ADK-TEC Programmers Guide

This document is for programmers and developers who want to understand and use the ADK-TEC.

Audience

This guide provides all the information required for application developers to integrate and utilize the functionality of the ADK-TEC.

Organization

This guide is organized as follows:

Introduction: Provides a summary of ADK-TEC.

Contactless Subcomponents: Provides an overview on contactless components, options and flows.

Getting Started: Presents an introduction in ADK-TEC usage.

Programming: Supplies ADK-TEC programming information.

System Setup and Requirements: Supplies information about required dependencies.

PP1000: Supplies information about pairing and PIN transfer with PP1000.

Troubleshooting: Gives solutions for possible issues in ADK-TEC.

Appendix: Links to related documents.

Introduction

ADK-TEC provides technology selection functionality. After starting technology selection you will be informed of the first detected technology. Supported technologies are magnetic cards (requires ADK-MSR), chip cards and contactless cards (requires ADK-EMV and ADK-NFC). Once a technology is detected the application can perform a transaction using that technology.

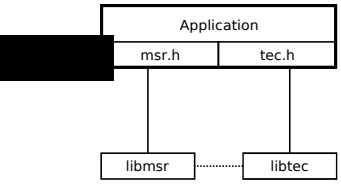
To make use of ADK-TEC you need the following components:

- tec.h
- libtec.so

Additionally you need

- msr.h from ADK-MSR
- libmsr.so from ADK-MSR
- EMV libraries from ADK-EMV
- NFC libraries from ADK-NFC (only if you use NFC)

This is illustrated in the following image (omitting components from ADK-EMV and ADK-NFC):



Android

In Android the ADK-TEC is hidden inside SDI. So application has to use the "Card Detection (23-01)" from ADK-SDI Programmers Guide.

VOS3

In VOS3 ADK-TEC is accessible through SDI. The ADK-TEC-interface is rebuild in ADK-SDI-Client Programmers Guide.

Contactless Subcomponents

There are 3 main contactless subcomponents:

- **EMV** for handling EMV transactions of the schemes
- **NFC Pass Through** for direct L1 access to ISO, MiFare and Felica cards
- **NFC Wallet Manager** for handling Value Added Services (Apple, Google, ...)

Depending on the configuration ADK-TEC will activate the CTLS card or handset and

- either perform the requested *transaction*,
- perform the requested *VAS/wallet* processing,
- or activates a card for low level *L1 access* by the application (ISO; MiFare, Felica, ...).

Standard Use Cases for MSR and (CT+CTLS) EMV transactions

MSR and CT activation and CTLS (Auto) EMV-Transaction

Beforehand the app has to set up the EMV ADK CTLS (with EMV_CTLS_SetupTransaction()).

App calls cts_StartSelection() with CTS_CHIP, CTS_MSR, and CTS_CTLS.

TEC component will:

- Detect if a swipe appears and let the app know about this event so that the app can fetch track 2 data
- Activate contact chip (depending on CTS_NO_POWERON) when a chip card is inserted and let the app know about it
- Do a complete EMV CTLS transaction when an EMV card is tapped and let the app know about it

App evaluates the TEC outcome:

- If(MSR):

  Read out track data and handle MSR transaction
- else if (CT):

  do transaction with help of EMV CT framework (EMV_CT_StartTransaction ...)
- else if (CTLS):

  call EMV_CTLS_ContinueOffline() to fetch result data from transaction

The CTLS txn is already through, card remove point was already reached inside TEC.

MSR and CT activation and CTLS NFC

- MSR Read if a swipe appears and let the app know about it so that the app can fetch track 2 data
- CT Activation if a chip card is inserted and let the app know about it
- CTLS Activation if a card is in the field and let the app know the data from NFC-ADK

In this way the app can decide if it just wants to search for EMV/ISO cards or also wants to detect MiFare or Felica or whatever is supported by NFC.

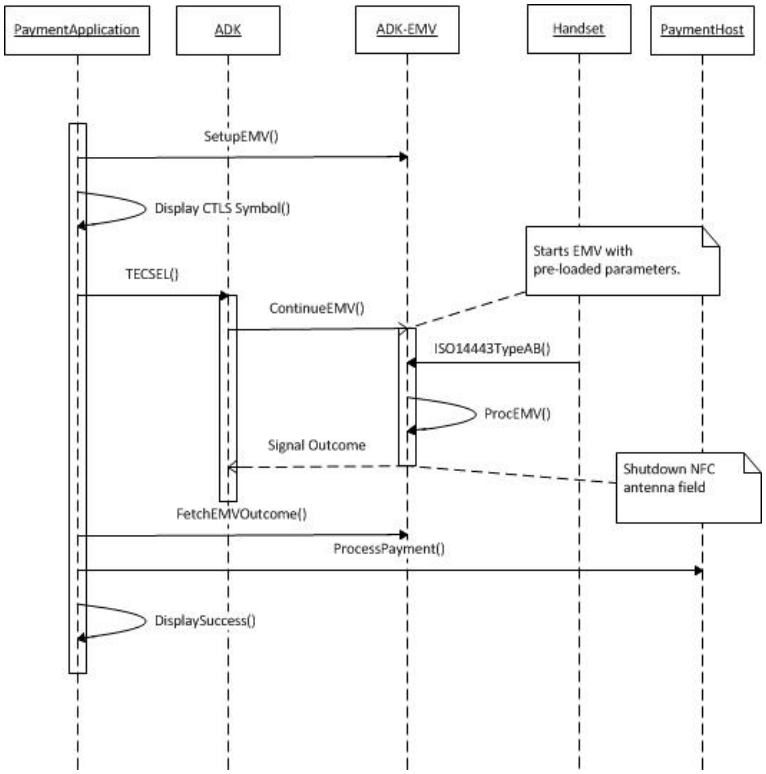
If this option is selected by the App TEC will use the NFC Poll function to perform technology detection.

