MULTOS tm

GLDA

Guide to Loading and Deleting

MAO-DOC-TEC-008 v2.29

Published by

MULTOS Limited,
350 Longwater Avenue,
Reading,
Berkshire,
RG2 6GF,
UK.

General Enquiries

Email: dev.support@multos.com
Web: http://www.multos.com

Document References

All references to other available documentation is followed by the document acronym in square [ ] brackets. Details of the content of these documents can be found in the MULTOS Guide to Documentation, and the latest versions are always available from the MULTOS web site at www.multos.com

Data References

All references to MULTOS data can be cross-referenced to the MULTOS Data Dictionary available in the back of the MULTOS Developers Reference Manual.

Contents

1 Application Load Overview

MULTOS application loading and deleting allows MULTOS card Issuers to create a card that is uniquely tailored to an individual customer or customer profile. Card Issuers can add, modify or delete applications at any time during the card's life, even after the card has been issued. Applications can be downloaded anywhere, using insecure networks without compromising security.

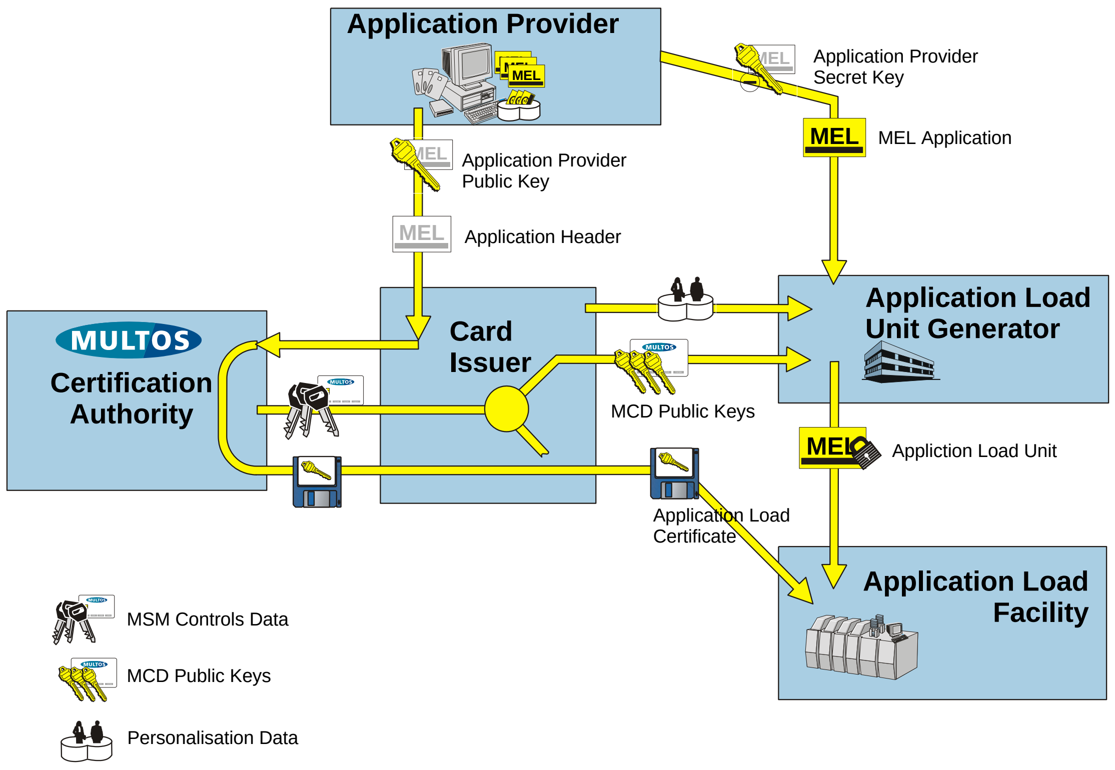
For loading, the application is packaged in a protected file format, called an Application Load Unit (ALU). The ALU mechanism provides a method to protect confidential application data as well as a digital signature to allow the MULTOS card to detect a corrupt ALU during loading. These integral security features mean ALU can be stored on any platform and transmitted over any network.

An application can only be loaded or deleted from a MULTOS card with the permission of the card issuer. This permission takes the form of an Application Load Certificate (ALC) and Application Delete Certificate (ADC). The MULTOS Key Management Authority (KMA) playing the role of the certifying authority is responsible for providing the cryptographic services for the MULTOS scheme. This includes the generation of ALC and ADC for live cards, which can only be requested by the MULTOS card issuer.

A system to load an application on a MULTOS card requires:

- An Application Load Unit
- An ALC. Depending on the issuer's requirements it can be either card specific or can permit application loading for a group of cards
- The Target MULTOS card. This is the Issuer's MULTOS card, which will go to the customer, on which the application is to be loaded

The diagram below shows an overview of the load process.


Application Provider
MEL
Application Provider Secret Key
MEL Application
Application Provider Public Key
Application Header
MULTOS Certification Authority
Card Issuer
Application Load Unit Generator
MCD Public Keys
Appliction Load Unit
Application Load Certificate
Application Load Facility
MSM Controls Data
MCD Public Keys
Personalisation Data

The Application Load Unit, ALU, is generated using the process described within the Guide to Generating Application Load Units [GALU].

Prior to requesting an ALC, the issuer must register their application with the MULTOS KMA. Details of the process and other related administrative tasks can be found on the site http://www.services.stepnexus.com. Once the application is registered the issuer can obtain an ALC from the MULTOS KMA.

1.1 Application Provider

In order to load an application one must be written. Within the MULTOS scheme this role is played by the application provider. They provide:

- The application details required to register the application
- The public key corresponding to the private key used to create an application signature, if any

This means that there is no need to send code or data neither to the MULTOS KMA nor even to the card issuer. The application signature private key may need to be made available when an ALU is generated.

1.2 Application Load Unit

To load an application onto a card, the Application Code must first be formatted into an Application Load Unit. An ALU can provide integrity and confidentiality of application code and data to protect sensitive information.

The process of generating an ALU is described in the Guide to Application Load Units [GALU] and the ALU file format is defined in the MULTOS CA File Interface Formats document [FIF].

1.3 Card Issuer

The card issuer is the principal coordinator for the loading of applications. They must register the application and request the ALC. In addition, ALU production must be sourced as must the loading facility. In practice the ALU production and loading takes place in a card bureau environment.

1.4 MULTOS card

Once the ALU and ALC are obtained a load can proceed, but can only do so when the target MULTOS card is present. The loading mechanism is intrinsic to MULTOS and every load-ready card operates in exactly the same fashion. A series Application Protocol Data Units (APDU) are exchanged between the card and the reader. The MULTOS Developers Reference Manual [MDRM] describes the interface between the card-reader and the MULTOS card. In addition, this document covers those commands with additional explanation as to their use.

1.4.1 Application Operational Mode

A MULTOS card can operate in one of four modes:

- Standard Mode: When the card is powered on no application is selected, but can be using the ISO defined command SELECT FILE.
- Default Mode: In this mode an application is considered selected immediately upon power on. However, other applications can be selected normally using SELECT FILE.
- Shell Mode: When present, a shell application is always selected and no other application can be selected. This means that a shell application must be able to handle any and all commands sent to the card.
- Proprietary Mode: As of MULTOS 4.5.1, implementers are able to define proprietary operational modes to meet specific product requirements.

The application operational mode is part of the information captured when the application is registered. See the MULTOS Developers Guide [MDG] for more details on operational mode.

2 Application Delete Overview

The process of application deletion is significantly simpler than application loading. The process is:

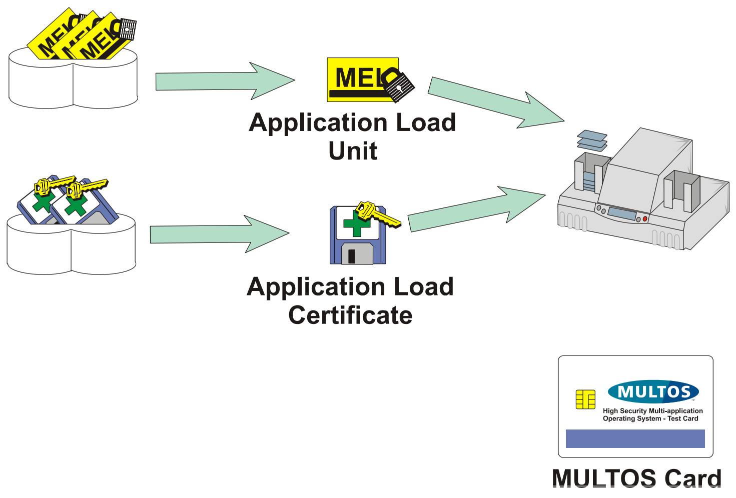
- After an application is registered and an ALC has been requested an issuer can then request the corresponding Application Delete Certificate (ADC)
- When the target MULTOS card is present, transmit the ADC to the chip using a single command APDU
- The chip will verify the certificate validity and, if valid, will remove the application

As of MULTOS 4.5.1, an application is able to **delete itself** without the need of an ADC by calling the *Exit to MULTOS and Restart* primitive.

When the application is deleted its code and memory areas are erased and rendered inaccessible. In addition, the space can be used again to hold an application loaded at a future date.

3 Generalised Application Load Procedure

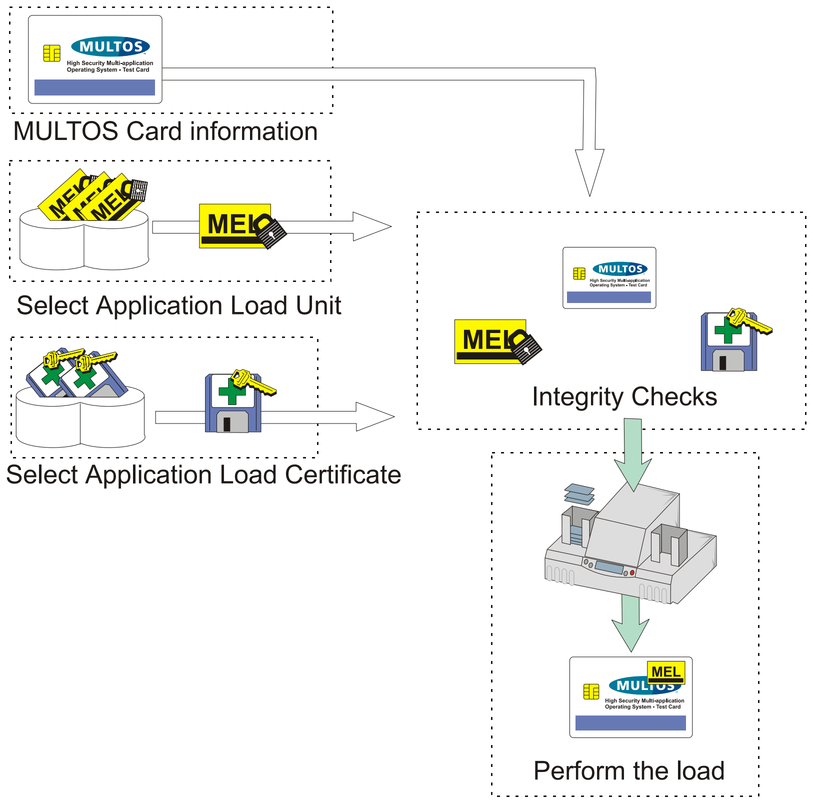
Application loading software always uses the same set of commands, but there may be a difference in scale. At one extreme there is mass loading of applications at a bureau, where thousands of MULTOS cards are being loaded with several applications. At the other extreme one application may be being dynamically loaded to a single MULTOS card. This may be local or over a secure or even insecure network.


MEL
Application Load Unit
Application Load Certificate
MULTOS
High Security Multi-application Operating System - Test Card
MULTOS Card

The generalised load procedure is a set of steps that must be carried out in order to load an application. Some of these steps may not be needed in all cases. However, the purpose of this section is to document all of the steps.

The procedure assumes little prior knowledge on the part of the application loader before the process starts. In practise the application loader must be told what application is to be loaded and given access to the Target MULTOS card. This is the minimum knowledge that any application loader can work with, but in most cases an application loader would be written which had much more information than this.

The application itself is assumed to be held as an AUR file. The AUR file contains all of the ALU related to the application. In practise the application could be held in any valid format.


MULTOS Card information
Select Application Load Unit
Select Application Load Certificate
Integrity Checks
Perform the load

The procedure is divided into five steps. The first four are preparation for the last step, application loading itself. The steps are:

- **Gather Information**: The first step is to gather information from the MULTOS card. This step interrogates the MULTOS card to determine its identity, information about its configuration and some basic information about communicating with it.
- **Acquire ALU**: The Application Load Unit (ALU) must be found and loaded. There are three ways that ALUs can be provided.
- **Acquire ALC**: The Application Load Certificate (ALC) must be found and loaded. In many cases there is a one to one correspondence between an application and an ALC, but this may not necessarily be the case. The ALC must be chosen such that it has the correct permissions to load the application onto the MULTOS card.
- **Integrity Checks**: This step consists of checking the MULTOS card, the ALU and the ALC to determine if the conditions for loading an application have been satisfied. Many of these checks are integrated into the procedure for selecting the ALC and ALU but there are a few extra checks that may be carried out to implement a robust application loader.
- **Application Loading**: The final step is to send the commands to load the application onto the MULTOS card.

