

MagTek Reader EMV Flowchart

Secure Card Reader Authenticator Programmer's Reference

November 2019

Manual Part Number:
D100005143-10

REGISTERED TO ISO 9001:2015

Information in this publication is subject to change without notice and may contain technical inaccuracies or graphical discrepancies. Changes or improvements made to this product will be updated in the next publication release. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of MagTek, Inc.

MagTek® is a registered trademark of MagTek, Inc.
MagneSafe® is a registered trademark of MagTek, Inc.
iDynamo™, and uDynamo are trademarks of MagTek, Inc.
eDynamo™, Dynamag, and DynaMAX are trademarks of MagTek, Inc.

The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by MagTek is under license.

Android™ is a trademark of Google LLC.

IOS is a trademark or registered trademark of Cisco in the U.S. and other countries and is used by Apple Inc. under license.

Microsoft®, Windows® and .NET® are registered trademarks of Microsoft Corporation.

EMV® is a registered trademark in the U.S. and other countries and an unregistered trademark elsewhere. The EMV trademark is owned by EMVCo, LLC. The Contactless Indicator mark, consisting of four graduating arcs, is a trademark owned by and used with permission of EMVCo, LLC.

All other system names and product names are the property of their respective owners.

Table 0.1 Revisions

Rev Number	Date	Notes
10	11/15/2019	Initial Release

SOFTWARE LICENSE AGREEMENT

IMPORTANT: YOU SHOULD CAREFULLY READ ALL THE TERMS, CONDITIONS AND RESTRICTIONS OF THIS LICENSE AGREEMENT BEFORE INSTALLING THE SOFTWARE PACKAGE. YOUR INSTALLATION OF THE SOFTWARE PACKAGE PRESUMES YOUR ACCEPTANCE OF THE TERMS, CONDITIONS, AND RESTRICTIONS CONTAINED IN THIS AGREEMENT. IF YOU DO NOT AGREE WITH THESE TERMS, CONDITIONS, AND RESTRICTIONS, PROMPTLY RETURN THE SOFTWARE PACKAGE AND ASSOCIATED DOCUMENTATION TO THE ADDRESS ON THE FRONT PAGE OF THIS DOCUMENT, ATTENTION: CUSTOMER SUPPORT.

TERMS, CONDITIONS, AND RESTRICTIONS

MagTek, Incorporated (the "Licensor") owns and has the right to distribute the described software and documentation, collectively referred to as the "Software."

LICENSE: Licensor grants you (the "Licensee") the right to use the Software in conjunction with MagTek products. LICENSEE MAY NOT COPY, MODIFY, OR TRANSFER THE SOFTWARE IN WHOLE OR IN PART EXCEPT AS EXPRESSLY PROVIDED IN THIS AGREEMENT. Licensee may not decompile, disassemble, or in any other manner attempt to reverse engineer the Software. Licensee shall not tamper with, bypass, or alter any security features of the software or attempt to do so.

TRANSFER: Licensee may not transfer the Software or license to the Software to another party without the prior written authorization of the Licensor. If Licensee transfers the Software without authorization, all rights granted under this Agreement are automatically terminated.

COPYRIGHT: The Software is copyrighted. Licensee may not copy the Software except for archival purposes or to load for execution purposes. All other copies of the Software are in violation of this Agreement.

TERM: This Agreement is in effect as long as Licensee continues the use of the Software. The Licensor also reserves the right to terminate this Agreement if Licensee fails to comply with any of the terms, conditions, or restrictions contained herein. Should Licensor terminate this Agreement due to Licensee's failure to comply, Licensee agrees to return the Software to Licensor. Receipt of returned Software by the Licensor shall mark the termination.

LIMITED WARRANTY: Licensor warrants to the Licensee that the disk(s) or other media on which the Software is recorded are free from defects in material or workmanship under normal use.

THE SOFTWARE IS PROVIDED AS IS. LICENSOR MAKES NO OTHER WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Because of the diversity of conditions and PC hardware under which the Software may be used, Licensor does not warrant that the Software will meet Licensee specifications or that the operation of the Software will be uninterrupted or free of errors.

IN NO EVENT WILL LICENSOR BE LIABLE FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE, OR INABILITY TO USE, THE SOFTWARE. Licensee's sole remedy in the event of a defect in material or workmanship is expressly limited to replacement of the Software disk(s) if applicable.