On detection of a card via NFC Tech Discovery, TEC will report which type of card and then hand over control to app. The App can e.g. on detection of a MiFare use the new NFC ADK commands for Authenticate, Read, Write for handling the card, on detection of an EMV ISO it can use the EMV interface (using a proper EMV activation if needed) to send ISO cards, on detection of Felica can handle Felica via the NFC ADK.

Typical use cases are:

a) EMV/ISO only is activated in TEC

No change, use EMV for detecting cards and once detected we have a proper EMV activation sequence already in place. App uses EMV ADK subsequently to manage the EMV transaction or domestic transaprent ISO card APDU access.



b) NFC only (currently Mifare/Felica/ISO) is activated in TEC

Use NFC only for detecting cards. Once detected report to app. App can use NFC ADK for handling the cards

- NFC_Mifare_XXX for MiFare cards
- NFC_Felica_Exchange for Felcia cards
- NFC_APDU_Exchange for ISO cards

c) NFC AND EMV/ISO is activated

Use NFC for detecting cards. Once a card is detected:

- TEC reports to app if MiFare or Felcia Card reported and use NFC ADK for handling it (see b)
- Do EMV if EMV/ISO card is reported and use EMV ADK for handling it (see a)
- if EMV preloaded: Do EMV automatically (do EMV conform activation)
- If not let the app use EMV ISO interface to handle domestic schemes or allow the app to start an EMV transaction after TEC

Supported use cases:

- MiFare
- Felica
- NFC/ISO (use ISO API of NFC ADK and let NFC ADK activate the card)
- EMV/ISO (use ISO API of EMV ADK and let EMV ADK activate the card)
- EMV/AUTO (automatically perform EMV txn after EMV compliant ISO detection)

MSR and CT activation and CTLS VAS (Value Added Services, Wallets)

ADK-NFC supports EITHER Pass Through (MiFare, Felica) OR VAS. These are mutual exclusive from the design of the NFC ADK. One of these can be combined with MSR, CT and/or EMV CTLS.

Supported use cases with the NFC wallet detection:

- VAS detection only (if no EMV pre configured or EMV not detected)
- VAS detection followed by EMV transaction
- EMV transaction only (if no VAS pre configured or no VAS detected)

Dependencies from EMV and NFC are kept apart from each other. The flow is as follows:

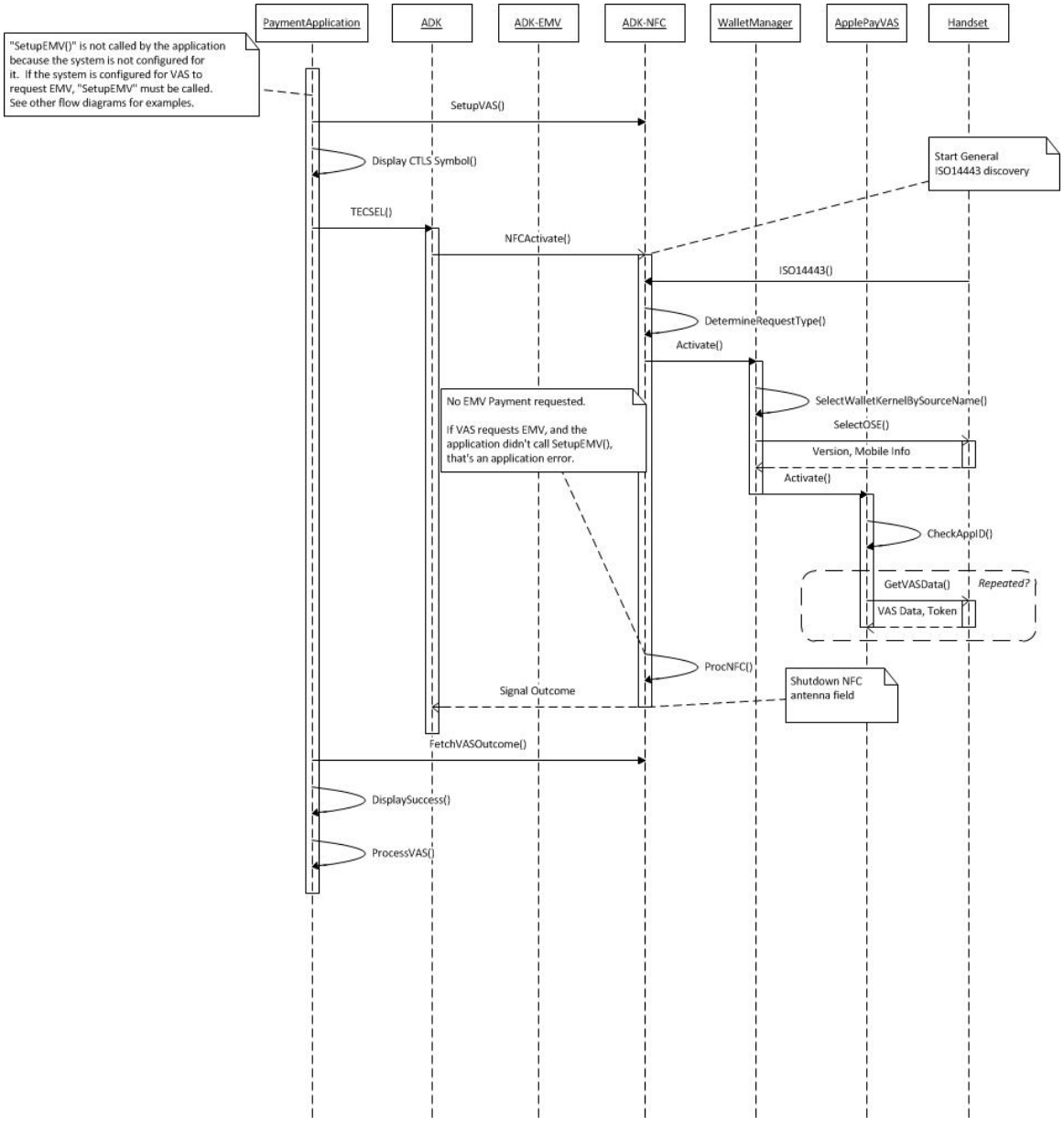
- App: SetupEMV(EMV params) by using EMV-ADK
- App: SetupVAS(VAS params) by using NFC-ADK
- App: Use TEC for technology selection
- TEC: Detects VAS or EMV or MSR (or anything else) by using the API's of NFC, EMV CT, EMV CTLS and MSR
- TEC: If VAS is detected and VAS tells to continue with EMV (without parameter update for EMV): Do EMV without returning to the app. (If VAS tells no EMV needed, return to app at this step)
- App: Fetch EMV TXN outcome (if there was any) by using EMV-ADK
- App: Fetch VAS outcome (if there was any) by using NFC-ADK or TEC response

a) VAS Only

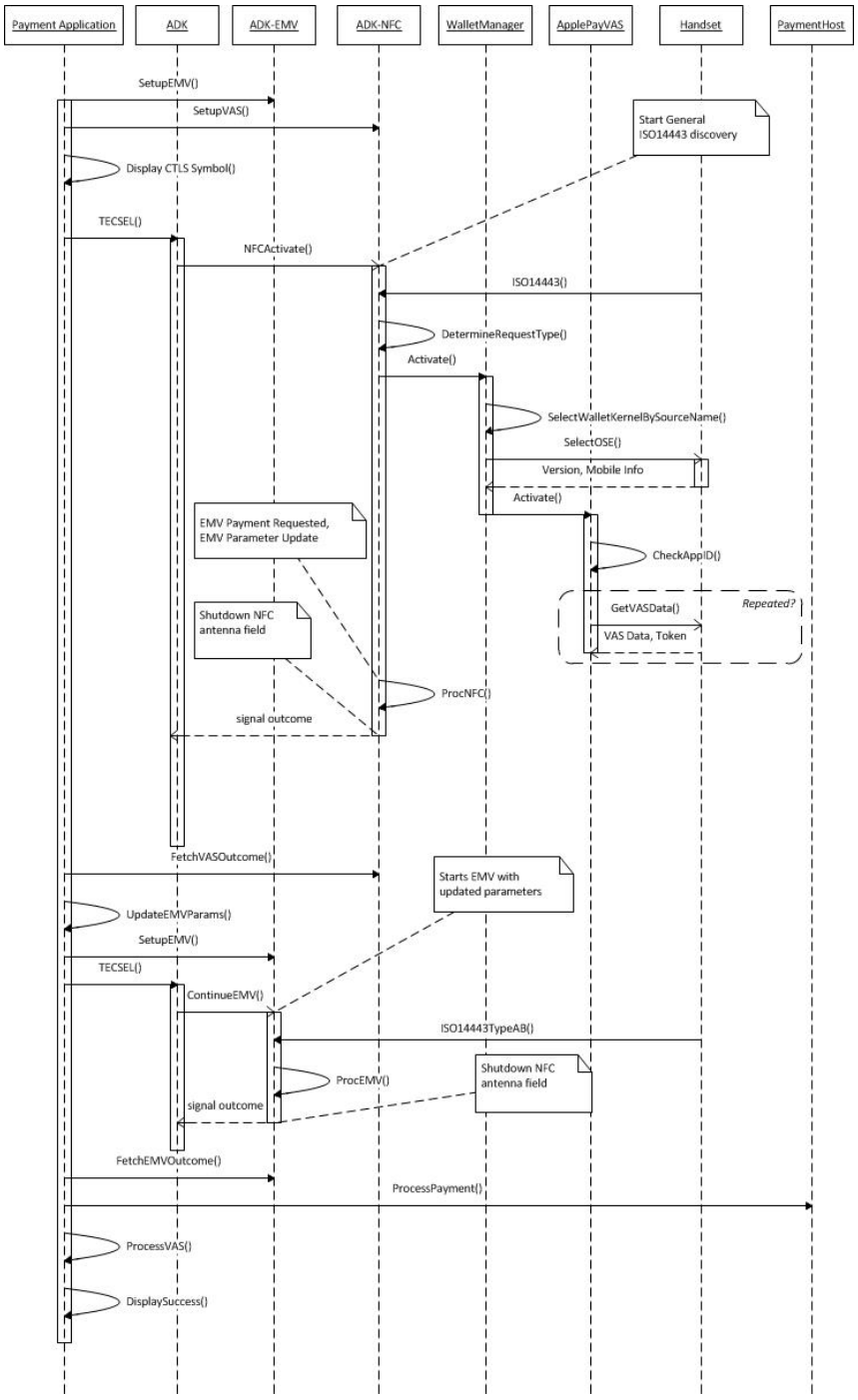
(e.g. the app wants to check coupons and needs to change the EMV amount before continuing with EMV, update EMV params needed) is