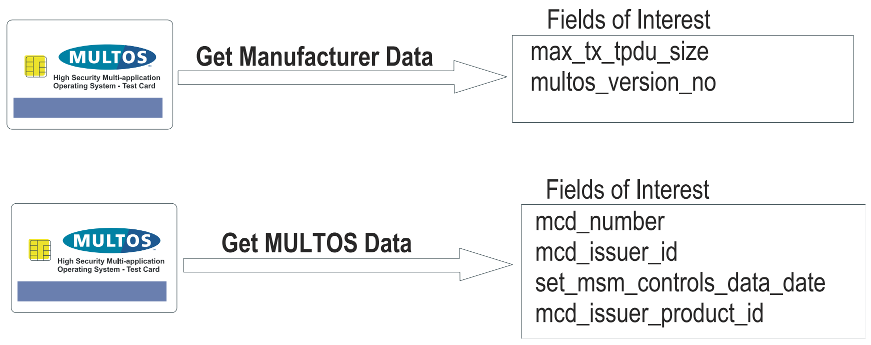
There is no requirement that all of the processing for the above steps has to occur at the same time, or on the same hardware. It is expected that the procedure will be split between machines.

In the case of bureaux, it is expected that much of the processing will be carried out beforehand and the actual loader itself passed pre-processed data to be sent to the MULTOS card. The MULTOS cards and data may be processed, and sorted, such that the loader merely has to send a sequence of commands to the current MULTOS card.

In the case of dynamically loading an application onto a MULTOS card out in the field then the terminal into which the MULTOS card has been placed will perform the actual communications with the MULTOS card whilst the processing may be implemented in a server across a network.

3.1 Gather Information

The purpose of this step is to gather information about the MULTOS card that the application is to be loaded onto. This step is only required if the information about the MULTOS card is to be used later in the process.



The Application Load Certificate may also have to be chosen to match the MULTOS card. If application loading is being carried out en masse it is likely that there is one certificate that is valid for the whole batch of MULTOS cards. In this case the certificate could be explicitly given beforehand and used for all MULTOS cards. For the purposes of this procedure, however, the assumption is that no prior knowledge about the MULTOS card is known and that all information must be gathered as part of the application load.

There are two commands to send to the MULTOS card to get the information needed. They are:

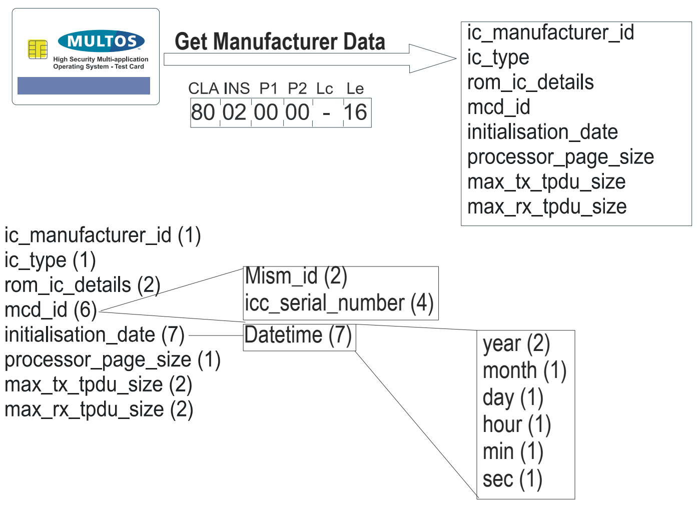
- **Get Manufacturer Data**: This command returns information about the underlying hardware and capabilities of the MULTOS card.
- **Get MULTOS Data**: This command returns information on the MULTOS implementation, identification of the card and the permissions required to load an application onto this card.

3.1.1 Get Manufacturer Data

Get Manufacturer Data is an APDU command that is sent to the MULTOS card.

APDU Command

Field	Value	Description
CLA	‘80’	This is the MULTOS specific command class
INS	‘02’	Indicates the Get Manufacturer Data command
P1	‘00’	Fixed value
P2	‘00’	Fixed value
Lc	Empty	No command data sent
Data	Empty	No command data sent
Le	‘16’	22 bytes of response data expected



The key piece of information found in the response data is the maximum size of the Transport Protocol Data Unit (TPDU) found in the field “max_tx_tpdu_size”. This may prove to be useful when the load commands are generated.

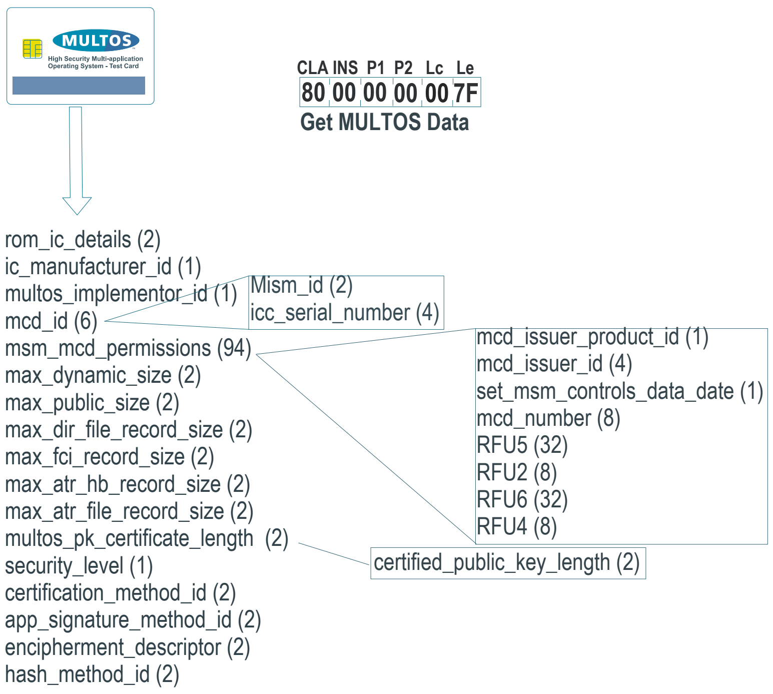
3.1.2 Get MULTOS Data

Get MULTOS data is an APDU command that may be sent to a MULTOS card to obtain the MULTOS Data. The MULTOS Data is defined within the Data Dictionary and most of the information is relevant to application loading.

APDU command

Field	Value	Description
CLA	‘80’	This is the MULTOS specific command class
INS	‘00’	Indicates the Get MULTOS Data command
P1	‘00’	Fixed value
P2	‘00’	Fixed value
Lc	Empty	No command data sent
Data	Empty	No command data sent
Le	‘7F’	127 bytes of response data expected

The response data is illustrated below.


MULTOS
High Security Multi-application Operating System - Test Card
CLA INS P1 P2 Lc Le
80 00 00 00 00 7F
Get MULTOS Data
rom_ic_details (2)
ic_manufacturer_id (1)
multos_implementor_id (1)
mcd_id (6)
msm_mcd_permissions (94)
max_dynamic_size (2)
max_public_size (2)
max_dir_file_record_size (2)
max_fci_record_size (2)
max_atr_hb_record_size (2)
max_atr_file_record_size (2)
multos_pk_certificate_length (2)
security_level (1)
certification_method_id (2)
app_signature_method_id (2)
encipherment_descriptor (2)
hash_method_id (2)
Mism_id (2)
icc_serial_number (4)
mcd_issuer_product_id (1)
mcd_issuer_id (4)
set_msm_controls_data_date (1)
mcd_number (8)
RFU5 (32)
RFU2 (8)
RFU6 (32)
RFU4 (8)
certified_public_key_length (2)

The data returned in response to this command contains data that is useful during the loading process. That information is:

- **Card Permissions**: The field “msm_mcd_permissions” holds data, which may also be used in an ALC. If used in the ALC and there is a mismatch with the card, the MULTOS card will not accept the certificate and the load will fail.
- **Card Number**: The sub-field “mcd_number” holds the number that uniquely identifies a MULTOS card. Both the Application Load Certificate and Application Load Unit may be specific to a particular MULTOS card and it is the mcd_number that is used to identify which ALC or ALU to select.
- **Maximum Directory File Size**: The field “max_dir_file_record_size” indicates maximum size for a directory record entry. This may be used to implement a more robust application loader. If the application's DIR record is larger than this value then the load will fail.
- **Maximum FCI Record Size**: The field “max_fci_record_size” indicates the maximum size for the file control information bytes. It also may be used to implement a more robust application loader. If the application's FCI data is larger than this value then the load will fail.
- **Certification Method ID**: The field “certification_method_id” holds a value that is an identifier that enumerates the internal methods and keys used for certification of public keys. This ID may be compared against a certificate to check that the MULTOS card is using the same method and keys to check the certificate than were used to generate the certificate.
- **Hash Method ID**: The field “hash_method_id” holds a value that is an identifier that enumerates the internal methods and keys used to generate an asymmetrical hash. These hashes are used in the verification of certificates. If the MULTOS card is using a different hash method from the certificate then the verification of the certificate will fail.

3.2 Acquire Application Load Unit

Application Load Units to be loaded usually come in one of three forms:

1. An unprotected ALU that is used for all MULTOS cards
2. A confidential ALU that has been encrypted using the public key of a particular MULTOS card (data preparation has been performed for a known target card or list of cards)
3. A confidential ALU that has been encrypted under a symmetric transport key (data preparation has been performed without any knowledge of the cards that will be used)

Confidential ALUs are usually used where the data segment contains personalised data.

In case (2) it may be required to search for the ALU in a store of generated ALUs. See 3.2.1.

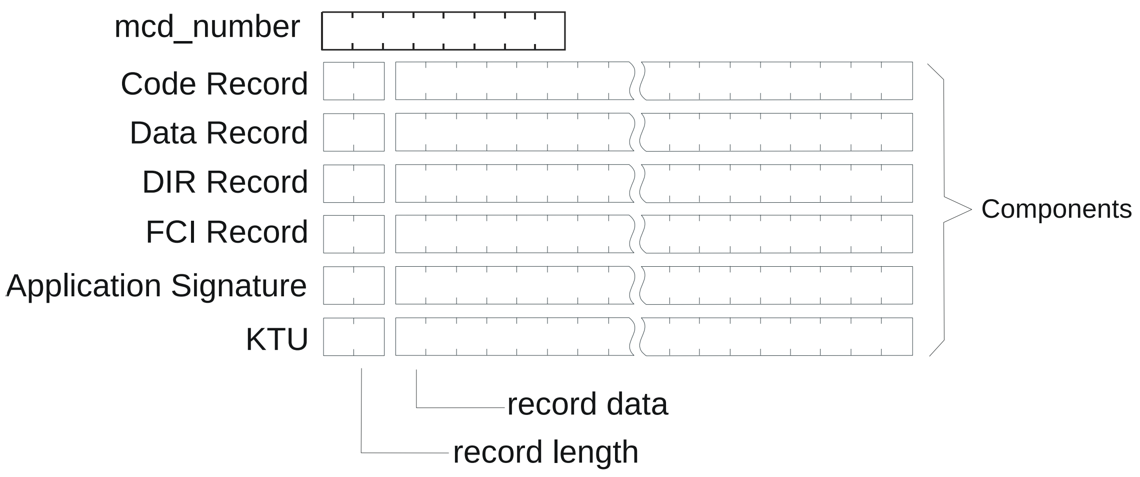
In case (3) it is necessary to use an HSM to re-encrypt the KTU part of the ALU from the transport key to the card’s public key.

The generation of the Application Load Units is described in the "Guide to Generating Application Load Units".

3.2.1 Selecting the Application Load Unit

Each Application Load Unit is the same fixed size and has the MCD Number of the target MULTOS card. If multiple ALU are held in a single file, the file may be searched for a specific MCD Number.

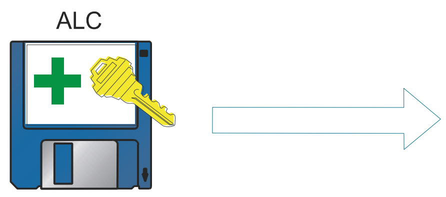
The following diagram shows the structure of an Application Load Unit. The first field of the structure is the mcd_number and this may be used to match an ALU to the MULTOS card. If an Application Load Unit is not specific to a particular MULTOS card then the mcd_number is set to zeros.



The MCD Number is known at this point, having been read from the MULTOS card and the selection of the Application Load Unit is therefore simply a match of the MULTOS card's number with that of an Application Load Unit.

3.3 Acquire Application Load Certificate

The Application Load Certificate is used to provide the security within the loading process. The MULTOS card uses the certificate to verify that the card owner, or Issuer, has granted permission for the application to be loaded onto the MULTOS card. It is not possible to load an application onto a MULTOS card without a valid certificate.



certified_public_key_length (2)
key_type (1)
key_identifier (8)
certification_method_id (2)
hash_method_id (2)
public_key_length (2)
certifying key length (2)
algorithm_id (1)
exponent_length (2)
public_exponent (4)
mcd_issuer_product_ids (32)
mcd_issuer_id (4)
set_msm_controls_data_dates (32)
mcd_number (8)
RFU (18)
application_id (17)
random_seed (8) (MULTOS v4 ONLY)
file_mode_type (1)
code_size (2)
data_size (2)
session_data_size (2)
dir_file_record_size (2)
fci_record_size (2)
app_ATR_type (1)
verify_certificate (1)
verify_ktu (1)
access_list (2) (MULTOS v4 ONLY)
application_code_hash_length (1) (MULTOS v4 ONLY)
ciphertext (Variable)

The Application Load Certificate contains information used to identify the application as well as the card. That information is:

- **Application Identifier (AID)**: The field "application_id" has a value that indicates the card unique AID. Each certificate is specific to a single application.
- **Application Code Hash**: The field "application_code_hash" contains a hash digest of the application code section as it will be loaded onto the card.
- **Issuer ID**: Each certificate is specific to a single issuer. The field "mcd_issuer_id" holds the domain unique Issuer ID. If the ALC and card Issuer ID do not match, the load will fail.
- **Product ID**: Each certificate has a list of permitted ID held in the bitmapped field "mcd_issuer_product_ids". A MULTOS card has a product ID and if this is not included in the certificate's list , the load will fail.
- **Card Enablement Date**: Each certificate has the bitmapped field "set_msm_controls_data_dates", which indicates which date range is permitted. MULTOS cards have an enablement date value in the field "set_msm_controls_data_date"(set by the MULTOS KMA when the card is enabled) and if this date is not in the list of dates in the certificate, the load will fail.

