFailure.getValue())
Log.i(GeneralString.TAG, "ScannerFailure " );
```

See Also

DeviceEvent

GeneralString.Intent_FWUpdate_ErrorMessage

Purpose: The error messages raised by Firmware Update.

Parameters:

GeneralString.FWUpdate_ErrorMessage	String Error message.
GeneralString.FWUpdate_ErrorCode	int Error code

Syntax:

String mse = intent.getStringExtra(GeneralString.FWUpdate_ErrorMessage);

int errorcode = intent.getIntExtra(GeneralString.FWUpdate_ErrorCode,-1);

Example:

```
if(intent.getAction().equals(GeneralString.Intent_FWUpdate_ErrorMes
sage)) {
String mse =
intent.getStringExtra(GeneralString.FWUpdate_ErrorMessage);
int errorcode =
intent.getIntExtra(GeneralString.FWUpdate_ErrorCode,-1);
if(mse!=null) {
if(errorcode==FWUpdateErrorCode.SameVersion.getValue())
{Log.d(TAG, "SameVersion");}
Toast.makeText(MainActivity.this, mse, Toast.LENGTH_SHORT).show();}}
```

See Also: FirmwareUpdate, FWUpdateErrorCode

GeneralString.Intent_FWUpdate_Percent

Purpose: Firmware update percent.

Parameters:

GeneralString.FWUpdate_Percent	int Percentage of firmware updates

Syntax: **int i = intent.getIntExtra(GeneralString.FWUpdate_Percent,0);**

Example:

```
if(intent.getAction().equals(GeneralString.Intent_FWUpdate_Percent)
) {
int i = intent.getIntExtra(GeneralString.FWUpdate_Percent,0);
if(i>=0) tv1.setText( Integer.toString(i)); }
```

See Also: FirmwareUpdate

GeneralString.Intent_FWUpdate_Finish

Purpose: Firmware update succeeded.

Parameters:

GeneralString.FWUpdate_ErrorMessage	String Error message.

Syntax:

if(intent.getAction().equals(GeneralString.Intent_FWUpdate_Finish){
Log.d(TAG, "Intent_FWUpdate_Finish"); }

See Also: FirmwareUpdate

1.3. Class

DeviceEvent

Purpose: The **DeviceEvent** enumeration defines the events raised by the UHF RFID device.

Parameters:

PowerSavingMode	Entering power saving mode.
LowBattery	Low battery power. This event is raised once every 120 seconds when if the battery is not being charged.
ScannerFailure	Scanner fails to initialize or not responding to tag scan.
BatteryLose	Battery loss
OverTemperature	Over temperature, pistol sends this event every 60 seconds if temperature is over defined value
Battery_Re_Plug	Battery re-plug

Syntax:

```
public enum DeviceEvent
{
  UnKnown(0),
  PowerSavingMode(0x0001),
  LowBattery(0x0010),
  ScannerFailure(0x0040),
  BatteryLose(0x2000),
  OverTemperature(0x4000),
  Battery_Re_Plug(0x8000);
}
```

FWUpdateErrorCode

Purpose

The **FWUpdateErrorCode** enumeration defines the error code raised by Firmware Update.

Parameters

NoFile	The update file does not exist.
FwUpdateing	Firmware Updating.
SameVersion	The UHF RFID Reader version is the same as the update file version.
ModuleSameVersion	The UHF RFID Reader module version is the same as the update file module version.
DownloadModeFail	The UHF RFID Reader didn't enter firmware upgrade mode.
FileInvalid	The update file is invalid.
UHFFwVersionInvalid	The UHF RFID Reader FW version is invalid.
UHFBatteryLow	The UHF RFID Reader battery must be greater than 30%.
DeviceBatteryLow	The device battery is less than 50%.
Unattached	Not attached the UHF RFID Reader.
UHFBatteryFail	Get the UHF RFID Reader battery fail.
UpdateError	Update error.

Syntax

```
public enum FWUpdateErrorCode
{
NoFile(0),
FwUpdateing(1),
SameVersion(2),
ModuleSameVersion(3),
DownloadModeFail(4),
FileInvalid(5),
UHFFwVersionInvalid(6),
UHFBatteryLow(7),
DeviceBatteryLow(8),
Unattached(9),
UHFBatteryFail(10),
UpdateError(11);
}
```

1.4. Sample Code