configured

- MSR Read if a swipe appears and let the app know about it so that the app can fetch track 2 data
- CT Activation if a chip card is inserted and let the app know about it
- NFC Wallet detection started and depending on the Wallet detection outcome
    - No VAS available/found/needed: Return to App with info 'CTLS detected' 'No VAS Data' 'EMV to follow'
    - VAS detected --> do not EMV: Return to app with info 'CTLS detected' 'VAS Data available' No EMV'



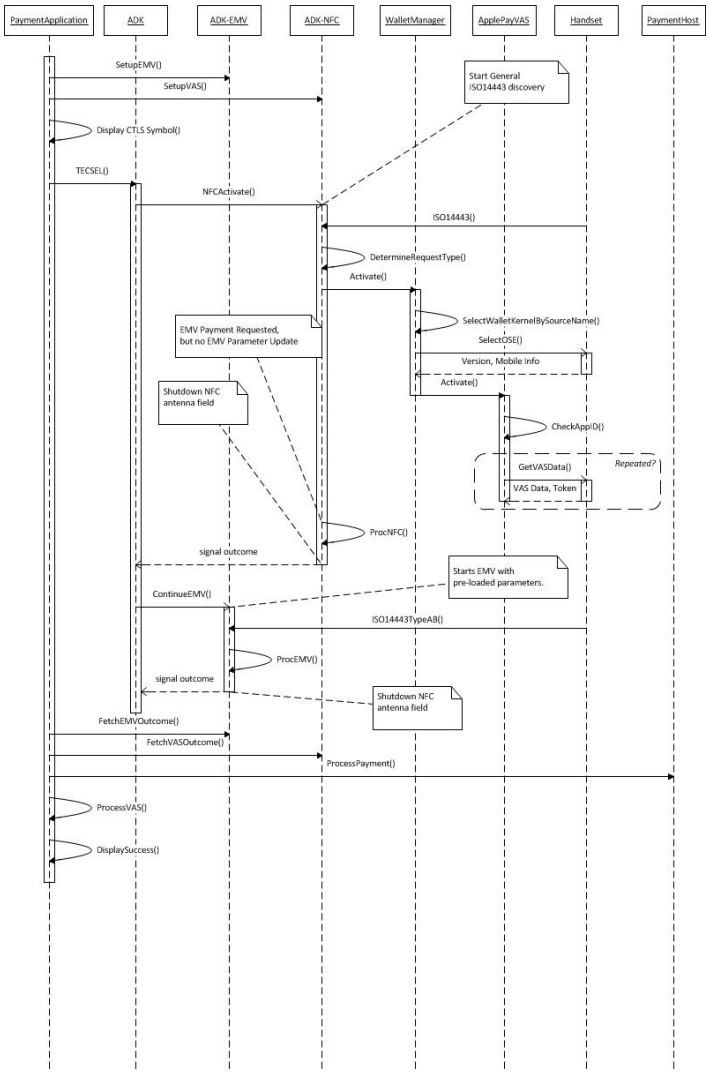
◦ VAS detected --> to be followed by EMV: Return to app with info 'CTLS detected' 'VAS Data available' 'EMV to follow'



b) VAS followed by EMV transaction

'MSR' and 'CT activation' and 'CTLS VAS processing followed by EMV transaction'

- MSR Read if a swipe appears and let the app know about it so that the app can fetch track 2 data
- CT Activation if a chip card is inserted and let the app know about it
- NFC Wallet detection started and depending on the Wallet detection outcome
    - No VAS available/found/needed --> do EMV only: Will do EMV TXN automatically (disabling card and do proper EMV Activation sequence within EMV) and return to App with info 'CTLS detected' 'EMV performed' 'No VAS Data'
    - VAS detected --> do not do EMV: Return to app with info 'CTLS detected' 'EMV not performed' 'VAS Data available'
    - VAS detected --> to be followed by EMV: Will do EMV TXN automatically (disabling card and do proper EMV Activation sequence within EMV) and return to App with info 'CTLS detected' 'EMV performed' 'VAS Data available'



Compared to an EMV transaction without VAS pre-processing the following steps have to be added on apps side:

- Call

  NFC_Terminal_Config(), NFC_VAS_UpdateConfig(), NFC_VAS_PreLoad()

  to configure VAS/Wallet processing before handing control over to TEC
- Once NFC or VAS/Wallet processing is activated, TEC will respond with data in TLV format.