- **MCD Number**: This is an optional field within the certificate. A certificate may be made specific to a single MULTOS card by setting the mcd_number within the certificate to the mcd_number of the target card. If the certificate's mcd_number is not zero, then it may only be used to load the application onto the specified card.
- **File Mode Type**: This one byte bitmapped field describes the application.

b7	b6	b5	b4	b3	b2	b1	b0	Meaning (MULTOS 4.2.1 and earlier)
0	0	0	0	0	0	0	0	Standard application (0x00)
0	1	0	1	1	0	1	0	Shell application (0x5A)
1	0	1	0	0	1	0	1	Default application (0xA5)
b7	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**	**Meaning (MULTOS 4.3 and later)**
			0					Single FCI application
			1					Dual FCI application
				0	0			Static memory size given in bytes
				0	1			Static memory size given in 255-byte blocks
				1	0			Static memory size given in bytes
				1	1			Static memory size given in bytes
						0	0	Standard application
						0	1	Default application
						1	0	Shell application
						1	1	Proprietary application type
		0						Standard application loading
		1						Controls the loading of the application in some proprietary implementation-specific way.
0	0							Fixed

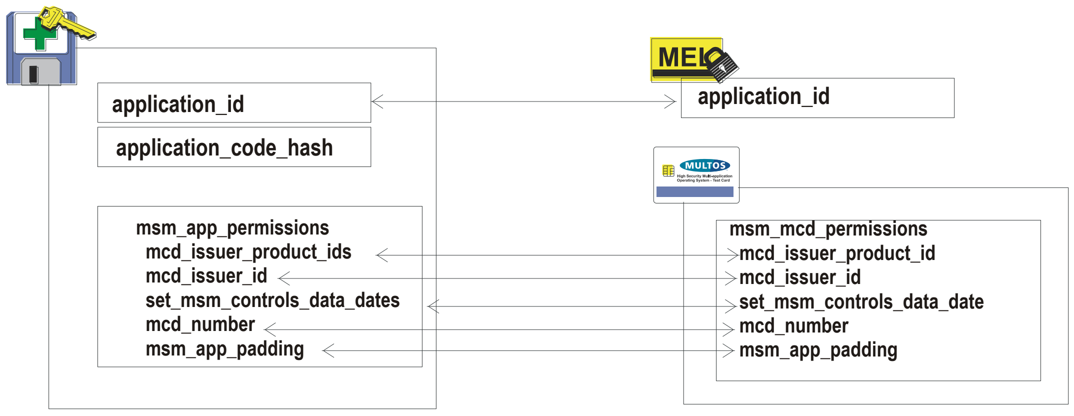
- **Access List:** The bits in this two byte value define the application's permissions and have the following meaning (set to 1 when the application has that permission).
  - bit0 - Strong Cryptographic functions
  - bit1 - Contact IFD interface
  - bit2 - Contactless PCD interface
  - bit3 - GSM Authenticate
  - bit4 - Card Block
  - bit5 - Card Unblock
  - bit6 - Retain session data
  - bit7 - Maintain selection
  - bit8 – PIN Access Level } 00 = Application, 01 = Global / Basic
  - bit9 – PIN Access Level } 10 = Global / Write, 11 = Global / Full
  - bit10 – Process Events permission
  - bit11 – Card Manager application
  - bit12 – Allow access to peripheral devices
  - bits13 to 15 – RFU

- **app_ATR_type:** A 1 byte value that indicates whether the application wishes to contribute to the historical bytes of the primary or alternative ATR and the ATS
  - None = 0x00,
  - Primary ATR = 0x41,
  - Secondary ATR = 0x42,
  - Both ATRs = 0x43,
  - Primary ATS = 0x44,

- Primary ATR and ATS = 0x45,
- Secondary ATR and ATS = 0x46,
- Both ATRs and ATSs = 0x47

3.3.1 Selecting an Application Load Certificate

The selection of an Application Load Certificate will depend largely on how the certificates are stored and organised. As indicated in the previous section there are a series of data elements that must match.



Please refer to the following sections for more specific details on some of the above fields. These are the more complex fields to match.

3.3.2 Note on the Application Code Hash

The Application Code Hash is a digest of the application code as it will be loaded onto the card. This means that if any part of the code is encrypted, then the hash must be calculated over the encrypted code.

When an application is registered with the MULTOS KMA the Application Code Hash must be supplied. When certificates are requested to load the application onto a MULTOS card this hash is included within the certificates and enables the MULTOS card to verify that the specific application has been loaded and that it has not been substituted or corrupted.

3.4 Integrity Checks

At this stage of the application loading procedure the Application Load Unit and Application Load Certificate have been selected, all of the data required for the loading has been obtained and the load commands may be generated and sent. This section describes some checks that may be carried out before the actual load commands are sent. These checks are for robustness only and may be omitted in a specific implementation.

These checks have been included within the generalised procedure for completeness; they represent a complete list of all checks that could be carried out by an application loader. The aim is that if all of the conditions are met then the application load will be successful.

An alternative approach would be to simply let MULTOS perform the checks and handle errors returned by MULTOS, but if errors occur during the Create MEL Application stage then a retry counter is incremented. When the retry counter exceeds the maximum allowed for that operation (See the MULTOS Implementation Report [MIR] for more details) the cards will no longer be able to have applications loaded (unless the retry delay mechanism is implemented. See 4.1 for details).

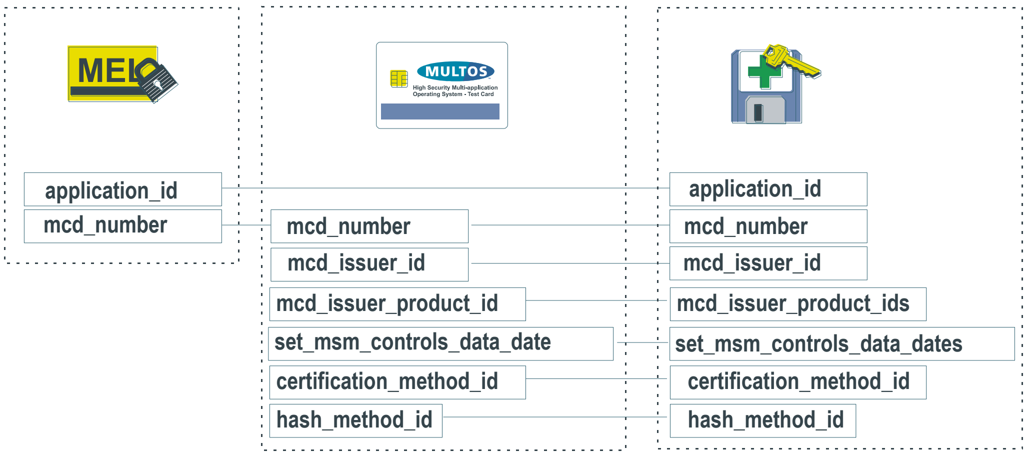
This is a complete list of all checks that are carried out by the MULTOS card during the application load process. Checks that are impractical to perform as part of the application loader have been included for completeness only.

The checks fall into three categories:

- **ALC, ALU and MCD cross checks**: These verify that the Application Load Certificate (ALC) corresponds to the Application Load Unit (ALU) and has the required permissions (MCD) to load the application onto the MULTOS card.
- **ALU Sizes**: These checks verify that the application will fit onto the MULTOS card, and that all of the required components of the ALU are present.
- **Cryptographic**: This is a list of the checks that will be carried out by MULTOS and represents a list of mismatches that will cause the load to fail.

3.4.1 ALC, ALU and MCD Cross Checks

These checks will have been performed already as part of the Application Load Unit and Application Load Certificate selection procedure. If the load process has been followed then all of these conditions will have been met.



Application Identifier (AID)

The application_id given in the Application Load Unit and Application Load Certificate should match to ensure that the correct MULTOS Application is being loaded. This sounds obvious, but the manner in which an application is identified should be understood. Both MULTOS and the MULTOS KMA use two fields to identity an application: the application_id and the Application Code Hash.

The application_id is the normal method to identify an application. Most applications will have a unique application_id and in most practical cases just this may be used to match applications. In terms of security, however, the application_id is not very useful. In the same way that any application on a PC could be given any filename so can any MULTOS Application could be given any application_id. MULTOS solves this problem by using a hash of the code to identify an application. If the code changes then the Application Code Hash changes and it appears to be a different application.

The application_id structure defined by MULTOS is seventeen bytes long. The first byte contains the length of the AID, followed by the AID itself and then padding bytes of 0xFF to the seventeen bytes length. If the padding is different then this will count as a mismatched AID and the load will fail. It is advised that the application_id be taken from the Application Load Certificate.

MCD Number
The mcd_number in the Application Load Unit (ALU) and the Application Load Certificate (ALC) must match the mcd_number of the MULTOS card. The MULTOS card will report it's mcd_number as part of the MULTOS Data and this number is also stored within the MULTOS card's public key.

The mcd_number is used with Application Load Certificate to limit the use of the certificate to a specific MULTOS card. Any MULTOS card whose mcd_number does not match the Application Load Certificates mcd_number will reject the Application Load Certificate with one exception.

If the mcd_number of the ALC is all zeroes (00 00 00 00 00 00 00 00), then the certificate is not specific to one MULTOS card.

Issuer ID
The mcd_issuer_id in the Application Load Certificate must match the mcd_issuer_id returned by the MULTOS card.

The mcd_issuer_id of a MULTOS card is used by the MULTOS card to identify its Issuer, or owner. The purpose of using Application Load Certificate is to enable Issuers to control what applications are loaded, or deleted, from their cards. A MULTOS card will only accept an Application Load Certificate that bears the same mcd_issuer_id. The MULTOS CA will only issue Application Load Certificates at the request, or authorisation, of an issuer and only with the appropriate mcd_issuer_id stored within them.

It is expected that the mcd_issuer_id will match in most cases since the MULTOS KMA will only issue certificates to a card issuer with their mcd_issuer_id within them. For the mcd_issuer_id to be mismatched between a MULTOS card and a Certificate implies an attempt to load an application onto a MULTOS card belonging to one issuer using a certificate from another issuer. In these cases it is likely that the certificates have been mixed up.

Product ID
The mcd_issuer_product_id of the MULTOS card must be in the list of mcd_issuer_product_ids in the Application Load Certificate.

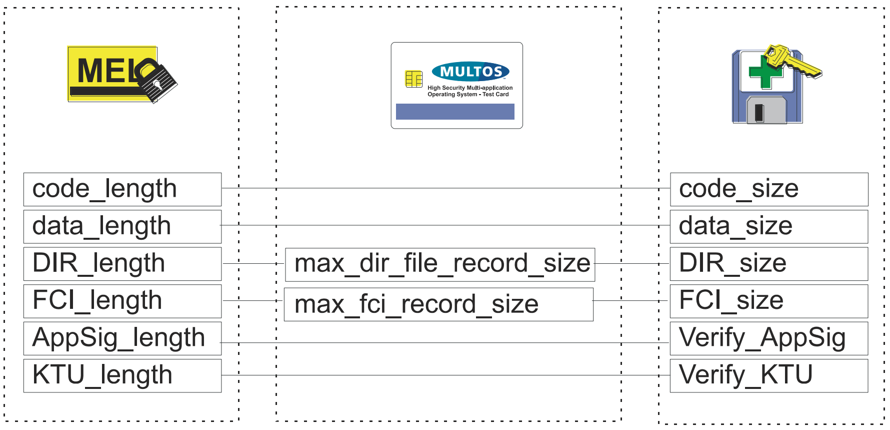
Enablement Date
The set_msm_controls_data_date of the MULTOS card must be in the list of set_msm_controls_data_dates in the Application Load Certificate.

Certification Method
This identifies the algorithm and key used in the certificate generation. If the certificate and card ids do not match then the card will not be able to decipher the ALC and the load will fail.

Hash Method
This identifies the algorithm and modulus used in MULTOS asymmetric hashing. If the certificate and card ids do not match then the card will not generate the same hash as used for the ALC and the load will fail.

3.4.2 ALU Sizes

These checks involve the examination of the Application Load Unit itself to ensure that the sizes of the components will fit onto the MULTOS card. The memory on a MULTOS card is allocated dynamically as applications are loaded and deleted.



There are a number of size checks that the chip performs.

Application Footprint
The footprint of the application refers to the amount of non-volatile memory that the application requires when loaded. The application may not occupy more space than is available on the chip either to store the application or to use even temporarily for the load.

Calculating the application footprint is slightly more complex than just the verifying the size of the Application Unit itself. There is additional overhead required by the MULTOS operating system, padding according to the underlying hardware page size and differences in how implementations implement memory management.

Firstly, the nature of EEPROM requires that memory is allocated in complete pages, typically of 32 bytes. The size of code and data must be rounded up to the nearest page to calculate the actual memory that will be required.

Secondly, each MULTOS Application has an overhead that is required by MULTOS to maintain control information for the application. The size of this overhead is implementation specific.