```
package com.cipherlab.rfidsample;


import com.cipherlab.rfid.ClResult;
import com.cipherlab.rfid.GeneralString;
import com.cipherlab.rfidapi.RfidManager;
import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;


public class MainActivity extends Activity {
       RfidManager mRfidManager = null;
       String TAG = "RFID_sample";
       TextView tv1 = null;
       @Override
       protected void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
              setContentView(R.layout.activity_main);
              mRfidManager = RfidManager.InitInstance(this);

              IntentFilter filter = new IntentFilter();
              filter.addAction(GeneralString.Intent_RFIDSERVICE_CONNECTED);
              filter.addAction(GeneralString.Intent_RFIDSERVICE_TAG_DATA);
              registerReceiver(myDataReceiver, filter);

              tv1 = (TextView) findViewById(R.id.textView1);
              Button b1 = (Button) findViewById(R.id.button1);
              b1.setOnClickListener(new OnClickListener() {

                     @Override
                     public void onClick(View v) {
```

```
                                int re = mRfidManager.SoftScanTrigger(true); //Scan on
                                //int re = mRfidManager.SoftScanTrigger(false); //Scan off
                                if (re != ClResult.S_OK.ordinal()) {
                                String m = mRfidManager.GetLastError();Log.e(TAG,
"GetLastError = " + m);}
                        }
                });
                }
        @Override
        protected void onDestroy() {
                super.onDestroy();
                unregisterReceiver(myDataReceiver);
                mRfidManager.Release();
        }
        private final BroadcastReceiver myDataReceiver = new BroadcastReceiver()
        {
                @Override
                public void onReceive(Context context, Intent intent) {
                        if
(intent.getAction().equals(GeneralString.Intent_RFIDSERVICE_CONNECTED))
                        {
                                String PackageName = intent.getStringExtra("PackageName");
                                // / make sure this AP does already connect with RFID service
(after call RfidManager.InitInstance(this)
                                String ver = "";
                                ver = mRfidManager.GetServiceVersion();
                                String api_ver = mRfidManager.GetAPIVersion();
                                tv1.setText(PackageName + "," + ver + " , " + api_ver);
                                Toast.makeText(MainActivity.this,
"Intent_RFIDSERVICE_CONNECTED", Toast.LENGTH_SHORT).show();
                        }
                        else
if(intent.getAction().equals(GeneralString.Intent_RFIDSERVICE_TAG_DATA))
                        {
                                // Fetch data from the intent
                               int type = intent.getIntExtra(GeneralString.EXTRA_DATA_TYPE,
-1);
                                int response =
intent.getIntExtra(GeneralString.EXTRA_RESPONSE, -1);
                                double data_rssi =
intent.getDoubleExtra(GeneralString.EXTRA_DATA_RSSI, 0);
                        String PC = intent.getStringExtra(GeneralString.EXTRA_PC);
                        String EPC = intent.getStringExtra(GeneralString.EXTRA_EPC);
```

```
                    String TID = intent.getStringExtra(GeneralString.EXTRA_TID);
                    String ReadData =
intent.getStringExtra(GeneralString.EXTRA_ReadData);
                    int EPC_length =
intent.getIntExtra(GeneralString.EXTRA_EPC_LENGTH, 0);
                    int TID_length =
intent.getIntExtra(GeneralString.EXTRA_TID_LENGTH, 0);
                    int ReadData_length =
intent.getIntExtra(GeneralString.EXTRA_ReadData_LENGTH, 0);