  Evaluate the TEC response for (additional/optional) VAS Data and use it for VAS processing.

  Reference: TEC result data tags
- The processing for EMV remains unchanged.

Getting Started

The following two examples show how to use technology selection (ADK-TEC):

Sample1: Application using ADK-TEC without callback

```
#include "tec/tec.h"#include "msr/msr.h"#include "EMV_CT_Interface.h"#include "EMV_CTLS_Interface.h".... a) MSR, CT, CTLS EMV only// initialize:EMV_CT_InitFramework(...);EMV_CTLS_InitFramework(...);// setup transactionEMV_CTLS_SetupTransaction(...);// start technology selection without callback:if (cts_StartSelection(CTS_CHIP|CTS_CTLS|CTS_MSR, 10, NULL, NULL, NULL, 0) == CTS_OK) b) MSR, CT, CTLS EMV + WALLET (Remark: NFC ADK supports EITHER VAS processing OR PassTrough processing, mutual exclusive)unsigned char options[2] = {0};// initialize:EMV_CT_InitFramework(...);EMV_CTLS_InitFramework(...);NFC_Client_Init(...);NFC_Terminal_Config(...);NFC_VAS_UpdateConfig(...);cts_SetOptions(...); // to set VAS appID// setup transaction:EMV_CTLS_SetupTransaction(...);NFC_VAS_PreLoad(...);options[1] = CTS_VAS_ENABLE;// start technology selection without callback:if (cts_StartSelection(CTS_CHIP|CTS_CTLS|CTS_MSR, 10, NULL, NULL, options, 2) == CTS_OK) c) MSR, CT, CTLS EMV + NFC Pass Through (Remark: NFC ADK supports EITHER VAS processing OR PassTrough processing, mutual exclusive)unsigned char options[16] = {0};// initialize:EMV_CT_InitFramework(...);EMV_CTLS_InitFramework(...);NFC_Client_Init(...);// setup transaction:EMV_CTLS_SetupTransaction(...);options[1] = CTS_NFC_ENABLE;options[1] |= CTS_EMV_AFTER_NFC_ISO; // optionaloptions[15] = 0x0F; // select card types // start technology selection without callback:if (cts_StartSelection(CTS_CHIP|CTS_CTLS|CTS_MSR, 10, NULL, NULL, options, 16) == CTS_OK){ unsigned char technology; unsigned char tlv_response = false; unsigned char data[100]; unsigned short data_len = sizeof(data); int ret; // wait for result: while ((ret = cts_WaitSelection(&technology, data, data_len, 100)) == CTS_IN_PROGRESS) { // if abort request arrived (from GUI, ECR, ...) stop technology selection: if (aborted) cts_StopSelection(); } switch (ret) { case CTS_OK: // technology detected if(technology & CTS_DATA_TLV) { // clear TLV response flag technology &= ~CTS_DATA_TLV; tlv_response = true; } switch (technology) { case CTS_MSR: MSR_GetData(...); MSR_Deactivate(...); break; case CTS_CHIP: EMV_CT_ContinueOffline(...); EMV_CT_ContinueOnline(...); // wait for removal of chip card cts_WaitCardRemoval2(10); break; case CTS_CTLS: if (tlv_response) { // Evaluate the response tags for NFC Pass Through, NFC VAS processing, and EMV processing if configured in b) or c) above. } else { EMV_CTLS_ContinueOffline(...); EMV_CTLS_ContinueOnline(...); } break; } break; case CTS_TIMEOUT: // no technology detected break; case CTS_STOPPED: // technology selection aborted
```

```
break; default: // error (see cts_WaitSelection() for details about all possible return values) break; }}else{
EMV_CTLS_Break();}....
```