Thirdly, if an Application Signature or Key Transformation Unit is included then these must be held in non-volatile memory until the load is complete. Although these are deleted once the load is complete the temporary memory required during the loading process limits the maximum size of an application.

Code Component
The size of the Code Component in the Application Load Unit must not exceed the size specified within the Application Load Certificate. The actual size of code may be less than specified in the Application Load Certificate, although this may be wasteful and potentially confusing.

Note that the size of the application's code is always taken to be that specified in the Application Load Certificate. For example, when MULTOS calculates the Application Code Hash then the size in the ALC will be used. If the ALC size is larger than the actual code size then padding bytes of '00' must be appended to the application's code such that the sizes match.

Data Component
The size of the Data Component in the Application Load Unit must not exceed* the size specified within the Application Load Certificate. The size of an application's data is always taken to be that specified in the Application Load Unit, when generating Application Signatures it is the size specified in the Application Load Unit that must be used.

An application may reduce or increase the amount of Static memory available to it using the primitive *Update Static Size* within the bounds of the size given in the Application Load Certificate and the amount of memory physically remaining.

DIR Record Component
The DIR Record Component in the Application Load Unit must not exceed* the size specified within the Application Load Certificate or that indicated by the MULTOS card in the Manufacturer Data. The size indicated within the Manufacturer Data is an absolute maximum that cannot be exceeded. The maximum size is implementation dependent and some implementations do not place any specific limit on DIR Record size. No check is made to ensure that the DIR entry is valid.

Note that this size is also used when generating Application Signatures.

FCI Component
The FCI Component in the Application Load Unit must not exceed* the size specified within the Application Load Certificate or that indicated by the MULTOS card in the Manufacturer Data.
The size indicated within the Manufacturer Data is an absolute maximum that cannot be exceeded. The maximum size is implementation dependent and some implementations do not place any specific limit on FCI size.

The size specified within the Application Load Certificate is the size that is taken to be the actual length of the File Control Information. For example, if the Application Load Certificate specifies that there are 20 bytes of FCI then 20 bytes of FCI will be returned when the application is selected. This size is also used when generating Application Signatures.

MULTOS does not verify that the File Control Information Component contains a valid entry.

***Note:** Prior to MULTOS 4.3.2, the data, DIR and FCI sizes specified in the ALC had to match the sizes in the ALU.

Application Signature
The Application Load Certificate contains a field that specifies if an Application Signature is to be verified. If the Application Signature verification is enabled then the MULTOS card will use the Application Signature to check the integrity of the application. If no Application Signature is present then this check, and the load, will fail.

The Application Signature is generated using the Application Provider Private Key. The public portion of this key is contained within the Application Load Certificate. Therefore the private key used to generate the Application Signature must match the public key in the certificate.

Note: If an Application Signature is loaded, MULTOS does not report an error if Application Signature verification is not enabled in the Application Load Certificate. Since this can only apply to plaintext Application Units the recommendation is for the application loader to detect this and throw an error. Otherwise the load process appears to be protected with an Application Signature, when in fact there is no protection - this is confusing and potentially dangerous.

Please refer to the "Guide to Generating Application Load Units" for more information on the generation of Application Signatures.

Key Transformation Unit

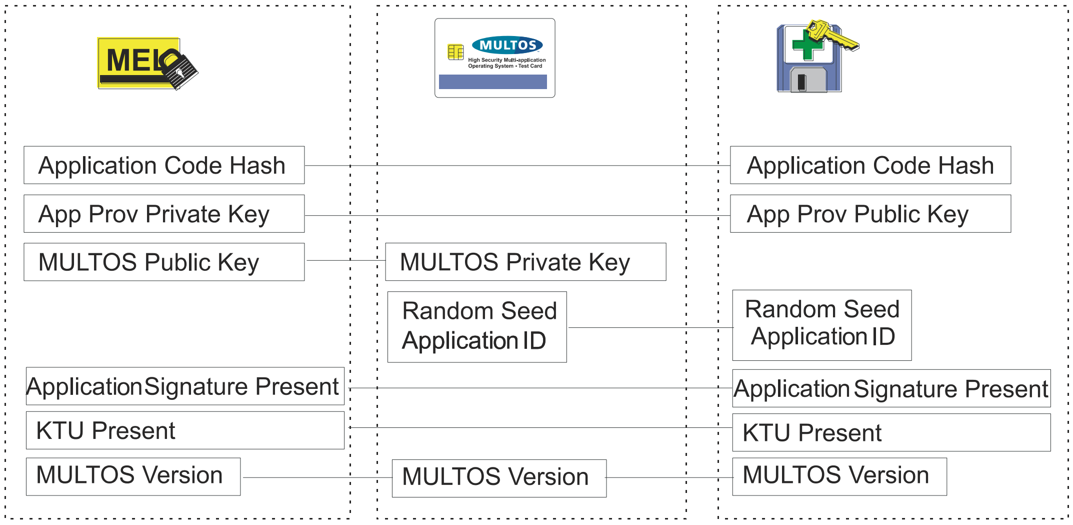
The Application Load Certificate contains a field that specifies if a Key Transformation Unit is to be used during the loading. The Key Transformation Unit contains information used by MULTOS to decipher an enciphered Application Unit. If a Key Transformation Unit is indicated in the Application Load Certificate, but not included within the Application Load Unit then MULTOS will still attempt to decipher the Application Unit and either fail due to a corrupted Key Transformation Unit, or corrupt the Application Unit.

The Key Transformation Unit contains information on how to decipher the Application Unit. The Key Transformation Unit itself is enciphered using the Public Key of the MULTOS card. This key is unique to the MULTOS card and only the specific MULTOS card will have the corresponding private key to correctly decipher the Key Transformation Unit. If a Key Transformation Unit is present then the Application Load Unit must be loaded onto the MULTOS card whose public key was used in the generation of the Key Transformation Unit.

Please refer to the "Guide to Generating Application Load Units" for more information on the generation of a Key Transformation Unit and the enciphering of an Application Unit.

3.4.3 Cryptographic

The cryptographic checks relate to those that, either directly or indirectly, involve cryptography and by their very nature are the most difficult to verify by an application loader. In fact, the only the chip is equipped to perform all the verifications required.



Application Provider Key
The Application Provider's public key is contained within the Application Load Certificate and is used by the MULTOS card to authenticate the application. If an Application Signature is not included within the Application Load Unit then the AP Key is not used during the load. If an Application Signature is present then the public key corresponding to the private key used to generate the Application Signature must be in the Application Load Certificate. If the public key does not match then the MULTOS card will fail to authenticate the application.

Note that the presence of an Application Signature requires that the certificate contains the Verify Application Signature flag. If this flag is not set then the MULTOS card will not check the Application Signature even if one is loaded.

MCD Public Key
The Public Key Certificate of the MULTOS card is returned by the MULTOS card in response to the Open MEL Application command. An ALU generator should first verify the validity of the public key certificate using the appropriate MULTOS KMA supplied Transport Key Certifying Key (TKCK). Once validated the public key may be extracted and is then used to encipher the Key Transformation Unit (KTU). If the KTU has been created prior to loading, there is no way to check the KTU and the application loader must match the ALU to the card using the MCD Number.

If the wrong public key is used to encipher the Key Transformation Unit, the chip will be unable to verify the KTU and the load will fail.

Random Seed
The Random Seed is an eight-byte value that allows certificate to be used only once per MULTOS card. If the Random Seed is set to a non-zero value then the Application ID and Random Seed values are stored in an Application History list on the card. The card checks the Application History list during the load process and will fail if the same AID and Random Seed appears in the list.

Note that an application loaded using a certificate that has a non-zero Random Seed can only be deleted with a delete certificate with the same Random Seed. When deleting an application where the Random Seed value is in use then a method of matching the delete certificate must be built into the system.

Application Signature

An application loader may check the Application Signature Method ID to verify that the MULTOS card will interpret the Application Signature in the expected manner. There is no corresponding field within the ALU to explicitly compare Application Signature Method ID.

MULTOS Version

The MULTOS Version should be checked to determine the format that the Open MEL Application command should take. There are additional fields for Version 4 of the MULTOS Specifications compared to Version 3. The MULTOS Version also affects the format of the certificate.

In practise a check on the version of MULTOS need not be done since the keys used to generate the certificate change between implementations and matching the correct keys will always match the correct MULTOS Version.

Hash Method ID

The Hash Method ID is used to enumerate the various methods that can be used to generate a Hash Digest. Hash Digests are used in the verification of any certificates. The MULTOS KMA is aware of what the correct Hash Method ID is for a specific MULTOS card and there is no danger of the certificate being incompatible for this reason. However, an Application Signature, if present, must be generated using the method specified by the Hash Method ID of the MULTOS card.

The generation of the Application Signature is documented within a "Guide to Generating Application Load Units" and there is little danger of mismatched Hash Method ID providing the procedure for generating the signature is followed. The Hash Method ID for a MULTOS card is contained within the MULTOS cards public key certificate and therefore an Application provider can easily determine what method to use for hash generation.

Certification Method ID

The Certification Method ID of the certificate and MULTOS card must match exactly. The Certification Method ID of the certificate indicates how the certificate was generated and gives both the format and key used during certification. The certification method ID of the MULTOS card gives the method and key that the MULTOS card will use to authenticate the certificate. If these two values are different then the authentication will fail.

Certificates are specific to a MULTOS implementation and when an Application Load Certificate is requested, the implementation of MULTOS with which they are to be used must be given.

4 MULTOS Application Load and Delete Process

This section provides an overview of the command sequences used to load or delete an application, and briefly describes the purpose of each command. These commands are always available, irrespective of the file (DF or EF) currently selected. A full description of these commands can be found in the MULTOS Developers Reference Manual [MDRM].

For the purposes of this section it is assumed that the commands are sent to the MULTOS card as they are generated; although this need not be the case. For example, If the application is being loaded over a network, specifically from a server to a terminal, then the command APDU may be constructed by one process stored in a server and then transmitted to a terminal. This simplifies the terminal application considerably.

4.1 Load Sequence Overview

The commands listed below are used to load a MULTOS application. A description of each of the load commands can be found in the sections that follow.

Command	Order
Open MEL Application	First
Load Code	Any Order
Load Data	Any Order
Load DIR File Record (Directory Record)	Any Order
Load FCI Record (File Control Information Record)	Any Order
Load Application Signature	Any Order
Load KTU Ciphertext (Key Transformation Unit Ciphertext)	Any Order
Create MEL Application	Last

The Open MEL Application command is the first command in the load sequence. The "load commands" may then be sent in any order. A separate MULTOS command is used for Code, Data, DIR and FCI since each data type is stored in separate areas of EEPROM. Separate commands are also used to load the application signature and KTU ciphertext but these are removed from memory once the load sequence has been completed.

The Create MEL Application command is the last command in the load sequence. It creates (enables) an application that has been previously opened and loaded.

The load procedure will be aborted if:

- The card is reset (i.e. power cycled)
- Another Open MEL Application command is sent part way through the load procedure
- A Delete MEL Application command is sent to the MULTOS card by mistake

If the load procedure is aborted, all code and data loaded is lost and memory reserved by the Open MEL Application command is made available to other applications.

If a MULTOS Application fails to load then the MULTOS decrements a retry counter, which can never be incremented. When this counter reaches zero the MULTOS card will act in one of two ways:

a) refuse to accept any more load commands or
b) introduce a delay between retries

The Integrity Checks carried out in the previous step are an important part of application loading.

A brief description of the commands used to load an application is provided below. Consultation of the MULTOS Developers Reference Manual will be necessary when using this document for further details of the command formats and specific error conditions.

4.1.1 Open MEL Application

This command checks whether there is sufficient EEPROM and RAM available to load the ALU onto the MULTOS card. The command data must be consistent with the data contained in the ALC; otherwise MULTOS will reject the load.

The Open MEL Application command may return success, an error or a warning, as defined in the MULTOS Developers Reference Manual. A warning indicates that although the Open MEL Application command has not failed, the Create MEL Application command will fail later in the load sequence.

APDU command

Field	Value	Description
CLA	'BE'	This is the MULTOS specific command class for all load / delete commands
INS	'12'	Indicates the Open MEL Application command
P1	'00'	Fixed value
P2	'00'	Fixed value
Lc	See comment	0x89, for no Application Code Hash 0x9D for a SHA-1 Application Code Hash
Data	See comment	94-byte msm_app_permissions + 17-byte application_id + 8-byte random_seed + 1-byte file_mode_type + 2-byte code_size + 2-byte data_size + 2-byte session_data_size + 2-byte application_signature_length + 2-byte ktu_length + 2-byte dir_file_record_size + 2-byte fci_record_size + 2-byte access_list + 1-byte application_code_hash_length + (*n*-byte application_code_hash)
Le	'00'	Value indicates that all bytes of the card public key certificate should be returned

All of the data is read from the Application Load Certificate except for the lengths of the Application Signature and Key Transformation Unit; the Application Load Certificate does not hold these lengths. The ALC has flags that indicate the presence of these two items in the ALU.