                    String Data = "EPC = " + EPC + "\r TID = " + TID;
                    tv1.setText(Data);
                    Log.w(TAG, "++++ [Intent_RFIDSERVICE_TAG_DATA] ++++");
                    Log.d(TAG, "[Intent_RFIDSERVICE_TAG_DATA] type=" + type + ", response=" + response + ", data_rssi="+data_rssi  );
                    Log.d(TAG, "[Intent_RFIDSERVICE_TAG_DATA] PC=" + PC );
                    Log.d(TAG, "[Intent_RFIDSERVICE_TAG_DATA] EPC=" + EPC );
                    Log.d(TAG, "[Intent_RFIDSERVICE_TAG_DATA] EPC_length=" + EPC_length );
                    Log.d(TAG, "[Intent_RFIDSERVICE_TAG_DATA] TID=" + TID );
                    Log.d(TAG, "[Intent_RFIDSERVICE_TAG_DATA] TID_length=" + TID_length );
                    Log.d(TAG, "[Intent_RFIDSERVICE_TAG_DATA] ReadData=" + ReadData );
                    Log.d(TAG, "[Intent_RFIDSERVICE_TAG_DATA] ReadData_length=" + ReadData_length );
                    }
              }
       };
       }
```

Chapter 2

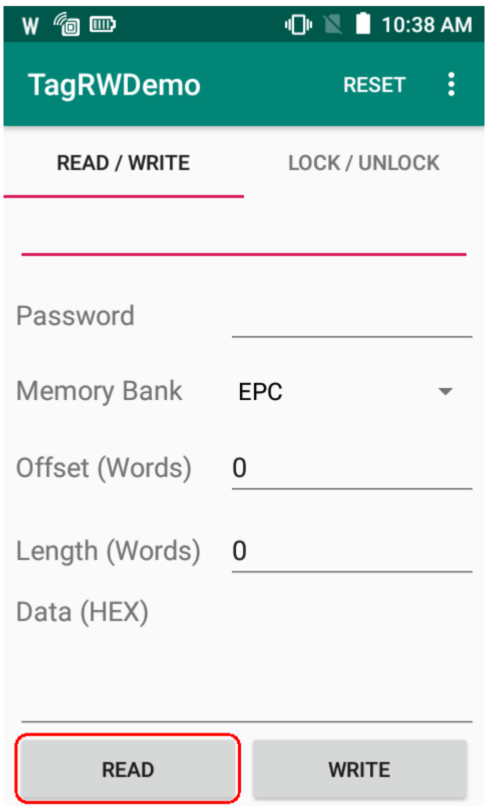
Tag Access Demonstration

This chapter will demonstrate an app accessing UHF RFID tags. Also the sample code of the app is provided for your reference.

IN THIS CHAPTER

2.1. Reade/Write

Launch the TagRWDemo app as the picture illustrated below. The Read/Write tab shows up by default. Tap on the READ button to scan tags nearby. When a tag is scanned with success, the tag data displays.



Password

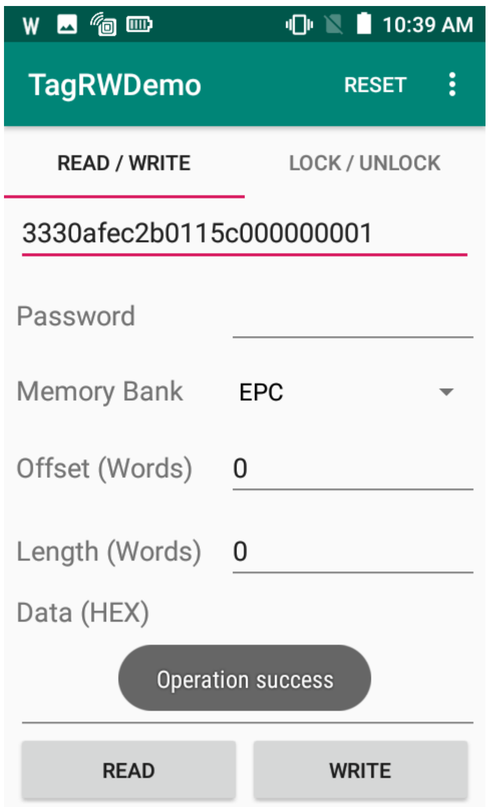
Memory Bank EPC

Offset (Words) 0

Length (Words) 0

Data (HEX)

Tapping the READ button



3330afec2b0115c000000001

Password

Memory Bank EPC

Offset (Words) 0

Length (Words) 0

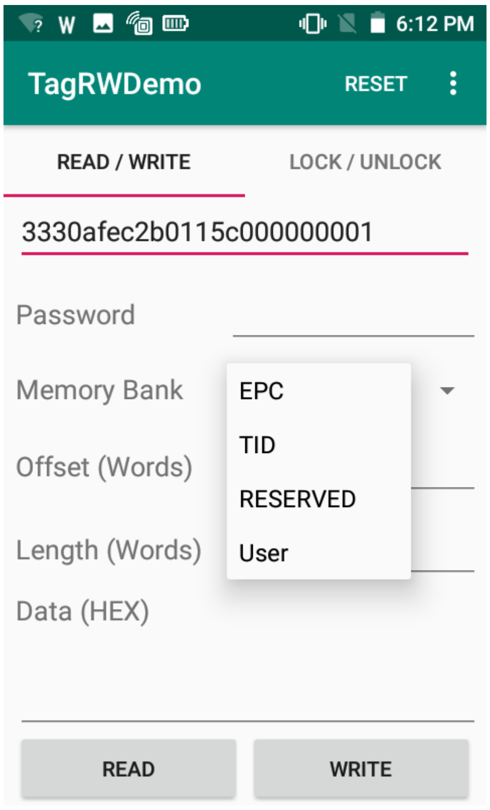
Data (HEX)

Operation success

A tag scanned with success

Note that the app is designed to edit a particular tag. Therefore, the physical trigger of the UHF RFID reader is disabled once the app is launched. The trigger is enebled again when you exit the app.