Sample2: Application using ADK-TEC with callback

```
#include <pthread.h>#include "tec/tec.h"#include "msr/msr.h"#include "EMV_CT_Interface.h"#include
"EMV_CTLS_Interface.h"....static pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;static pthread_cond_t
condition = PTHREAD_COND_INITIALIZER; static unsigned char technology;static unsigned char data[100];static
unsigned short data_len = sizeof(data);static int ret; static void tecselCallback(void *data){
pthread_mutex_lock(&mutex); ret = cts_WaitSelection(&technology, data, data_len, 0)
pthread_cond_signal(&condition); pthread_mutex_unlock(&mutex);} .... a) MSR, CT, CTLS EMV only//
initialize:EMV_CT_InitFramework(...);EMV_CTLS_InitFramework(...);// setup
transactionEMV_CTLS_SetupTransaction(...);// start technology selection with callback:if
(cts_StartSelection(CTS_CHIP|CTS_CTLS|CTS_MSR, 10, &tecselCallback, NULL, NULL, 0) == CTS_OK) b) MSR, CT, CTLS
EMV + WALLET (Remark: NFC ADK supports EITHER VAS processing OR PassTrough processing, mutual exclusive)unsigned
char options[2] = {0};//
initialize:EMV_CT_InitFramework(...);EMV_CTLS_InitFramework(...);NFC_Client_Init(...);NFC_Terminal_Config(...);N
FC_VAS_UpdateConfig(...);cts_SetOptions(...); // to set VAS appID// setup
transaction:EMV_CTLS_SetupTransaction(...);NFC_VAS_PreLoad(...);options[1] = CTS_VAS_ENABLE;// start technology
selection with callback:if (cts_StartSelection(CTS_CHIP|CTS_CTLS|CTS_MSR, 10, &tecselCallback, NULL, options, 2)
== CTS_OK) c) MSR, CT, CTLS EMV + NFC Pass Through (Remark: NFC ADK supports EITHER VAS processing OR PassTrough
processing, mutual exclusive)unsigned char options[16] = {0};//
initialize:EMV_CT_InitFramework(...);EMV_CTLS_InitFramework(...);NFC_Client_Init(...);// setup
transaction:EMV_CTLS_SetupTransaction(...);options[1] = CTS_NFC_ENABLE;options[1] |= CTS_EMV_AFTER_NFC_ISO; //
optionaloptions[15] = 0x0F; // select card types // start technology selection with callback:if
(cts_StartSelection(CTS_CHIP|CTS_CTLS|CTS_MSR, 10, &tecselCallback, NULL, options, 16) == CTS_OK){ unsigned char
tlv_response = false; pthread_mutex_lock(&mutex); // wait for callback: pthread_cond_wait(&condition, &mutex);
pthread_mutex_unlock(&mutex); switch (ret) { case CTS_OK: // technology detected if(technology & CTS_DATA_TLV) {
// clear TLV response flag technology &= ~CTS_DATA_TLV; tlv_response = true; } switch (technology) { case
CTS_MSR: MSR_GetData(...); MSR_Deactivate(...); break; case CTS_CHIP: EMV_CT_ContinueOffline(...);
EMV_CT_ContinueOnline(...); // wait for removal of chip card cts_WaitCardRemoval2(10); break; case CTS_CTLS: if
(tlv_response) { // Evaluate the response tags for NFC Pass Through, NFC VAS processing, and EMV processing if
configured in b) or c) above. } else { EMV_CTLS_ContinueOffline(...); EMV_CTLS_ContinueOnline(...); } break; }
break; case CTS_TIMEOUT: // no technology detected break; case CTS_STOPPED: // technology selection aborted
break; default: // error (see cts_WaitSelection() for details about all possible return values) break; }}else{
EMV_CTLS_Break();}....
```

Link your application with libtec.so and load the shared library libtec.so on your device. If you don't want to periodically call cts_WaitSelection() you can supply a callback function to cts_StartSelection(). This callback function is called exactly once after the technology selection has finished (due to detected technology, timeout, or error). After this callback function has been called (or even withing the callback function) you can obtain the result by cts_WaitSelection() setting its timeout to 0. A callback function is supported for card removal detection as well (see cts_WaitCardRemoval()).

Programming

Programming and API Principles

The API consists of the following functions:

tec.h	**tecclient.h**
	cts_ConfigureServer()

cts_Version()	cts_Version()
cts_SetTraceCallback()	cts_SetTraceCallback()
cts_SetOptions()	cts_SetOptions()
cts_StartSelection()	cts_StartSelection()
cts_StopSelection()	cts_StopSelection()
cts_WaitSelection()	cts_WaitSelection()
cts_RemoveTechnologies()	cts_RemoveTechnologies()
cts_AddTechnologies()	
cts_WaitCardRemoval()	cts_WaitCardRemoval()
cts_WaitCardRemoval2()	cts_WaitCardRemoval2()
	pintransfer_Pairing()
	pintransfer_MovePin()
ped_SetSendRcvCb()	
ped_Pairing()	
ped_MovePin()	
cts_SetNotificationCallback()	

Some notes regarding the different technologies

In general only one of CTS_MSR, CTS_CHIP, CTS_CTLS is detected but in special cases (see Detecting MSR and CTLS simultaneously and Special behavior on UX devices) two technologies can be detected at once.