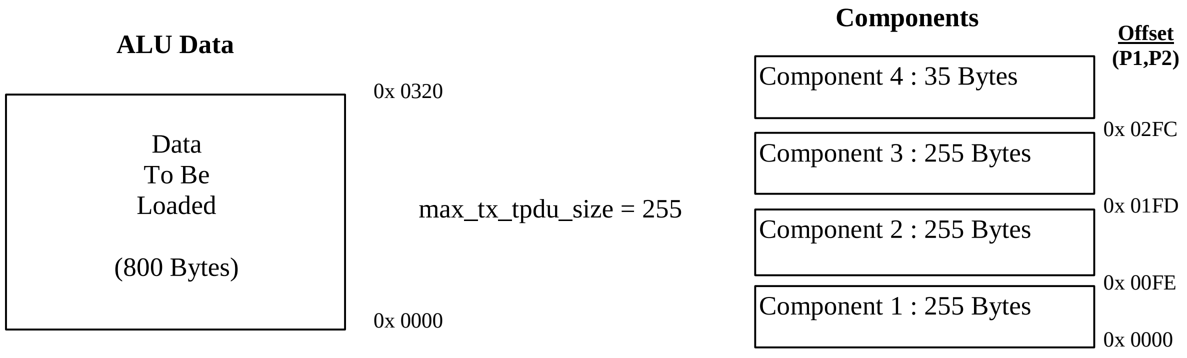
The application_code_hash may be read directly from the Application Load Certificate and may be zero or twenty bytes long depending upon if the hash has been included and which hashing algorithm was used. A zero length Application Code Hash is only allowable for Application Developer Cards. That is, a zero hash length is not allowable for a Live MULTOS card.

The response to this command should be the MULTOS cards public key certificate. If an error response is returned then the loading process should be aborted.

4.1.2 Load Commands General

These commands are used to load the ALU contents onto a MULTOS card. This must include the application code and may optionally include application data, DIR entry, FCI data, application signature and/ or KTU Ciphertext. Application code and data is loaded into a separate application specific segregated EEPROM area. The MDRM defines separate MULTOS commands to load each data type.

For a given MULTOS card and ISO transport protocol being used, a restriction is placed on the maximum size Transportation Protocol Data Unit (TPDU) that can be used, i.e. the amount of data that can be sent in one block. The document [7816-4] has more information on TPDU. To accommodate TPDU block size restrictions, the load commands allow data to be loaded in "components". Each component is given an offset, in bytes, from the first byte of data (zero offset). For example, if the code size is 800 bytes and the maximum TPDU size is 255 bytes then 4 is the minimum number of code components (and hence Load Code commands) required is 4.



The maximum TPDU length the card supports can be determined using the Get Manufacturer Data command. By splitting the data into maximum size components, the number of IFD-MULTOS commands (and hence data traffic between the IFD and card) is minimised. This may be particularly important if down-loading an application over a tariff network.

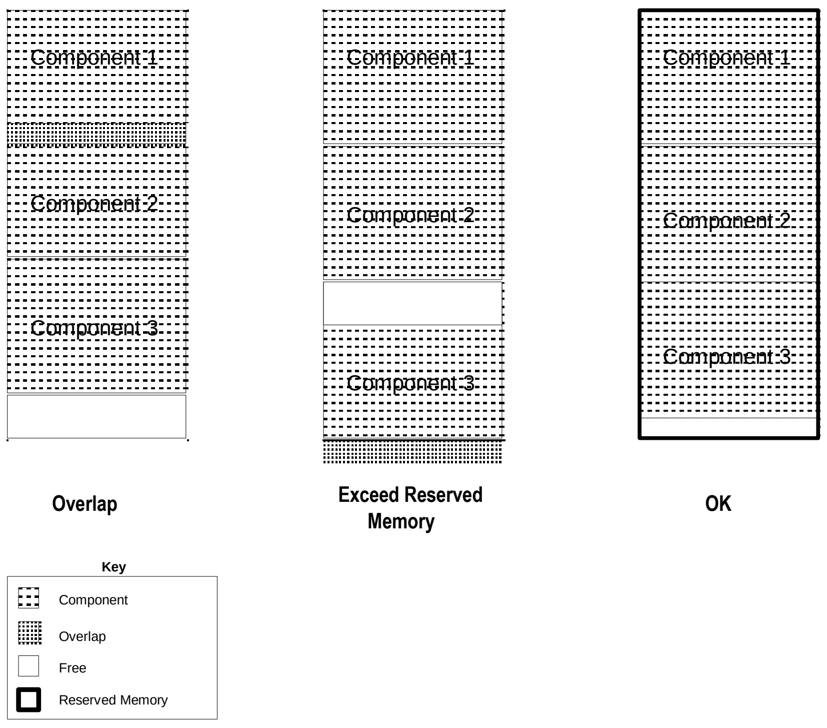
The standard ISO command header parameters P1 and P2 code the offset in bytes, where P1 is the most significant byte.

If a load command attempts to load a component outside the memory area reserved for it (for example, if the offset has been calculated incorrectly) then the MULTOS card will return a "Invalid Offset" error message. See the [MDRM] for further information on error handling.

MULTOS ensures that the components are only written to the memory areas reserved by the Open MEL Application command, but it does not check that the components do not overlap or that all components are loaded. Such errors will be identified when MULTOS checks the Application Signature.

Note that an application does not have to completely fill the code, data, DIR and FCI space reserved for it. Any unused space cannot be used by other applications. This means an application provider can reserve, for example, spare data space to allow an application's private data files to expand during an application's lifetime.

If an application attempts to extend its data files beyond the data space reserved for it, the card will detect this and an "Abend" (abnormal end) will occur. Application execution is terminated. After an "Abend", the MULTOS card becomes non-responsive and this should be detected by the IFD protocol in use (e.g. T0 Work Waiting Time). When the IFD resets the MULTOS card, control is returned to MULTOS.


Component 1
Component 2
Component 3
Overlap
Component 1
Component 2
Component 3
Exceed Reserved Memory
Component 1
Component 2
Component 3
OK
Key
Component
Overlap
Free
Reserved Memory

4.1.3 Load Code

Repeated use of the Load Code command allows an IFD to load application code into a MULTOS card's EEPROM. In the example above, the Load Code command APDU for the third block of data (bytes 0x01FE to 0x02FC inc.) being sent to the card would be:

APDU command

Field	Value	Description
CLA	'BE'	This is the MULTOS specific command class for all load / delete commands
INS	'24'	Indicates this is a Load Code command.
P1	'01'	The most significant byte of the offset.
P2	'FE'	The least significant byte of the offset.
Lc field	'FF'	The length of the command data field in bytes (255).
Data field	Component 3	255 bytes of application code (bytes 511 .. 765 inclusive)
Le field	Empty	No response data expected.

A response of '9000' should be returned if the command was successful. If any other response is returned then the load is likely to fail and should be aborted. See [MDRM] for further details.

4.1.4 Load Data

Repeated use of the Load Data command allows an IFD to load application static data into a MULTOS card’s EEPROM. See [MDRM] for further details. For example, the Load Data command APDU to load the first 10 bytes of application data is:

APDU command

Field	Value	Description
CLA	‘BE’	This is the MULTOS specific command class for all load / delete commands
INS	‘26’	Indicates this is a Load Data command.
P1	‘00’	The most significant byte of the offset.
P2	‘00’	The least significant byte of the offset.
Lc field	‘0A’	The length of the command data field in bytes (10).
Data field	-	10 bytes of application data (bytes 0 .. 9 inclusive)
Le field	Empty	No response data expected.

A response of ‘9000’ should be returned if the command was successful. If any other response is returned then the load is likely to fail and should be aborted.

4.1.5 Load DIR File Record

The document [7816-4] defines the DIR file record format. The DIR file is read by the IFD to identify the applications currently loaded on the MULTOS card. Information in the DIR file can be subsequently used to select the application using the Select File command. The maximum DIR file record size the MULTOS card supports can be determined using the Get Manufacturer Data command.

Repeated use of the Load DIR File Record command allows an IFD to load a DIR file record into a MULTOS card's EEPROM. For example, the Load DIR File Record command APDU to load the first 99 bytes of the DIR record is:

APDU command

Field	Value	Description
CLA	'BE'	This is the MULTOS specific command class for all load / delete commands
INS	'20'	Indicates this is a Load DIR File Record command.
P1	'00'	The most significant byte of the offset.
P2	'00'	The least significant byte of the offset.
Lc field	'63'	The length of the command data field in bytes (99).
Data field	-	99 bytes of DIR record (bytes 0 .. 98 inclusive)
Le field	Empty	No response data expected.

DIR is optional. So if the size specified in the ALU is zero then this command is not needed. A response of '9000' should be returned if the command was successful. If any other response is returned then the load is likely to fail and should be aborted.

4.1.6 Load FCI Record

File Control Information (FCI) details the file management and control parameters for an ISO file and is available in response to a Select File command. See the document [7816-4] for further details.

Repeated use of the Load FCI Record command allows an IFD to load an FCI into a MULTOS card's EEPROM. For example, the Load FCI Record command APDU to load the first 16 bytes of the FCI is:

APDU command

Field	Value	Description
CLA	'BE'	This is the MULTOS specific command class for all load / delete commands
INS	'22'	Indicates this is a Load FCI Record command.
P1	'00'	The most significant byte of the offset.
P2	'00'	The least significant byte of the offset.
Lc field	'10'	The length of the command data field in bytes (16).
Data field	-	16 bytes of the FCI record (bytes 0 .. 15 inclusive)
Le field	Empty	No response data expected.

FCI is optional. So if the size specified in the ALU is zero then this command is not needed. A response of '9000' should be returned if the command was successful. If any other response is returned then the load is likely to fail and should be aborted.

4.1.7 Load Application Signature

The application signature allows a MULTOS card to verify the integrity of an application_unit once loaded onto the card. The use of this mechanism is optional and it is up to the Application Provider to indicate their preference when the application is registered with the MULTOS KMA. However, it is recommended that ALU are protected in this way to allow the MULTOS card to detect corrupt ALU (due to transmission errors, deliberate tampering etc.).

Repeated use of the Load Application Signature command loads a signature of the application_unit (code, data, DIR file record and FCI record) onto a MULTOS card. The application_unit signature is signed with an Application Signature Private Key. This key may belong to either the Application Provider or the MULTOS card Issuer, as appropriate. The Application Signature Public Key is required to verify the signature is provided in the ALC. The ALC is loaded onto the card using the Create MEL Application command.

The Load Application Signature command APDU syntax is:

APDU command

Field	Value	Description
CLA	'BE'	This is the MULTOS specific command class for all load / delete commands
INS	'28'	Indicates this is a Load Application Signature command.
P1	'00'	The most significant byte of the offset.
P2	'00'	The least significant byte of the offset.
Lc field	Var.	The length of the command data field in bytes, which is equal to the modulus size
Data field	-	n bytes of the Application Signature
Le field	Empty	No response data expected.

The application signature is optional. So if the size specified in the ALU is zero then this command is not needed. A response of '9000' should be returned if the command was successful. If any other response is returned then the load is likely to fail and should be aborted.

4.1.8 Load KTU (Key Transformation Unit) Ciphertext

If an Application Provider wishes to keep parts of its application confidential, it can be encrypted to protect it. The encrypted areas can only be decrypted once safely loaded onto the correct MULTOS card.

The KTU Ciphertext contains information the MULTOS card needs to decrypt the code and data. The KTU Ciphertext itself is encrypted with the MULTOS card's public key. Thus the KTU Ciphertext (and hence ALU) is unique to a MULTOS card and can only be decrypted by the card having the corresponding private key.

Repeated use of the Load KTU Ciphertext command loads the KTU Ciphertext onto a MULTOS card. KTUs are optional, and it is up to the Application Provider to indicate their preference, when the Issuer registers the application with the MULTOS CA.

The Load KTU Ciphertext command APDU syntax is:

APDU command

Field	Value	Description
CLA	'BE'	This is the MULTOS specific command class for all load / delete commands
INS	'2A'	Indicates this is a Load KTU Ciphertext command.
P1	'00'	The most significant byte of the offset.
P2	'00'	The least significant byte of the offset.
Lc field	'48'	The length of the command data field in bytes (72).
Data field	-	72 bytes of the KTU (bytes 0 .. 71 inclusive)
Le field	Empty	No response data expected.

The KTU is optional. So if the size specified in the ALU is zero then this command is not needed. A response of '9000' should be returned if the command was successful. If any other response is returned then the load is likely to fail and should be aborted.

4.1.9 Create MEL Application

The Create MEL Application command loads an ALC onto a MULTOS card. An ALC contains information about the application when it was registered with the MULTOS KMA.

The MULTOS card:

- Verifies the ALC is a valid load certificate.
- Confirms the ALC application load permissions (msm_app_permissions) are compatible with the card's permissions (msm_mcd_permissions). When a MULTOS card is enabled, the card is given a set of Issuer specified "permissions", which the Card Issuer can use to control which applications are loaded onto its card base. These load permissions are checked when the ALC is loaded.
- Confirms the card Issuer has authorised the loading of the application.
- Optionally checks the application unit's integrity using the Application Signature
- Optionally decrypts and validates the ciphertext KTU
    - If valid, decrypts the indicated area(s)

Any failure detected by the MULTOS card will cause the load process to fail and the Create MEL Application command will return a MULTOS specific error code.

All decryption is performed on the MULTOS card, which means the terminal loading the application does not need any cryptographic functions. This means that due to the cryptographic operations performed the last Create MEL Application command tends to take longer to return 9000, if it is successful.

APDU command

Field	Value	Description
CLA	'BE'	This is the MULTOS specific command class for all load / delete commands
INS	'16'	Indicates this is a Create MEL command.
P1	'00'	The most significant byte of the offset.
P2	'00'	The least significant byte of the offset.
Lc field	Var.	The length of the ALC component
Data field	-	The ALC component
Le field	Empty	No response data expected.

An Application Load Certificate may be longer than the maximum permitted within a single APDU. If this is the case, the certificate must be split into consecutive components and sent to the MULTOS card in sequential order. It differs from the way in which the Application Load Unit commands operate in that the Application Load Certificate chunks must be sent in the correct sequence and the P1, P2 parameters are not used to specify an offset.