The tag memory consists of four memory banks including EPC, TID, Reserved, and User. As the picture shown below, click the Memory Bank drop-down menu to select from banks.

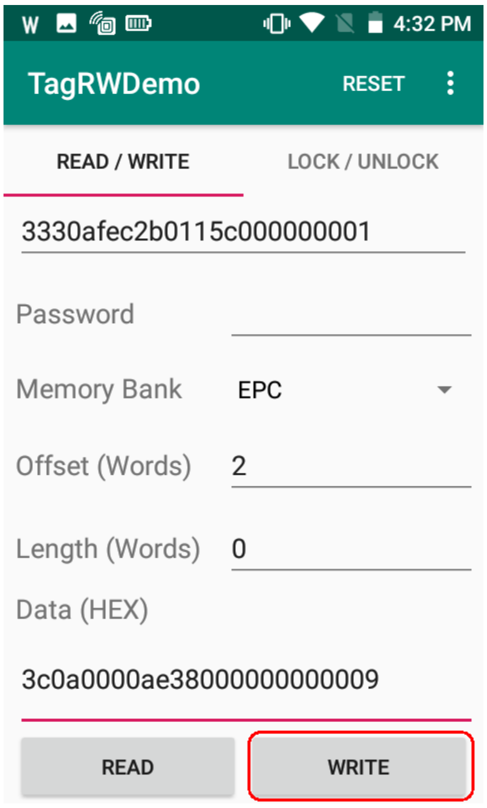


Tapping the Memory Bank drop-down menu

2.1.1. EPC

The EPC bank contains CRC (cyclic-redundancy check), PC (Protocol Control), and EPC (Electronic Product Code).

To edit the EPC data, please specify 2 words (1 word = 2 bytes) in the Offset field and type new data in Hexadecimal in the Data (HEX) field. Make sure the new data is typed correctly, tap the WRITE button.



3330afec2b0115c000000001

Password

Memory Bank EPC

Offset (Words) 2

Length (Words) 0

Data (HEX)

3c0a0000ae38000000000009

Editing the EPC memory bank

2.1.2. TID

The TID bank, containing an 8-bit ISO 15963 allocation class identifier, is read only.

2.1.3. Reserved

The Reserved bank contains the kill and access passwords if passwords are implemented on the tag.

KILL PASSWORD

A 32-bit "**Kill**" password allows a tag to be permanently silenced.

- The default Kill password value is zero.

- The **Kill** command will only be executed when the password has been set to a non-zero value.

ACCESS PASSWORD

A 32-bit "**Access**" password allows the tag to transition to the **Secured** state.

- A tag in the **Secured** state can execute all **Access** commands, including writing to locked blocks.

Reserved memory can be read-locked.

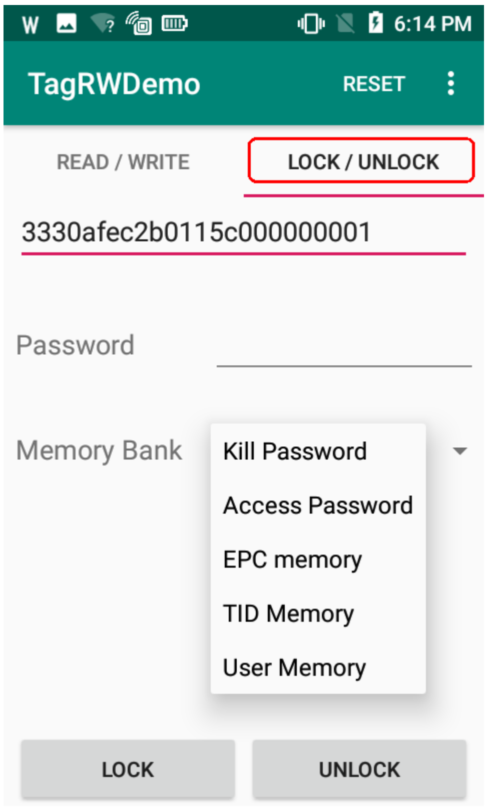
2.1.4. User Memory

The User bank is an optional memory area containing user-specific data.

2.2. LOCK/UNLOCK

Tap the LOCK/UNLOCK tab to lock or unlock the selected memory bank.

Type the password in the Password field. Then tap the Memory Bank drop-down menu to select a memory bank to be locked or unlocked.



Locking/unlocking the selected memory bank

When the password and the memory bank are correctly specified, tap the LOCK or UNLOCK button.

2.3. Sample Code of the Demonstration App