GOVERNING LAW: If any provision of this Agreement is found to be unlawful, void, or unenforceable, that provision shall be removed from consideration under this Agreement and will not affect the enforceability of any of the remaining provisions. This Agreement shall be governed by the laws of the State of California and shall inure to the benefit of MagTek, Incorporated, its successors or assigns.

ACKNOWLEDGMENT: LICENSEE ACKNOWLEDGES THAT HE HAS READ THIS AGREEMENT, UNDERSTANDS ALL OF ITS TERMS, CONDITIONS, AND RESTRICTIONS, AND AGREES TO BE BOUND BY THEM. LICENSEE ALSO AGREES THAT THIS AGREEMENT SUPERSEDES ANY AND ALL VERBAL AND WRITTEN COMMUNICATIONS BETWEEN LICENSOR AND LICENSEE OR THEIR ASSIGNS RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

QUESTIONS REGARDING THIS AGREEMENT SHOULD BE ADDRESSED IN WRITING TO MAGTEK, INCORPORATED, ATTENTION: CUSTOMER SUPPORT, AT THE ADDRESS LISTED IN THIS DOCUMENT, OR E-MAILED TO SUPPORT@MAGTEK.COM.

Table of Contents

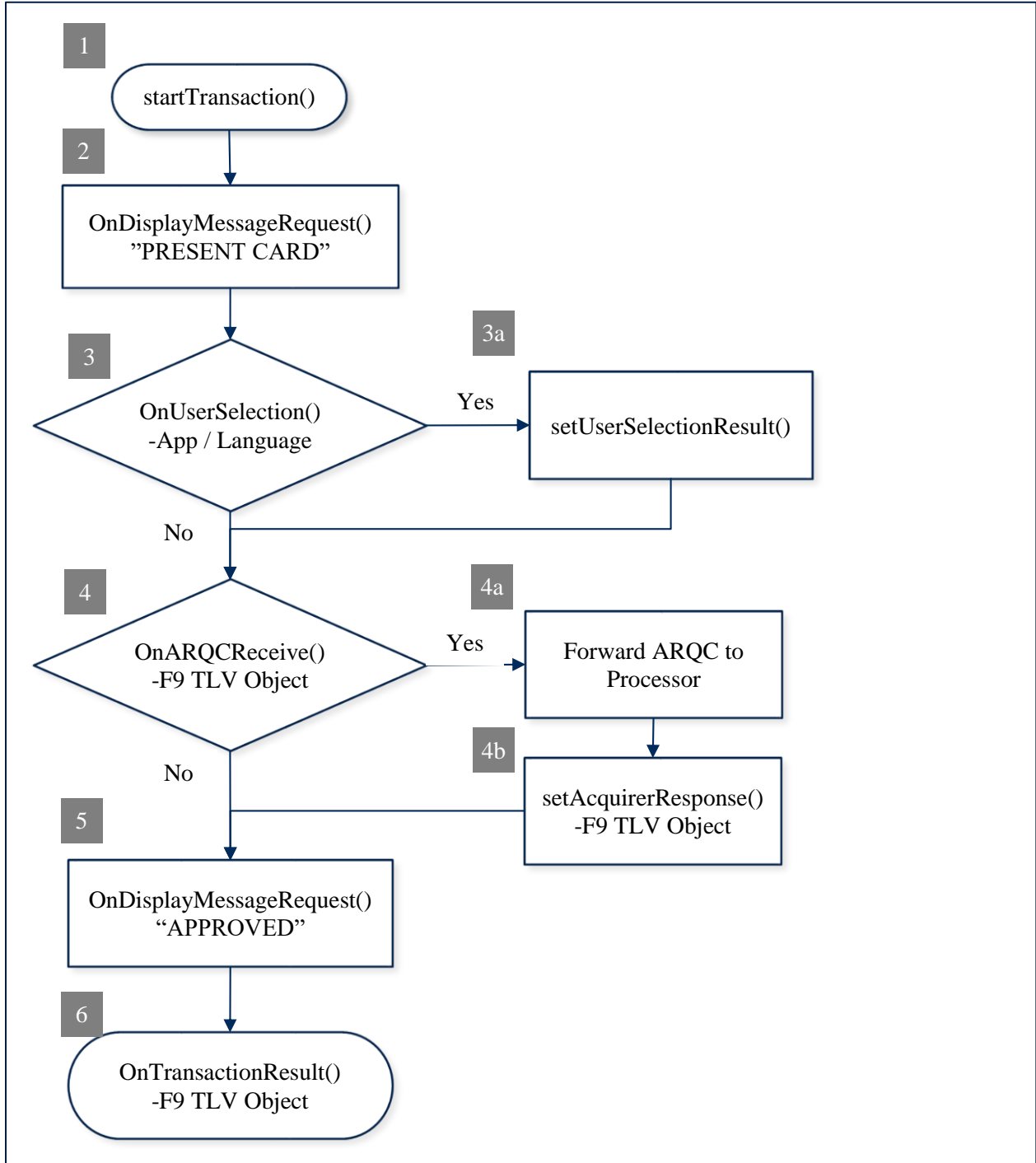
SOFTWARE LICENSE AGREEMENT	3
Table of Contents	5
1 Introduction	6
2 EMV Transaction Flow	7
2.1 Flow Chart.....	7
2.2 Sample Flow Code: C#.....	8
2.3 Sample Flow Code: C++	11
2.4 Sample Flow Code: Java.....	14
2.5 Sample Flow Code: Android (Java)	17
2.6 Sample Flow Code: iOS	20