For example, if the Application Load Certificate is 266 bytes long then the certificate may be sent to the MULTOS card as two chunks of 133 bytes. The first 133 bytes of the certificate are sent first followed by the remaining 133 bytes. The Lc field of the APDI command is used to specify the length of the current chunk. The following would be the APDU commands:

BE,16,00,00, Lc=133, data = ALC[0..132]

BE,16,00,00, Lc=133, data = ALC[133.265]

For each Create MEL Application command, if it is successful, the MULTOS card returns 9000. For the last Create MEL Application command sent to the card, it creates the application ready for use by:

- creating a DIR file record entry, if provided
- creating an ATR file record entry, if provided
- creating FCI for the application (DF), if provided
- removing the KTU and Application Signature from temporary memory

The application is then loaded and ready for use.

4.2 Delete Sequence Overview

The Delete MEL Application command is the only command needed to delete an application from a MULTOS card.

4.2.1 Delete MEL Application

The Delete MEL Application command sends the ADC to a MULTOS card. The MULTOS card:

- Verifies the ADC is a valid delete certificate.
- Confirms the application’s delete permissions (msm_app_permissions) are compatible with the card’s permissions (msm_mcd_permissions).
- Confirms the random seed
- Confirms the Issuer has authorised the deletion of the application.

Any failure detected by the MULTOS card will cause the delete process to fail and the Delete MEL Application command will return a MULTOS specific error code.

APDU command

Field	Value	Description
CLA	‘BE’	This is the MULTOS specific command class for all load / delete commands
INS	‘18’	Indicates this is a Delete MEL Application command.
P1	‘00’	The most significant byte of the offset.
P2	‘00’	The least significant byte of the offset.
Lc field	Var.	The length of the ADC component
Data field	-	The ADC component
Le field	Empty	No response data expected.

If the Delete MEL Application command is successful, MULTOS deletes the application’s:

- Dedicated File (code and data)
- DIR file record entry, if present
- FCI, if present
- ATR file record, if present
- ATR Historical Characters, if application has control
- Session data

The memory cleared by MULTOS is now available to load other applications. The precise details regarding how a MULTOS card manages memory are implementation dependent.

If the ATR Historical Characters were controlled by the deleted application, these will now be generated (hence controlled) by MULTOS in Standard Mode. (See MULTOS Developers Reference Manual for further details).

4.3 Application Load History

The application load history is a list of applications that have been loaded onto the MULTOS card during its lifetime. Applications are only added to the list if the random seed within the Application Load Certificate is non-zero.

The application load history contains the application_id and random seed value. The MULTOS card will not allow an application to be loaded where the application_id and random seed value from the ALC already appear in the history. This has the effect of making certificates with a non-zero random seed single use per MULTOS card. They may be used to load an application once but cannot be used to reload the application.

If an Application Load Certificate with a non-zero random seed is used then the application may only be deleted using a delete certificate which has a random seed value equal to the random seed value used when loading the application.

5 Detailed Application Load Procedure

The following section describes application load procedure in more detail. This is a suggested general design for a system to load a MULTOS application onto a single card. It addresses some design issues that are appropriate to Bureaux, but it does not address inventory management, batch processing considerations etc. necessary if loading cards in high volumes. It is not intended to preclude other possible designs.

In the sections that follow, several flowcharts are used to explain the load sequence. The 'Load Procedure' diagram is the top-level flowchart, which can be broken down into the following processes:

	Process	Diagram Reference
A	Establishing communications with the MULTOS card.	Steps 1 - 3
B	Selecting the correct ALU and ALC.	Steps 4 and 5
C	Check there is sufficient memory on the MULTOS card to load the application.	Steps 6 and 7
D	Splitting the ALU in components.	Step 8
E	Loading the ALU components onto the MULTOS card.	Steps 9 - 14
F	Loading the ALC onto the MULTOS card.	Step 15

In the flowcharts, references are made to record structures defined in the MULTOS CA document File Interface Formats [FIF]

Note: The flowcharts do not show what action to take when an error occurs; this is left to the user's discretion.

5.1 Load Procedure

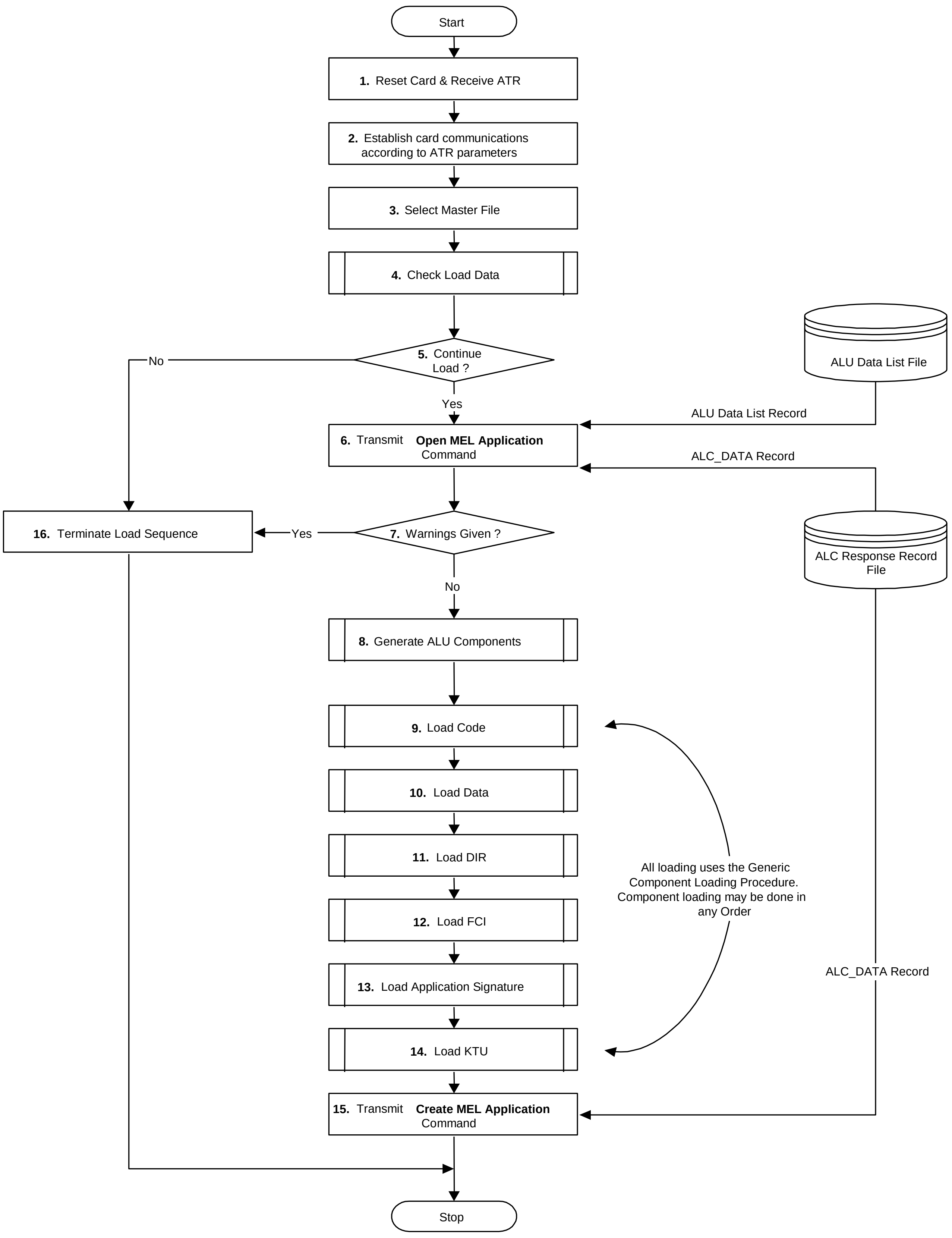

Start
1. Reset Card & Receive ATR
2. Establish card communications according to ATR parameters
3. Select Master File
4. Check Load Data
5. Continue Load ?
No
Yes
ALU Data List File
ALU Data List Record
6. Transmit Open MEL Application Command
ALC_DATA Record
16. Terminate Load Sequence
Yes
7. Warnings Given ?
ALC Response Record File
No
8. Generate ALU Components
9. Load Code
10. Load Data
11. Load DIR
12. Load FCI
13. Load Application Signature
14. Load KTU
All loading uses the Generic Component Loading Procedure. Component loading may be done in any Order
ALC_DATA Record
15. Transmit Create MEL Application Command
Stop

Figure 1

5.1.1 Design Notes: Load Procedure

1. Apply a cold reset to the card according to [7816-3]. The card will respond with the (primary) ATR.
2. The ATR Interface Characters should be checked to ensure they conform to [7816-3]. The ATR Historical Bytes may or may not conform to [7816-3], depending on whether they are generated by MULTOS or a currently loaded application.

   Post ATR communication should be established with the card according to the ATR parameters. ISO default values should be used if any Interface Character is not present in the ATR. Wherever possible, IFDs should use the maximum clock frequency the card supports to minimise data transmission (and hence load) times.
3. When a MULTOS card is reset in "Standard" mode, the Master File (MF) is implicitly selected. In this case, there is no need to select the MF explicitly. If the card is in "Shell" mode, then the shell application is selected at reset, in which case the MF must be explicitly selected using the Select File command.

   The Get Manufacturer Data and Get MULTOS Data commands are only available if the MF (3F 00) is selected and are used later in the load procedure. The MF can be selected with the APDU: 00 A4 00 00 02 3F 00
4. Before continuing with the load process, it is prudent to check the correct ALU and ALC are available.
   The complexity of selecting the correct ALU and ALC will vary from system to system. If the ALU is plaintext, then only one ALU may be needed to load several cards. Otherwise, if the ALU contains encrypted information, then one ALU is required per card. Similarly, one or more ALC may be needed. Generating ALUs is described in [GALU].

   The figure above describes a method for selecting the correct ALU and ALC. It essentially consists of a 3 way cross check between data on the MULTOS card, ALU and ALC. In practice, a subset of these checks may only be necessary (for example selecting the ALU and ALC based on the application_id only).

   High volume systems (e.g. Bureau equipment) running batch processes are likely to have sophisticated inventory systems, which pre-sort ALU and ALC based on file consignment identifiers etc. prior to loading a batch of cards.
5. The load procedure should be terminated if the correct ALU and ALC for the card cannot be found.
6. The Open MEL Application command is of type ISO "case 4". Data for the command APDU should be extracted from either the ALU or ALC as described in section 0. The response APDU data includes the card's public key certificate, which can be discarded and ignored.
7. The [MDRM] defines the possible warnings the Open MEL Application command may produce, including:

   Invalid MCD Issuer Product Id
   Invalid MCD Issuer Id
   Invalid Set MSM Controls Data Date
   Invalid MCD Number

   These indicate that although the Open MEL Application command has not failed, the Create MEL Application command will fail later in the load sequence. To avoid unnecessary transmission of further MULTOS commands, it is recommended an IFD terminates the load sequence if any warning is given.

8. If the Open MEL Application command is successful then the next step is to load the ALU onto the card. To do this, the ALU must first be split into components. Application code components (code_record_components) are loaded using one or more Load Code commands.
9. Application data components (data_record_components) are loaded using one or more Load Data commands.
10. An application's DIR file record (dir_record_components) is loaded using one or more Load DIR File Record commands.
11. An application's FCI record (fci_record_components) is loaded using one or more Load FCI Record commands.
12. The application signature (app_signature_components) is loaded using one or more Load Application Signature commands.
13. The KTU Ciphertext (ktu_components) is loaded using one or more Load KTU Ciphertext commands.
14. The ALC is loaded onto the card using the Create MEL Application command and an ALC_DATA record from the ALC Response Record File returned to the Card Issuer by the MULTOS CA upon request. The file format is defined in the CA document File Interface Formats [FIF].
15. The load termination sequence, when an error occurs, will be design dependent but should generally involve:
    a. Logging errors and notifying connected systems and users.
    b. Performing the card de-activation sequence defined in [7816-3].