```
package sw.programme.rfid.uhf.demo;

import android.content.Context;
import android.content.Intent;
import android.content.res.Resources;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.text.InputFilter;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;

import com.cipherlab.rfid.GeneralString;
import com.cipherlab.rfid.RFIDMemoryBank;


public class ReadWriteFragment extends Fragment {

    private EditText mEPC;
    private EditText mPassword;
    private EditText mOffset;
    private EditText mLength;
    private EditText mData;
    private Spinner mBank;
    private Button mReadBtn;
    private Button mWriteBtn;

    private final AppHelper helper = AppHelper.getInstance();

    public ReadWriteFragment() {
        // Required empty public constructor
    }
```

```
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
   }

   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,
                      Bundle savedInstanceState) {
      // Inflate the layout for this fragment
      final  View  rootView  =  inflater.inflate(R.layout.fragment_access,  container,
false);

      mEPC = rootView.findViewById(R.id.edt_access_epc);
      mPassword = rootView.findViewById(R.id.edt_access_password);
      InputFilter[] FilterArray1 = new InputFilter[1];
      FilterArray1[0] = new InputFilter.LengthFilter(8);
      mPassword.setFilters(FilterArray1);

      mOffset = rootView.findViewById(R.id.edt_access_offset);
      InputFilter[] FilterArray2 = new InputFilter[1];
      FilterArray2[0] = new InputFilter.LengthFilter(3);
      mOffset.setFilters(FilterArray2);
      mLength = rootView.findViewById(R.id.edt_access_length);
      mLength.setFilters(FilterArray2);
      mData = rootView.findViewById(R.id.edt_access_data);

      Resources res = getResources();
      mBank = rootView.findViewById(R.id.select_access_bank);
      ArrayAdapter<String> adapter1 = new ArrayAdapter<String>(rootView.getContext(),
            R.layout.dropdown_item, res.getStringArray(R.array.bank_option));
      mBank.setAdapter(adapter1);
      mBank.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {

         @Override
         public void onItemSelected(AdapterView<?> parentView, View selectedItemView,
int position, long id) {