1 Introduction

This document provides EMV transaction flow instructions for software developers who want to create software solutions that include a MagTek Secure Card Reader / Authenticator (SCRA) device connected to a Windows PC, Android, or iOS device.

2 EMV Transaction Flow

2.1 Flow Chart



2.2 Sample Flow Code: C#

```
// #1

MTSCRA m_SCRA = new MTSCRA();

// Delegate the MTSCRA Events.
m_SCRA.OnDisplayMessageRequest += OnDisplayMessageRequest;
.

// Assign parameters.
byte timeLimit = 0x3C;
byte cardType = 0x07;
byte option = 0x00;
byte[] amount = new byte[] { 0x00, 0x00, 0x00, 0x00, 0x15, 0x00 };
byte transactionType = 0x00; // Purchase
byte[] cashBack = new byte[] { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
byte[] currencyCode = new byte[] { 0x08, 0x40 };
byte reportingOption = 0x02; // All Status Changes

// Start transaction.
long result = m_SCRA.startTransaction(
timeLimit,
cardType,
option,
amount,
transactionType,
cashBack,
currentCode,
reportingOption);
```

```
// #2

protected void OnDisplayMessageRequest(obj sender, byte[] data)
{
    String message;

    // Get the message.
    if (data != NULL)
    {
        message = System.Text.Encoding.UTF8.GetString(data);
    }

    // A data size of 0 is an instruction to clear the display.
    if (data.Length == 0)
    {
        // Clear the display.
    }
}
```


2 - EMV Transaction Flow

```
}  
}
```

```
// #3  
  
protected void OnUserSelectionRequest(object sender, byte[] data)  
{  
    /* data[0]    - selection type  
       data[1]    - timeout  
       data[2..n] - remainder contains zero-terminated string items */  
  
    // display/retrieve user selection.  
    .  
    // set status and selection result.  
    m_SCRA.setUserSelectionResult(status, selection);  
}
```

```
// #4  
  
protected void OnARQCReceived(object sender, byte[] data)  
{  
    /* data[0..1] - ARQC length  
       data[2..n] - remainder contains the ARQC TLV object */  
  
    // #4a Forward ARQC to Processor.  
  
    /* An application function to forward the ARQC  
       to a Processor for approval. */  
    processorResponse = sendARQCToProcesor(data);  
  
    /* No need to send ARQC Response if transaction option  
       had enabled Quick Chip mode. */  
    if (isQuickChipEnabled())  
    {  
        return;  
    }  
  
    // #4b Set Acquirer Response.  
  
    // An application function to build Acquirer Response.  
    byte[] response = buildAcquirerResponse(processorResponse);  
  
    // Set Acquirer Response.
```