- CTS_MSR: If you want to use the magnetic card reader, you do not need to call MSR_Activate() before starting technology selection. ADK-TEC will do this for you. After technology selection finishes and the detected technology is not CTS_MSR, MSR_Deactivate() is internally called as well. So you do not need to do this either. Only if the detected technology is CTS_MSR, ADK-MSR is still activated to allow the application to fetch the magnetic card data with MSR_GetData(). After this you shall call MSR_Deactivate(). If using an UX device MSR_Deactivate() shall be called as well if technology selection detects CTS_CHIP or returns CTS_NO_CHIP (see Special behavior on UX devices).
- CTS_CHIP: If you want to use the chip card reader, you should first call EMV_CT_InitFramework() to enable the contact part of ADK-EMV. After technology selection detects a chip card the card is already powered up (except if you set option CTS_NO_POWERON) and the application can call EMV_CT_ContinueOffline().
- CTS_CTLS: (EMV only, see below for NFC)
    - Card detection only (option CTS_PURE_CARD_DETECTION)

ADK-TEC activates the card by means of EMV_CTLS_SmartReset().

So calling application can continue to work with the card by EMV_CTLS_SmartISO().

And finally it shall call EMV_CTLS_SmartPowerOff() to switch off the RF field.

In case an EMV transaction is desired the EMV_CTLS_SmartPowerOff() has to be called.

And then EMV_CTLS_SetupTransaction() and EMV_CTLS_ContinueOffline().

- EMV transaction

First call EMV_CTLS_InitFramework() and prior to each technology selection you have to call EMV_CTLS_SetupTransaction().

ADK-TEC will internally call EMV_CTLS_ContinueOffline() to detect the card and perform the transaction.

If a contactless card is detected, the application can call EMV_CTLS_ContinueOffline() again to obtain the transaction results.

If no contactless card is detected, ADK-TEC internally calls EMV_CTLS_Break().

One additional remark regarding EMV_CTLS_SetupTransaction(): If ADK-TEC is used, you must not set parameter ServerPollTimeout because in this case ADK-TEC takes care of polling.

Processing NFC with ADK-TEC

This is the general routine used in ADK-TEC for detecting and processing CTLS cards (pseudocode), it should help you to understand how ADK-TEC behaves depending on the various CTLS options.

```
[0] if both CTS_NFC_ENABLE and CTS_VAS_ENABLE are set: exit end[1] if CTS_NFC_ENABLE is set: call
NFC_PT_Polling() if ISO A/B card found and CTS_EMV_AFTER_NFC_ISO is set: goto [3] end exit end[2] if
CTS_VAS_ENABLE is set: call NFC_VAS_Activate() if VAS_DO_PAY is returned: goto [3] end exit end[3] if
CTS_PURE_CARD_DETECTION is set: call EMV_CTLS_SmartReset() else call EMV_CTLS_ContinueOffline() end
```

If it is possible that EMV_CTLS_ContinueOffline() is called by ADK-TEC, application has to call EMV_CTLS_SetupTransaction() before starting technology selection. If ADK-TEC detects a card with NFC_PT_Polling() and no subsequent EMV transaction is started, ADK-TEC keeps the RF field on to allow the application to communicate with this card. In this case the application has to call NFC_PT_FieldOff() and NFC_PT_Close() afterwards. Furthermore the first CTLS LED is left on by ADK-TEC in this case. The application generally wants it to shine while communicating with the card or even wants to switch on further LEDs. So as soon as the application has finished the transaction, it needs to switch off the LEDs or restart idle blinking.

Detecting MSR and CTLS simultaneously

After CTS_CTLS has been detected technology selection can wait a certain amount of time for CTS_MSR before returning the result to the application. If a magnetic card is swiped within this period of time technology selection will return CTS_CTLS|CTS_MSR as technology. The timeout can be configured by the options parameter of cts_StartSelection().

Special behavior on UX devices

When using an UX device it is recommended to activate the MSR UX enhancements:

```
unsigned char options[] = { MSR_UX_ENHANCEMENTS };MSR_SetOptions(options, sizeof(options));
```

This has to be done only once, before the first call of cts_StartSelection(). These enhancements will prevent MSR from reading the magnetic card on insertion. Technology selection will behave as follows:

- If a chip card is inserted CTS_CHIP will be detected but in contrast to other devices MSR_Deactivate() will not be called internally. This is to avoid that MSR data from card removal gets lost. So if the contact transaction fails or application does not want to do contact transaction, it is able to call MSR_GetData() to obtain the MSR data. If there is no MSR data and the card is still inserted, the application can try again to read MSR data after card removal.
- If a card without chip is inserted, technology selection will normally return CTS_NO_CHIP. In case of UX the application is able to set a timeout to cts_StartSelection(). Technology selection waits for this amount of time for MSR data from card removal. If there is MSR data CTS_MSR will be detected, if not CTS_NO_CHIP is returned. In the latter case application can initiate card removal and after that call MSR_GetData().

In both cases (CTS_CHIP detected or CTS_NO_CHIP returned) MSR_Deactivate() is not internally called, so application has to do this manually after optional call of MSR_GetData().

If the MSR UX enhancements are not activated, the former remarks are valid as well. Additionally it is possible that technology selection detects both CTS_MSR and CTS_CHIP in parallel.

System Setup and Requirements

Compiler and Linker Settings

include tec.h and link libtec.so

libtec requires libmsr.

Hardware

ADK-TEC is hardware platform agnostic and supports installation on V/OS and VOS2 terminals.

Software

ADK-TEC is designed to be platform agnostic and will be supported on V/OS and VOS2 terminal operating systems.