         }

         @Override
```

```
    public void onNothingSelected(AdapterView<?> parentView) {

    }
});

mReadBtn = rootView.findViewById(R.id.btn_read);
mReadBtn.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        mData.setText("");
        String epc_data = mEPC.getText().toString();
        if (StringEX.IsNullOrEmpty(epc_data)) {
            helper.Inventory();
        } else {

            byte[] EPCByteArray = GetEpcData();
            if (EPCByteArray == null)
                return;

            byte[] PasswordByteArray = GetPassWordData();
            if (PasswordByteArray == null)
                return;

            int arg1 = GetOffset();
            if (arg1 < 0)
                return;
            int arg2 = GetLength();
            if (arg2 < 0)
                return;

            switch (mBank.getSelectedItemPosition()) {
                case 0:
                    helper.ReadTag(PasswordByteArray, EPCByteArray,
                    RFIDMemoryBank.EPC, arg1, arg2);
                    break;
                case 1:
                    helper.ReadTag(PasswordByteArray, EPCByteArray,
                    RFIDMemoryBank.TID, arg1, arg2);
                    break;
                case 2:
                    helper.ReadTag(PasswordByteArray, EPCByteArray,
                    RFIDMemoryBank.Reserved, arg1, arg2);
                    break;
                case 3:
                    helper.ReadTag(PasswordByteArray, EPCByteArray,
                    RFIDMemoryBank.User, arg1, arg2);
                    break;
            }
        }
    }
});

mWriteBtn = rootView.findViewById(R.id.btn_write);
mWriteBtn.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        String epc_data = mEPC.getText().toString();

        if (StringEX.IsNullOrEmpty(epc_data)) {
            StringEX.ShowMessage(getActivity(), R.string.MSG_EpcEmpty);
            return;
        } else {

            byte[] EPCByteArray = GetEpcData();
            if (EPCByteArray == null)
                return;

            byte[] PasswordByteArray = GetPassWordData();
            if (PasswordByteArray == null)
                return;

            byte[] DataByteArray = GetData();
            if (DataByteArray == null)
                return;

            int arg1 = GetOffset();
            if (arg1 < 0)
                return;

            switch (mBank.getSelectedItemPosition()) {
                case 0:
                    helper.WriteTag(PasswordByteArray, EPCByteArray,
                    RFIDMemoryBank.EPC, arg1, DataByteArray);
                    break;
                case 1:
                    helper.WriteTag(PasswordByteArray, EPCByteArray,
                    RFIDMemoryBank.TID, arg1, DataByteArray);
                    break;
                case 2:
                    helper.WriteTag(PasswordByteArray, EPCByteArray,
                    RFIDMemoryBank.Reserved, arg1, DataByteArray);
                    break;
                case 3:
                    helper.WriteTag(PasswordByteArray, EPCByteArray,
                    RFIDMemoryBank.User, arg1, DataByteArray);
                    break;
            }
        }
    }
});

if (helper.GetConnectionStatus())
    EnableControl(true);
else
    EnableControl(false);
return rootView;
}

private byte[] GetEpcData() {
    String epc = mEPC.getText().toString();
    if (StringEX.IsNullOrEmpty(epc)) {
        StringEX.ShowMessage(getActivity(), R.string.MSG_EpcEmpty);
        return null;
    }

    if (epc.length() % 2 == 1) {
        StringEX.ShowMessage(getActivity(), R.string.MSG_BinaryError);
        return null;
    }

    byte[] ByteArray;
    try {
        Log.i(AppHelper.TAG, "EPC = " + epc);
        ByteArray = UtilityHelp.StringToByteArray(epc);
    } catch (Exception e) {
            StringEX.ShowMessage(getActivity(), R.string.MSG_IncorrectEPC);
            return null;
        }

        return ByteArray;
    }

    private byte[] GetPassWordData() {
        String password = mPassword.getText().toString();
        byte[] PasswordByteArray;

        if (StringEX.IsNullOrEmpty(password)) {
            PasswordByteArray = new byte[]{(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte)
0x00};
        } else {
            if (password.length() != 8) {
                StringEX.ShowMessage(getActivity(), R.string.MSG_PasswordLengthError);
                return null;
            }

            try {
                Log.i(AppHelper.TAG, "Password = " + password);
                PasswordByteArray = UtilityHelp.StringToByteArray(password);
            } catch (Exception e) {
                StringEX.ShowMessage(getActivity(), R.string.MSG_IncorrectPassword);
                return null;
            }
        }
        return PasswordByteArray;
    }