2 - EMV Transaction Flow

```
m_SCRA.setAcquirerResponse(response);  
}
```

```
// #5  
  
protected void OnDisplayMessageRequest(obj sender, byte[] data)  
{  
    String message;  
  
    // Get the message.  
    if (data != NULL)  
    {  
        message = System.Text.Encoding.UTF8.GetString(data);  
    }  
  
    // A data size of 0 is an instruction to clear the display.  
    if (data.Length == 0)  
    {  
        // Clear the display.  
    }  
}
```

```
// #6  
  
protected void OnTransactionResult(obj sender, byte[] data)  
{  
    /* data[0]      - Signature Required  
       data[1..2] - Batch Data length  
       data[3..n] - remainder contains the Batch Data TLV object */  
  
    // Parse the TLV from data[.].  
    .  
    // Abstract Approval status from TLV tag "DFDF1A".  
    .  
    // Abstract Signature Required status from TLV tag data[0].  
    .  
}
```

2.3 Sample Flow Code: C++

```
// #1

// Delegate the MTSCRA Events.
::OnDisplayMessageRequest(this->OnDisplayMessageRequest);
.

// Assign parameters.
unsigned char timeLimit = 0x3C;
unsigned char cardType = 0x07;
unsigned char option = 0x00;
const char* amount[] = { 0x00, 0x00, 0x00, 0x00, 0x15, 0x00 };
unsigned char transactionType = 0x00; // Purchase
const char* cashBack[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
const char* currencyCode[] = { 0x08, 0x40 };
unsigned char reportingOption = 0x02; // All Status Changes

// Start transaction.
long result = m_MTSCRA.startTransaction(
timeLimit,
cardType,
option,
amount,
transactionType,
cashBack,
currencyCode,
reportingOption);
```

```
// #2

void OnDisplayMessageRequest(const LPSTR data)
{
    // Get the message.
    if (data != NULL)
    {
        TCHAR* message = HexToASCII(A2T(data));
    }

    // A data size of 0 is an instruction to clear the UI display.
    if (strlen(data) == 0)
    {
        // Clear the display.
    }
}
```

```
// #3

protected void OnUserSelectionRequest(const LPSTR data)
{
    /* data[0]    - selection type
       data[1]    - timeout
       data[2..n] - remainder contains user selection strings delimited
                   by "0x00".
    */

    // display/retrieve status and user selection.
    unsigned char status;
    unsigned char selection;

    // set status and selection result.
    SetUserSelectionResult(status, selection);
}

```

```
// #4

protected void OnARQCReceived(const LPSTR data)
{
    /* data[0..1] - ARQC length
       data[2..n] - remainder contains the ARQC TLV object */

    // #4a Forward ARQC to Processor.

    /* An application function to forward the ARQC
       to a Processor for approval. */
    processorResponse = sendARQCToProcessor(data);

    /* No need to send ARQC Response if transaction option
       had enabled Quick Chip mode. */
    if (isQuickChipEnabled())
    {
        return;
    }

    // #4b Set Acquirer Response.

    /* An application function to build Acquirer Response.
       const char* response = buildAcquirerResponse(processorResponse);

    // Set Acquirer Response.
    SetAcquirerResponse(response);
}

```

2 - EMV Transaction Flow

```
int result = GetResultCode();  
}
```

```
// #5  
  
void OnDisplayMessageRequest(const LPSTR data)  
{  
    // Get the message.  
    if (data != NULL)  
    {  
        TCHAR* message = HexToASCII(A2T(data));  
    }  
  
    // A data size of 0 is an instruction to clear the UI display.  
    if (strlen(data) == 0)  
    {  
        // Clear the display.  
    }  
}
```

```
// #6  
  
void OnTransactionResult(const LPSTR data)  
{  
    /* data[0]      - Signature Required  
       data[1..2] - Batch Data length  
       data[3..n] - remainder contains the Batch Data TLV object */  
  
    // Parse the TLV from data[].  
    .  
    // Abstract Approval status from TLV tag "DFDF1A".  
    .  
    // Abstract Signature Required status from TLV tag data[0].  
    .  
}
```