5.2 Check Load Data

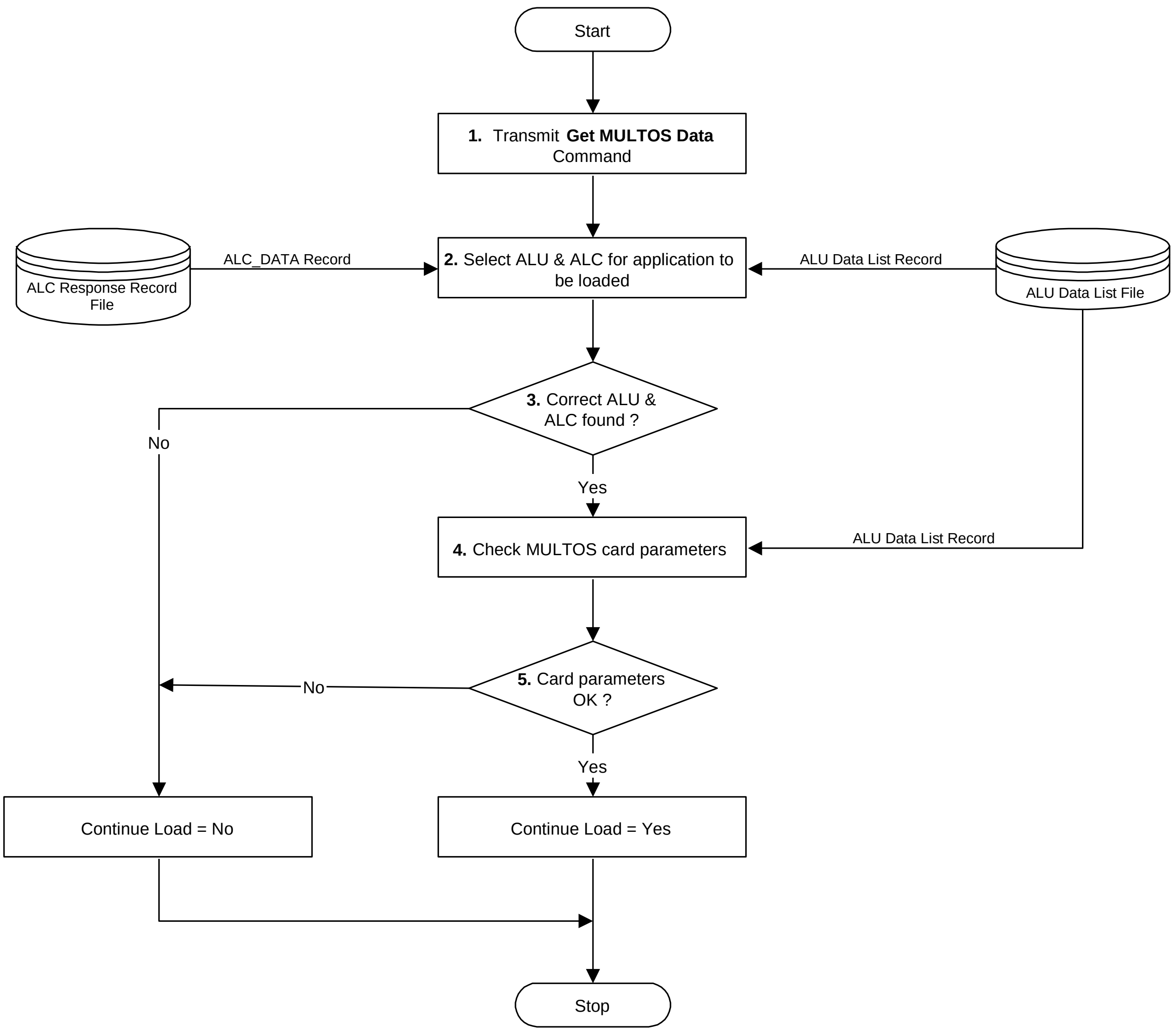


Figure 2

5.2.1 Design Notes: Check Load Data

1. The Get MULTOS Data response contains information about the card (multos_data). In particular, multos_data details the card permissions for loading and deleting applications (msm_mcd_permissions) and the maximum sizes of various data items. The use of multos_data is described in the following sections.
2. Certain data items in the ALU and ALC must match the card otherwise the Create MEL Application command will fail later in the load sequence. These are :

Card Data (multos_data. msm_mcd_permissions)	ALU	ALC (alc_data)
N/A	header_record. application_id	application_id
mcd_issuer_products_id	N/A	msm_app_permissions. mcd_issuer_products_ids
set_msm_controls_data_d	N/A	msm_app_permissions.

ate		set_msm_controls_data_dates
mcd_number	alu_data_list_record. mcd_number	msm_app_permissions. mcd_number
mcd_issuer_id	N/A	msm_app_permissions. mcd_issuer_id
Certification Method ID		Certification Method ID
Hash Method ID		Hash Method ID

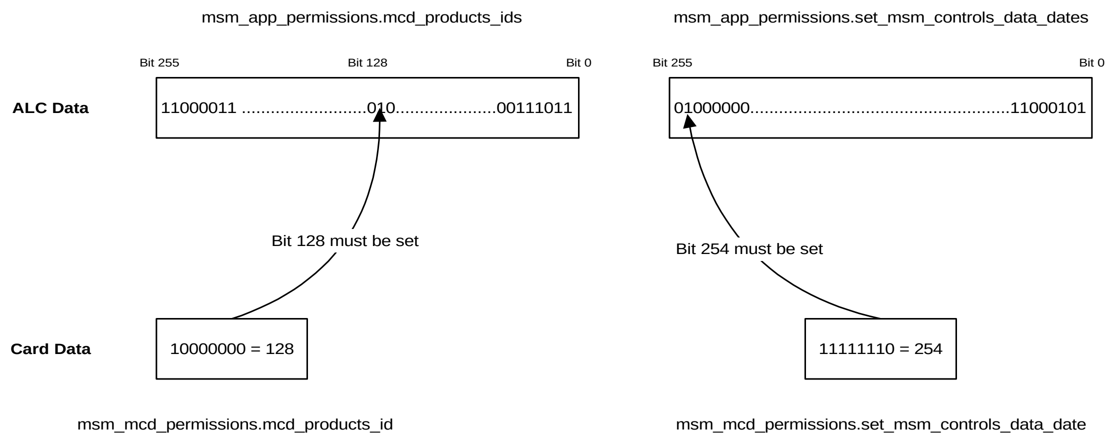
The certification and hash method id fields in the ALC and from the MULTOS card should match. If either of these fields is different the application load will fail.

The certification method ID field is used to indicate how the certificate has been constructed and which key certifying key has been used. If the certificate has been generated in a different way, or using a different key to the MULTOS card, then the card will fail to authenticate the certificate and the load will fail.

The hash method ID field is used to indicate how asymmetrical hashes are calculated and the hash modulus used.

ALU, ALC and Card “matching” rules:

- application_id:
- The application_id identifies the application to be loaded.
- The ALU and ALC application_id must match exactly.
- mcd_issuer_product_id and set_msm_controls_data_date:
- The card’s mcd_issuer_product_id (1 byte) must match the ALC’s mcd_issuer_product_ids (32 bytes) and the card’s set_msm_controls_data_date (1 byte) must match the ALC’s set_msm_controls_data_dates (32 bytes).
- The card’s data is 1 byte, which represents a binary coded decimal between 0 and 255.
- The ALC’s data is 32 bytes long, which is 256 bits (bit 0 to bit 255).
- The card will only permit the application to be created if :
- mcd_issuer_product_ids’ bit value is flagged in the mcd_issuer_product_id bit map AND set_msm_controls_data_dates’ bit value is flagged in the set_msm_controls_data_date bit map. This is explained the in the next diagram:



The set_msm_controls_data_date is assigned by the MULTOS CA when the enablement data is generated; the number originally referred to the number of months that had passed since January

1998. For example, if enablement data was generated in January 2001 then the set_msm_controls_data_date would be 23 to indicate that 23 months have passed since January 1998. This date rolled over in May 2019, so for enabled data generated since then it refers to the number of months that have passed since May 2019.

mcd_number:
The mcd_number uniquely identifies the card. The ALU and ALC mcd_number should match the card's mcd_number exactly or set to zero as explained below.
If the ALU contains encrypted information (i.e. data in the code_record or data_record has been encrypted with the card's public key and the ALU contains a KTU), the mcd_number must match the card exactly. Otherwise the ALU mcd_number can be set to zero and the check can be ignored.
If the mcd_number is non-zero then the ALC is card specific (i.e. the ALC's mcd_number must match the card's mcd_number). Alternatively, the card Issuer may request an ALC from the MULTOS CA where the mcd_number is zero. In this case, the ALC may be used to load the same application (application_id) onto several Issuer cards (mcd_issuer_id), subject to the other matching rules listed below.

MCD Issuer ID:
The mcd_issuer_id identifies the card Issuer. The card and ALC mcd_issuer_id must match exactly.

Certification / Hash Method ID:
If the certification / hash method ID used by the MULTOS card does not match the certification / hash method ID used to generate the certificate then the MULTOS card will not be able to authenticate the certificate correctly. The load will fail due to an invalid certificate.

3. If a matching ALU or ALC could not be found then there is no point continuing the load.
4. multos_data contains the max_dir_file_record_size and max_fci_record_size. If either the ALU dir_record_length or ALU fci_record_length exceed these limits then the load sequence will fail.

   This check is optional as any discrepancies will be detected by the Open MEL Application command, before any data is loaded onto the MULTOS card.
5. If the card parameters are checked and a discrepancy is found then there is obviously no point continuing the load.

5.3 ALU Component Generation

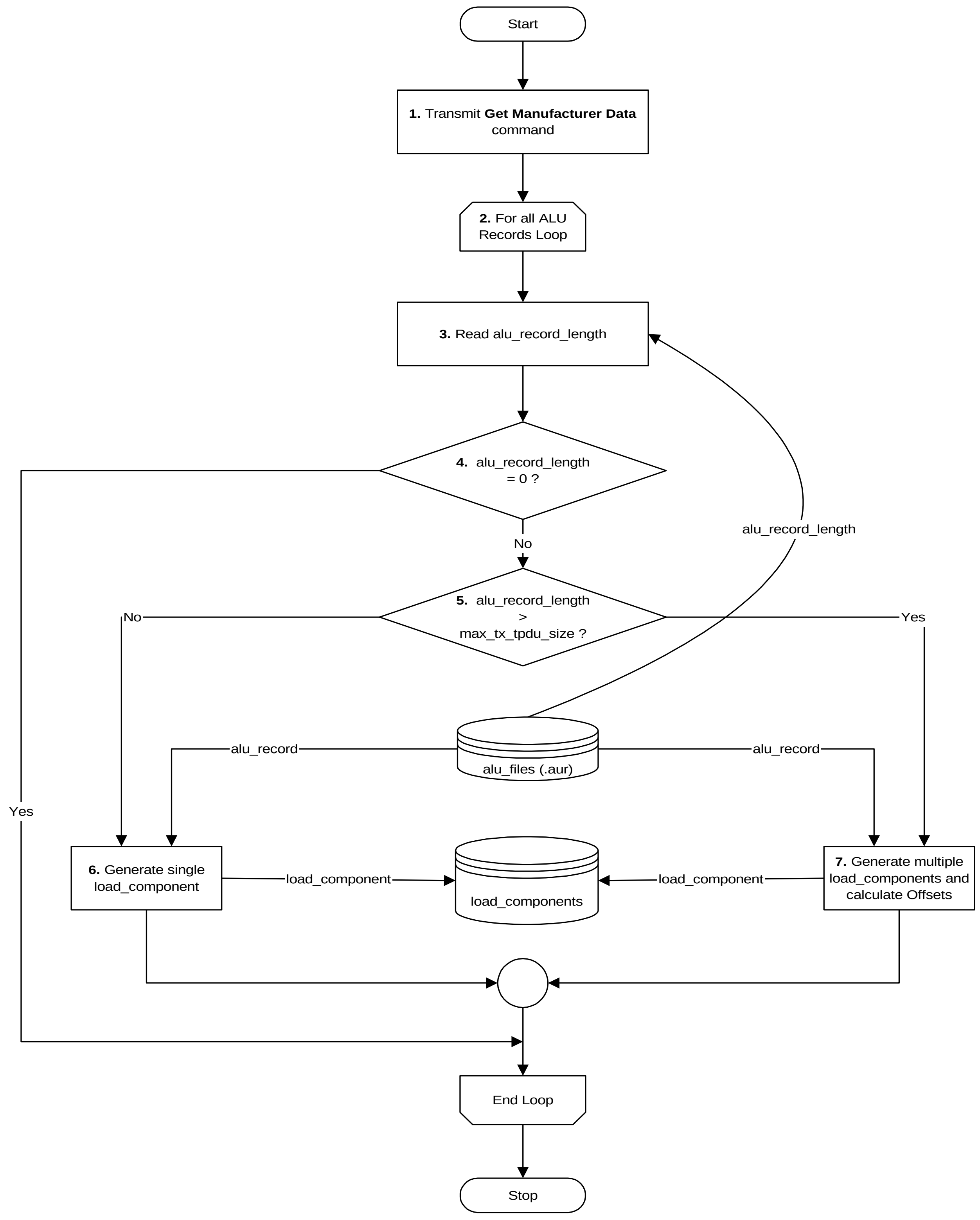

Start
1. Transmit Get Manufacturer Data command
2. For all ALU Records Loop
3. Read alu_record_length
4. alu_record_length = 0 ?
No
Yes
5. alu_record_length > max_tx_tpdu_size ?
No
Yes
alu_record_length
alu_record
alu_files (.aur)
alu_record
6. Generate single load_component
load_component
load_components
load_component
7. Generate multiple load_components and calculate Offsets
End Loop
Stop

Figure 3

In the following section, the following items are defined which are not in the Data Dictionary:

- alu_record: This defines the particular component about to be loaded e.g. code, data.
- alu_record_length: Length of above component
- load_component: APDU command + section of load data
- maxTPDU: Maximum size of data sent to MULTOS
- command_data_field: part of alu_record, split into a size that can be sent in one command.

5.3.1 Design Notes : ALU Component Generation

1. The Get Manufacturer Data response contains information about the card's hardware, including the maximum command and response TPDU lengths supported (max_tx_tpdu_size and max_rx_tpdu_size respectively). The IFD must ensure that any TPDUs sent to the card do not exceed max_tx_tpdu_size.