Deliverables and Deployment

Packages delivered (x - version number digit):

Package name	Description
`tec-doc-x.x.x-xx.zip`	Documentation

`tec-vos-dev-x.x.x-xx.zip`	VOS **development** package, to be installed in PC build environment
`tec-vos2-dev-x.x.x-xx.zip`	VOS2 **development** package, to be installed in PC build environment

PP1000

Pairing and PIN transfer with PP1000

ADK-TEC is capable of performing pairing a countertop device (CTP) with a PP1000 device and transferring the PIN entered at the PP1000 into the vault of the countertop device. On the PP1000 you only need to install the current AQUILA version, no ADK-TEC component is running on the PP1000. The application running on the CTP needs to include the header file ped.h which is provided by ADK-TEC. Within this file the three functions ped_SetSendRcvCb(), ped_Pairing(), and ped_MovePin() are declared. If you call one of these functions you have to additionally install the library libPP1000.so on the CTP. This library is shipped together with ADK-TEC. ped_Pairing() pairs the two devices. The actual pairing is only to be done once. However, if one of the devices is paired with a third device in between, the devices must be repaired. ped_Pairing() first checks if the two devices are successfully paired and performs the pairing only if this is necessary.

If the pairing is successful, a PIN can be transferred from PP1000 to CTP. The function ped_MovePin() does not collect the PIN on the PP1000, so the PIN entry must be triggered by the application. It can directly send the commands to the PP1000 or use the function pp1000_acceptPin() which is provided by libPP1000. After the PIN has been entered, the PIN can be transferred into the vault of the CTP by calling ped_MovePin(). If this is successfully done, the application can proceed as usual, e.g. call EMV_CT_Send_PIN_Offline() if this is an offline PIN.

The communication between PP1000 and CTP has to be handled on application level. Both ADK-TEC and PP1000 lib are platform independent and do not have communication built in. The application has to call either ped_SetSendRcvCb() (provided by ADK-TEC) or pp1000_registerComs() (provided by PP1000 lib) to set functions that send and receive data to/from the PP1000. So the application can freely decide which communication method it wants to use, e.g. you may use ADK-COM or directly call OS functions.

Troubleshooting

Frequently Asked Questions

Q: cts_WaitSelection->timeout_msec: What is the purpose of this timeout if compare with cts_StartSelection->timeout_sec? Provide use cases.

A: cts_StartSelection->timeout_sec is the timeout for the whole technology selection process, e.g. 30 seconds might be reasonable value. cts_WaitSelection->timeout_msec is the timeout for the cts_WaitSelection() function. It blocks and returns only if the timeout expires (in this case CTS_IN_PROGRESS is returned) or a result is available (something != CTS_IN_PROGRESS is returned). The timeout value to use here depends on your application design. If you have set a callback function to cts_StartSelection(), this callback is invoked as soon as a result is available. So you have to call cts_WaitSelection() exactly once after the callback is invoked, set timeout=0 (waiting makes no sense because you know that a result is available) If you do not want to use callback function you can call cts_WaitSelection() with different timeout values. If you have set timeout in cts_StartSelection() to 30 seconds, the easiest thing to do is set cts_WaitSelection->timeout_msec to 35000 ms (maybe even longer if you set options[8..9] because this may prolongate the technology selection). Then you have to call cts_WaitSelection()

only once, it blocks and returns as soon as a result is available. This works of course only if cts_StartSelection->timeout_sec does not exceed ~60 seconds. If you set cts_WaitSelection->timeout_msec to smaller values you have to call the function in a loop until a result (something != CTS_IN_PROGRESS) is returned. This makes sense if you want to do other things in the same thread while waiting for result of technology selection, e.g. you may want to call cts_StopSelection() if abort request arrived from GUI or ECR. So in this case the timeout depends on the frequency with that you want to do the other things, e.g. a timeout of 0 is possible but will lead to high system load wheras a timeout of 100ms seems reasonable.

Q: Some time ago, upon reviewing our test logs, you pointed out that we should not call the API MSR_Activate() if next we start the selection with the API cts_StartSelection() because the latter activates the reader by itself. And what about the scenario when we want to establish the MSR callback and then use the selection? Here, MSR_Activate() is the only way to establish such a callback. Is this a legal use case to use simultaneously the MSR callback and the selection which, in turn, may have its own callback?

A: No, this is not a legal use case. You should not establish the MSR callback if you use technology selection. This is confusing and not necessary anyway. If MSR data is available, technology selection will finish, so you get the information from TEC, no need to set MSR callback. If you even call MSR_GetData() upon receiving MSR callback, TEC would most likely not be able to detect that MSR data is available and continue waiting for technology (TEC calls MSR_DataAvailable() and as soon as MSR_GetData() is called, the former will return 'no data available'). So please do not do anything like this.

Logging

You have two options to enable logging, choose one of them (if you think this is helpful, you could actually use both at once):

- Register a trace callback function with cts_SetTraceCallback().
- Use ADK-LOG: Configure logging channel "TEC" by means of log control panel.

Appendix

Appendix is empty.