2.4 Sample Flow Code: Java

```
// #1

m_MTSCRA = new MTSCRA();
m_MTSCRA.init(this);

// Assign parameters.
byte timeLimit = 0x3C;
byte cardType = 0x07;
byte option = 0x00;
byte[] amount = new byte[] { 0x00, 0x00, 0x00, 0x00, 0x15, 0x00 };
byte transactionType = 0x00; // Purchase
byte[] cashBack = new byte[] { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
byte[] currencyCode = new byte[] { 0x08, 0x40 };
byte reportingOption = 0x02; // All Status Changes

// Start transaction.
m_MTSCRA.startTransaction(
timeLimit,
cardType,
option,
amount,
transactionType,
cashBack,
currencyCode,
reportingOption);

int result = m_MTSCRA.getResultCode();
```

```
// #2

public void onDisplayMessageRequest(String data)
{
    String message;

    // Get the message.
    if (data != NULL)
    {
        message = data;
    }

    // A data size of 0 is an instruction to clear the UI display.
    if (data.Length == 0)
    {
        // Clear the display.
    }
}
```

```
}  
}
```

```
// #3  
  
public void onUserSelectionRequest(String data)  
{  
    /* data[0]    - selection type  
       data[1]    - timeout  
       data[2..n] - remainder contains user selection strings delimited  
                   by "\0".  
    */  
  
    // display/retrieve status and user selection.  
    byte status;  
    byte selection;  
    .  
    .  
    // set status and selection result.  
    m_MTSCRA.setUserSelectionResult(status, selection);  
}
```

```
// #4  
  
public void onARQCReceived(String data)  
{  
    /* data[0..1] - ARQC length  
       data[2..n] - remainder contains the ARQC TLV object */  
  
    // #4a Forward ARQC to Processor.  
  
    /* An application function to forward the ARQC  
       to a Processor for approval. */  
    procesorResponse = sendARQCtoProcesor(data);  
  
    /* No need to send ARQC Response if transaction option  
       had enabled Quick Chip mode. */  
    if (isQuickChipEnabled())  
    {  
        return;  
    }  
  
    // #4b Set Acquirer Response.
```

```
// An application function to build Acquirer Response.
byte[] response = buildAcquirerResponse(processorResponse);

// Set Acquirer Response.
m_MTSCRA.setAcquirerResponse(response);
}
```

```
// #5

public void onDisplayMessageRequest(String data)
{
    String message;

    // Get the message.
    if (data != NULL)
    {
        message = data;
    }

    // A data size of 0 is an instruction to clear the UI display.
    if (data.Length == 0)
    {
        // Clear the display.
    }
}
```

```
// #6

public void onTransactionResult(String data)
{
    /* data[0]      - Signature Required
       data[1..2]  - Batch Data length
       data[3..n] - remainder contains the Batch Data TLV object */

    // Parse the TLV from data.
    .
    // Abstract Approval status from TLV tag "DFDF1A".
    .
    // Abstract Signature Required status from TLV tag data[0].
    .
}
```


2.5 Sample Flow Code: Android (Java)

```
// #1

m_MTSCRA = new MTSCRA(this, m_scraHandler);

// Assign parameters.
byte timeLimit = 0x3C;
byte cardType = 0x07;
byte option = 0x00;
byte[] amount = new byte[] { 0x00, 0x00, 0x00, 0x00, 0x15, 0x00 };
byte transactionType = 0x00; // Purchase
byte[] cashBack = new byte[] { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
byte[] currencyCode = new byte[] { 0x08, 0x40 };
byte reportingOption = 0x02; // All Status Changes

// Start transaction.
m_MTSCRA.startTransaction(
timeLimit,
cardType,
option,
amount,
transactionType,
cashBack,
currencyCode,
reportingOption);

int result = m_MTSCRA.getResultCode();
```

```
// #2

protected void onDisplayMessageRequest(byte[] data)
{
    String message;

    // Get the message.
    if (data != NULL)
    {
        message = TLVParser.getTextString(data, 0);
    }

    // A data size of 0 is an instruction to clear the UI display.
    if (data.Length == 0)
    {
        // Clear the display.
    }
}
```

```
// #3

protected void onUserSelectionRequest(byte[] data)
{
    /* data[0]      - selection type
       data[1]      - timeout
       data[2..n]  - remainder contains user selection strings delimited
                    by "\0".
    */

    // display/retrieve status and user selection.
    byte status;
    byte selection;
    .
    .
    // set status and selection result.
    m_MTSCRA.setUserSelectionResult(status, selection);
}
}
```

```
// #4

protected void onARQCReceived(byte[] data)
{
    /* data[0..1] - ARQC length
       data[2..n] - remainder contains the ARQC TLV object */

    // #4a Forward ARQC to Processor.

    /* An application function to forward the ARQC
       to a Processor for approval. */
    processorResponse = sendARQCtoProcessor(data);

    /* No need to send ARQC Response if transaction option
       had enabled Quick Chip mode. */
    if (isQuickChipEnabled())
    {
        return;
    }

    // #4b Set Acquirer Response.

    // An application function to build Acquirer Response.
    byte[] response = buildAcquirerResponse(processorResponse);
}
```