   This is particularly relevant to the "Load Commands". These are ISO "case 3" commands (see the MULTOS Developers Guide [MDG] and [7816-4] for a definition of "case 3") which means that Lc must not exceed max_tx_tpdu_size. In other words, the maximum size of a load_component is max_tx_tpdu_size.
2. load_components are needed for each type of alu_record :

   code record
   data record
   DIR record
   FCI record
   Application Signature
   KTU

   An ALU may contain all or only some of these records. Typically an application, which requires no encryption to protect the ALU, will only need a code record, data record and DIR record.
3. Read the alu_record_length from the alu_files.
4. If the alu_record_length is zero then the corresponding alu_record is empty (null). In which case there are no load_components to generate.
5. If an ALU record is longer than the maximum TPDU size the MULTOS card supports then the record must be split into components. Otherwise only one load_component is needed and the ALU record data can be sent in one command.
6. If only a single load_component is needed then set :

   the offset to zero i.e. P1 = 0x00 and P2 = 0x00
   Lc = alu_record_length
   command_data_field = alu_record
7. the alu_record must be divided into 2 or more load_components as described below (see section 0 also) :

   - Determine the max_command_data_field_size (maxTPDU). Ideally, this should be max_tx_tpdu_size as this ensures the minimum number of components are needed. Otherwise any size less than max_tx_tpdu_size can be used.
   - Determine number of components needed = (alu_record_length /maxTPDU) rounded up to the nearest integer.
   - Calculate the offset (P1 and P2) from the base (0x0000) where :
     i. 1st component offset = 0x0000

   ii. 2nd component offset = 0x(maxTPDU) – 1
   iii. 3rd component offset = 2nd component offset + 0x(maxTPDU)
   iv. 4th component offset = 3rd component offset + 0x(maxTPDU) etc.

- Split alu_record into command_data_fields where :
   i. 1st command_data_field = 1st maxTPDU bytes of alu_record
   ii. 2nd command_data_field = next maxTPDU bytes following 1st maxTPDU bytes
   iii. 3rd command_data_field = next maxTPDU bytes following 2nd maxTPDU bytes
   iv. The length of the command_data_field will be 6 bytes for all except (possibly) the last command_data_field. This will contain the remaining bytes of the alu_record.

Example

code_record_length = 0x0320 (800 Bytes) and
Let maxTPDU = max_tx_tpdu_size = 0x00FF (255 Bytes) and
Number of data blocks needed = (800/255) rounded up to nearest integer = 4

Calculate offsets	P1	P2
1st data block offset = 0x0000	0x00	0x00
2nd data block offset = 0x00FF	0x00	0xFF
3rd data block offset = 0x00FF + 0x00FF = 0x01FE	0x01	0xFE
4th data block offset = 0x01FE + 0x00FF = 0x02FD	0x02	0xFD

Split code_record into components of max. size 0x00FF bytes	Lc
1st command_data_field = 1st 255 bytes of code_record (bytes 1 .. 255)	0xFF
2nd command_data_field = next 255 bytes of code_record (bytes 256 .. 510)	0xFF
3rd command_data_field = next 255 bytes of code_record (bytes 511 .. 765)	0xFF
4th command_data_field = remaining 35 bytes of code_record (bytes 766 .. 800)	0x23

The table below shows the load_components (Load Code Command APDUs):

CLA	INS	P1	P2	Lc	Data Field	Le
0xBE	0x24	0x00	0x00	0xFF	1st command_data_field	Null
0xBE	0x24	0x00	0xFF	0xFF	2nd command_data_field	Null
0xBE	0x24	0x01	0xFE	0xFF	3rd command_data_field	Null
0xBE	0x24	0x02	0xFD	0x23	4th command_data_field	Null

5.4 Component Loading

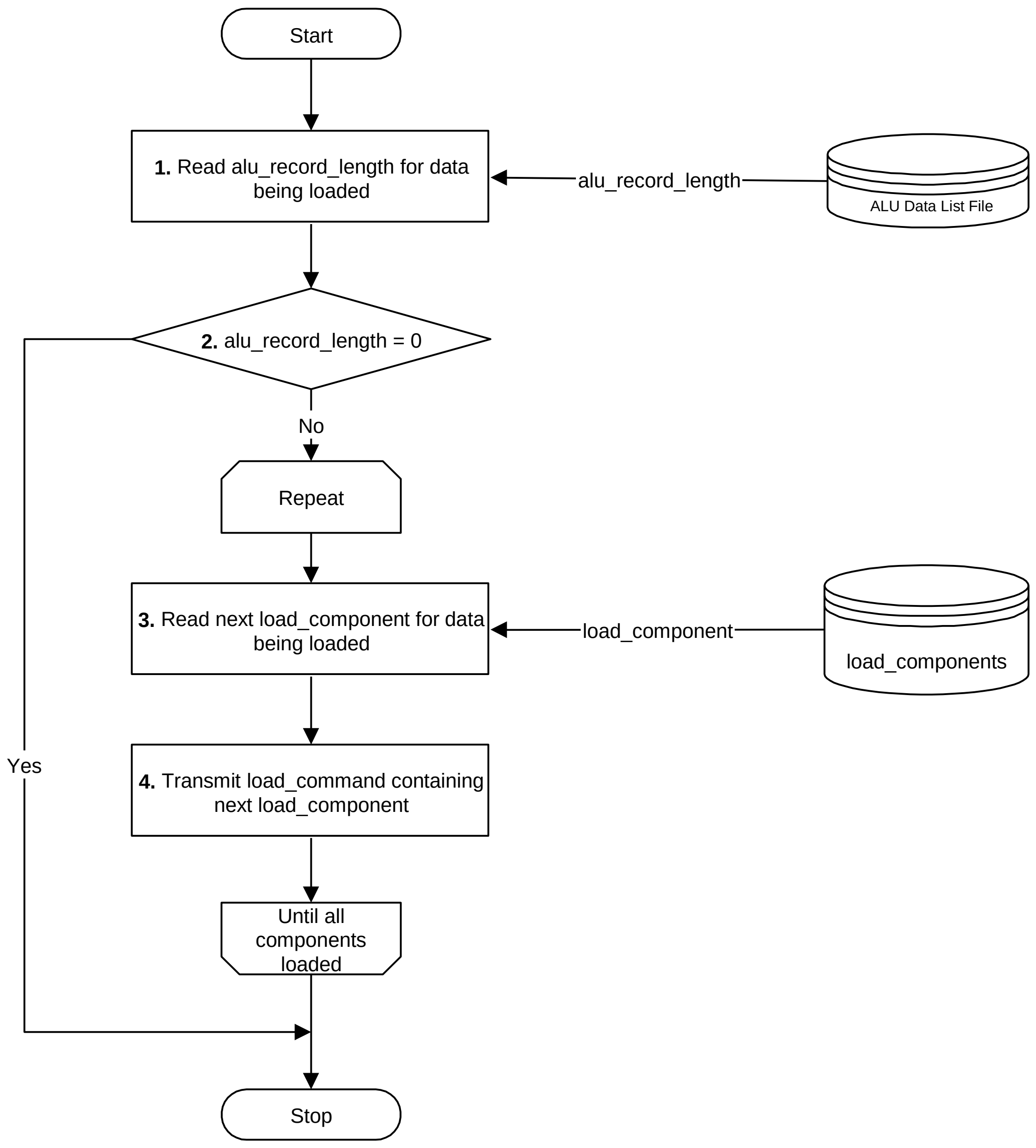

Start
1. Read alu_record_length for data being loaded
alu_record_length
ALU Data List File
2. alu_record_length = 0
No
Yes
Repeat
3. Read next load_component for data being loaded
load_component
load_components
4. Transmit load_command containing next load_component
Until all components loaded
Stop

Figure 4

5.4.1 Design Notes: Component Loading

1. Read the alu_record_length from the alu_files.
2. If the alu_record_length is zero then the associated alu_record is empty (null) and hence no loading is necessary.
3. Read the load_component for the appropriate load_command. The load components can be transmitted in any order.
4. Transmit the load_command APDU to the card

6 Detailed Application Delete Procedure

This section describes the application delete procedure in more detail. This is a suggested general design for a system to delete a MULTOS application from a single card. It addresses some design issues that are appropriate to Bureaux, but it does not address inventory management, batch processing considerations etc. necessary if deleting applications from cards in high volumes. It is not intended to preclude other possible designs.

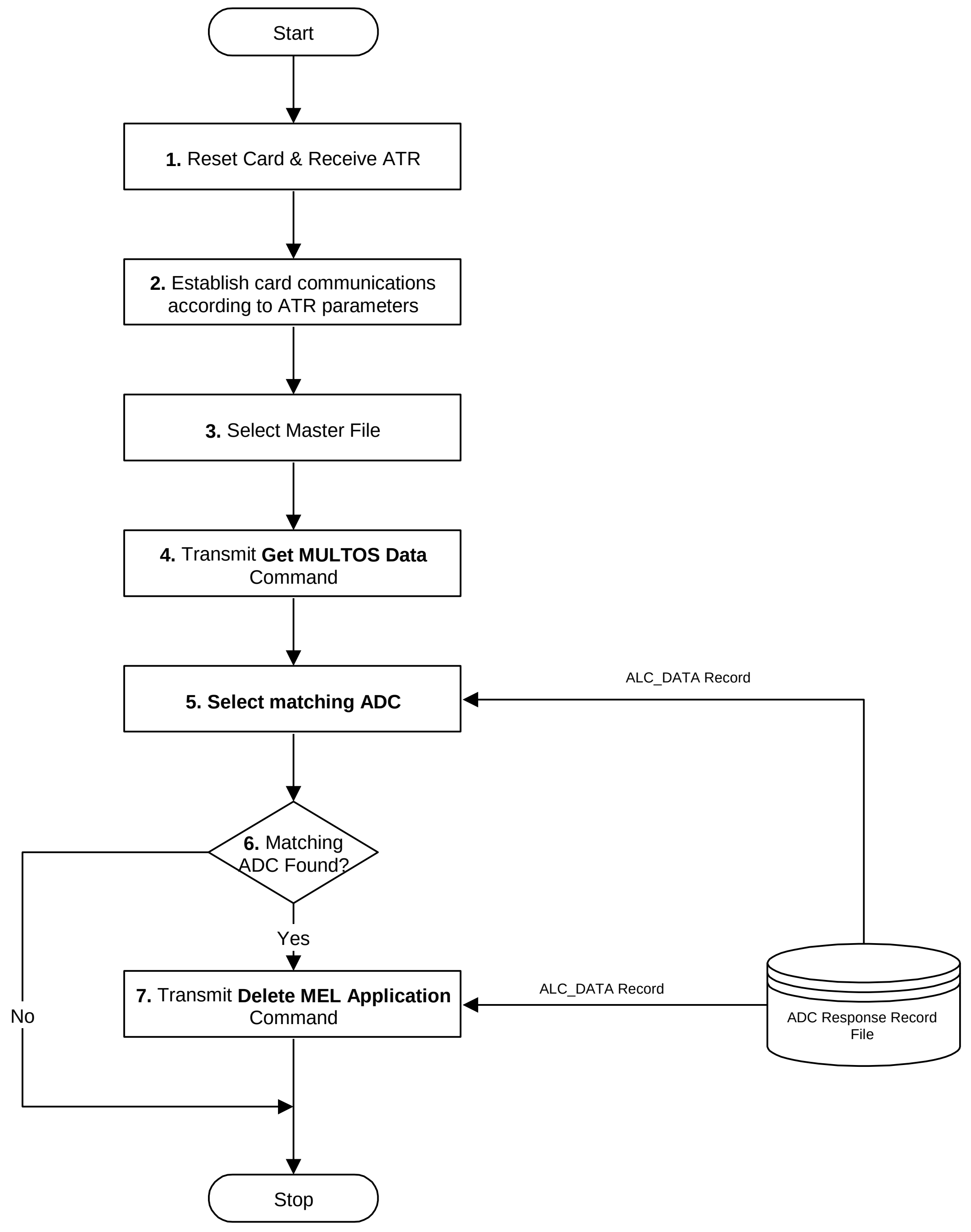
The application delete procedure is much simpler than the load procedure and involves:

	Process	Reference
A	Establishing communications with the MULTOS card.	Steps 1 - 4
B	Selecting the correct ADC.	Steps 5 - 6
C	Loading the ADC onto the MULTOS card.	Step 7

In the sections that follow, flowcharts are used to explain the application delete sequence and should be read together with the notes referenced in the flowchart.

Note: The flowcharts do not show what action to take when an error occurs; this is left to the user's discretion.

6.1 Delete Procedure


Start
1. Reset Card & Receive ATR
2. Establish card communications according to ATR parameters
3. Select Master File
4. Transmit Get MULTOS Data Command
5. Select matching ADC
ALC_DATA Record
6. Matching ADC Found?
Yes
No
7. Transmit Delete MEL Application Command
ALC_DATA Record
ADC Response Record File
Stop

6.2 Design Notes : Application Delete Procedure

1-3 See the corresponding Figure 1 design notes.

4. The Get MULTOS Data response contains information about the card (multos_data). In particular, multos_data details the card permissions for loading and deleting applications (msm_mcd_permissions) and the maximum sizes of various data items. The use of multos_data is described in the following sections.
5. Certain data items in the ADC must match the card otherwise the Delete MEL Application command will fail. These are:

Card Data (multos_data. msm_mcd_permissions)	ADC (adc_data)
mcd_issuer_products_Id	msm_app_permissions.mcd_issuer_products_ids
set_msm_controls_data_date	msm_app_permissions.set_msm_controls_data_dates
mcd_number	msm_app_permissions.mcd_number
mcd_issuer_id	msm_app_permissions.mcd_issuer_id

ADC and Card “matching” rules for the above are the same as those detailed in section previous section.

6. If no matching ADC was found then there is no point continuing the delete procedure.
7. The ADC is loaded onto the card using the Delete MEL Application command and an ADC_DATA record from the ADC Response Record File returned to the Card Issuer by the MULTOS CA upon request. The file format is defined in the CA document File Interface Formats [FIF].

----- End of Document -----