    private byte[] GetData() {
        String data = mData.getText().toString();
        if (StringEX.IsNullOrEmpty(data)) {
            StringEX.ShowMessage(getActivity(), R.string.MSG_DataEmpty);
            return null;
        }

        if (data.length() % 2 == 1) {
            StringEX.ShowMessage(getActivity(), R.string.MSG_BinaryError);
        return null;
    }

    byte[] DataByteArray;
    try {
        DataByteArray = UtilityHelp.StringToByteArray(data);
    } catch (Exception e) {
        StringEX.ShowMessage(getActivity(), R.string.MSG_IncorrectData);
        return null;
    }
    return DataByteArray;
}

private int GetOffset() {
    String offset = mOffset.getText().toString();
    if (StringEX.IsNullOrEmpty(offset)) {
        StringEX.ShowMessage(getActivity(), R.string.MSG_StartEmpty);
        return -1;
    }
    int value = Integer.parseInt(offset);
    return (value * 2);
}

private int GetLength() {
    String length = mLength.getText().toString();
    if (StringEX.IsNullOrEmpty(length)) {
        StringEX.ShowMessage(getActivity(), R.string.MSG_LengthEmpty);
        return -1;
    }
    int value = Integer.parseInt(length);
    return (value * 2);
}

public void ResetField() {
    mEPC.setText("");
    mPassword.setText("");
    mOffset.setText("0");
    mLength.setText("0");
    mData.setText("");
    mBank.setSelection(0);
    }

    public void EnableControl(boolean flag) {
        if (mReadBtn != null)
            mReadBtn.setEnabled(flag);
        if (mWriteBtn != null)
            mWriteBtn.setEnabled(flag);
    }

    public String GetEPC() {
        return mEPC.getText().toString();
    }

    public void Process(Context context, Intent intent) {
        int type = intent.getIntExtra(GeneralString.EXTRA_DATA_TYPE, -1);
        Log.i(AppHelper.TAG, "type = " + type);
        int response = intent.getIntExtra(GeneralString.EXTRA_RESPONSE, -1);

        switch (response) {
            case 0: /* OPERATION_SUCCESS */
                Log.i(AppHelper.TAG, "OPERATION_SUCCESS");
                if (type == 1) {
                    String epc_data = mEPC.getText().toString();
                    String EPC = intent.getStringExtra(GeneralString.EXTRA_EPC);

                    Log.i(AppHelper.TAG, "EPC = " + EPC);
                    if (StringEX.IsNullOrEmpty(epc_data)) {
                        if (EPC != null)
                            mEPC.setText(EPC);
                    } else {
                        mData.setText(EPC);
                    }
                } else if (type == 3) {
                    String                          ReadData                            =
intent.getStringExtra(GeneralString.EXTRA_ReadData);
                    Log.i(AppHelper.TAG, "ReadData = " + ReadData);
                    mData.setText(ReadData);
                }
                break;
            case 1: /* OPERATION_FINISH */
```

```
                Log.i(AppHelper.TAG, "OPERATION_FINISH");
                StringEX.ShowMessage(context, R.string.MSG_OperationSuccess);
                break;
            case 2: /* OPERATION_FAIL */
            case 7:
                Log.i(AppHelper.TAG, "OPERATION_FAIL");
                StringEX.ShowMessage(context, R.string.MSG_OperationFail);
                break;
            case 3: /* TAG_LOCK */
                Log.i(AppHelper.TAG, "TAG_LOCK");
                StringEX.ShowMessage(context, R.string.MSG_TagLocked);
                break;
            case 6:
                Log.i(AppHelper.TAG, "Access Password Fail");
                StringEX.ShowMessage(context, R.string.MSG_AccessPasswordFail);
                break;
            case 251: /* DEVICE_BUSY */
                Log.i(AppHelper.TAG, "DEVICE_BUSY");
                StringEX.ShowMessage(context, R.string.MSG_OperationFail);
                break;
            default:
                Log.i(AppHelper.TAG, "response = " + response);
                break;
        }
    }
}
```

Appendix I

Response Code Instructions

Value	Instruction
ClResult.S_OK	Successful completion of request
ClResult.S_ERR	Unknown error
ClResult.ERR_NotSupport	Symbology not supported
ClResult.ERR_InvalidParameter	Invalid parameter