2 - EMV Transaction Flow

```
// Set Acquirer Response.
m_MTSCRA.setAcquirerResponse(response);
}
```

```
// #5
protected void onDisplayMessageRequest(byte[] data)
{
    String message;

    // Get the message.
    if (data != NULL)
    {
        message = TLVParser.getTextString(data, 0);
    }

    // A data size of 0 is an instruction to clear the UI display.
    if (data.Length == 0)
    {
        // Clear the display.
    }
}
```

```
// #6
protected void onTransactionResult(byte[] data)
{
    /* data[0] - Signature Required
       data[1..2] - Batch Data length
       data[3..n] - remainder contains the Batch Data TLV object */

    // Parse the TLV from data.
    .
    // Abstract Approval status from TLV tag "DFDF1A".
    .
    // Abstract Signature Required status from TLV tag data[0].
    .
}
```

2.6 Sample Flow Code: iOS

```
// #1

self.mtSCRALib = [[MTSCRA new];

// Delegate the MTSCRA Events.
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onDisplayMessageRequest:) name:@"
onDisplayMessageRequest" withObject:obj];
.
.

// Assign parameters.
Byte timeLimit = 0x3C;
Byte cardType = 0x07;
Byte option = 0x00;
Byte amount[6] = { 0x00, 0x00, 0x00, 0x00, 0x15, 0x00 };
Byte transactionType = 0x00; // Purchase
Byte cashBack[6] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
Byte currencyCode[2] = { 0x08, 0x40 };
Byte reportingOption = 0x02; // All Status Changes

// Start transaction.
[self.mtSCRALib startTransaction:timeLimit cardType:cardType
option:option amount:amount transactionType:transactionType
cashBack:cashBack currencyCode:currencyCode
reportingOption:reportingOption];
```

```
// #2

-(void)OnDisplayMessageRequest:(NSData *)data
{

    // Get the message.
    NSString* message = [HexUtil stringWithHexString:[self
getHexString:data]];

    // A data size of 0 is an instruction to clear the display.
    if (message.Length == 0)
    {
        // Clear the display.
    }
}
```

```
// #3
-(void)OnUserSelectionRequest:(NSData *)data
{
    /* data[0]      - selection type
       data[1]      - timeout
       data[2..n]  - remainder contains zero-terminated string items */

    // display/retrieve user selection.
    .
    // set status and selection result.
    [self.mtSCRALib.setUserSelectionResult:(Byte)status
selection:(Byte)userSelection];
}
}
```

```
// #4
-(void)OnARQCReceived:(NSData *)data
{
    /* data[0..1] - ARQC length
       data[2..n] - remainder contains the ARQC TLV object */

    // #4a Forward ARQC to Processor.

    /* An application function to forward the ARQC
       to a Processor for approval. */
    processorResponse = sendARQCtoProcessor(data);

    /* No need to send ARQC Response if transaction option
       had enabled Quick Chip mode. */
    if (isQuickChipEnabled())
    {
        return;
    }

    // #4b Set Acquirer Response.

    /* An application function to build Acquirer Response.
       NSData* response = buildAcquirerResponse(processorResponse);

    // Set Acquirer Response.
    [self.mtSCRALib.setAcquirerResponse:(unsigned char *)response
length:(int)response.length];
}
}
```

```
// #5
-(void)OnDisplayMessageRequest:(NSData *)data
{
    // Get the message.
    NSString* message = [HexUtil stringWithHexString:[self
getHexString:data]];

    // A data size of 0 is an instruction to clear the display.
    if (message.Length == 0)
    {
        // Clear the display.
    }
}
```

```
// #6
-(void)OnTransactionResult:(NSData *)data
{
    /* data[0] - Signature Required
    data[1..2] - Batch Data length
    data[3..n] - remainder contains the Batch Data TLV object */

    // Parse the TLV from data[].
    .
    // Abstract Approval status from TLV tag "DFDF1A".
    .
    // Abstract Signature Required status from TLV tag data[0].
    .
}
```