



Cyberflex[®] Access[™] Cards Programmer's Guide

*Cyberflex Access
Software Development Kit 4.4*

Trademarks

Schlumberger, SchlumbergerSema, Cryptoflex, Cyberflex, Cyberflex Access, e-gate, and J-card are trademarks or registered trademarks of Schlumberger or SchlumbergerSema.

Adobe and Acrobat Reader are trademarks or registered trademarks of Adobe Systems, Inc. Microsoft, Windows, Windows NT, Visual J++, and Developer Studio are trademarks or registered trademarks of Microsoft Corporation. Sun and Java are trademarks or registered trademarks of Sun Microsystems, Inc. Other company, product, and service names may be trademarks or service marks of others.

Document Edition	Date
C300473_rev2	February 2003

Copyright© 1998-2003 Schlumberger and SchlumbergerSema

All rights reserved.

You will find the Cyberflex Access Software Development Kit software license agreement (*SDK_license.rtf*) in the following directory: *\Program Files\Schlumberger\Smart Cards and Terminals\Cyberflex Access Kits\v4\Documentation*.

The Cyberflex Access Software Development Kit software license agreement is also available from the Cyberflex Access support website: *www.cyberflex.com/Support/support.html*.

Your feedback about this manual is welcome! Comments, questions, and suggestions about any part of the Cyberflex Access documentation library can be posted to the Docs & Samples Conference of the User Discussion Forums: *www.flexforum.com/cgi-bin/dcforum/dcboard.cgi*.



Preface

Who Should Use This Guide.	xi
What Is in This Guide	xii
Document Conventions	xiii
Acronyms	xiv
Other Sources of Information	xvi

1 Background Concepts

Elements Found by Default on a New Card.	2
Applications.	3
Security Domains	3
Card Manager	4
Secure Channels	5
Stages in a Secure Channel Session	5
Security Levels for Secure Channels	7
Keys and Key Sets	8
Uses for Keys.	8
Default Key Set	9
Session Keys and Static Keys	10
Application Privileges	10
Privileges and Attributes Specified in the Application	
Privileges Byte.	11
Working with the Application Privileges Byte	12

DAP Verification	13
How Verification Requirements Are Set	13
Types of DAP Verification Supported	14
MAC Verification	14
MAC+Enc Verification	15
Life Cycle States and Transitions	16
Card Life Cycle States and Transitions	16
Applet Instance Life Cycle States and Transitions	18
Load File Life Cycle States and Transitions	21
Security Domain Life Cycle States and Transitions	21
Terminology	23
Other Card Elements	25
Global PIN	25
Card Production Life Cycle (CPLC) Data	25
Card Issuer and BIN Data	26

2 Guide to Using the Card Commands

Personalizing the Card	28
Roadmap for Personalizing a Card	28
Using Secure Channels	29
Establishing a Secure Channel	29
Working with Keys and Key Sets	31
Setting the Security Level for a Secure Channel Session	31
Data Authentication Patterns Supported	32

Protected Commands	33
Adding a Key Set to the Card	33
Retrieving Key Set Data	34
Changing Key Values and Key Set Version Numbers	35
Adding and Working with Card Applications	36
Creating an Applet Instance	36
Deleting Load Files and Applet Instances	39
Selecting an Application	41
Making an Application Selectable or Selected By Default	42
Blocking, Locking, and Terminating Card Elements and Cards	44
Working with Card Status and Other Data	47
Retrieving Life Cycle State Data	47
Changing the Life Cycle State of an Application	48
Retrieving Card Production Life Cycle Data	50
Adding and Updating Card Production Life Cycle Data	50
Retrieving Card Issuer Data or Card Issuer BIN Data	51
Adding and Updating Card Issuer Data and Card Issuer BIN	51
Updating AID Values	51
Adding and Updating the Global PIN	52
Other Data You Can Retrieve from the Card	52
Encrypting and Decrypting Commands	53
MAC Verification	53
MAC+Enc Verification	57

3 Card Commands

Introduction..... 61

Command Overview..... 62

Overview of Status Words..... 65

Delete..... 66

 Example 68

 Response Data 69

ExternalAuthenticate..... 70

 Host Cryptogram Calculation..... 72

 Example 73

GetData 75

 Example 76

 CPLC Response Data..... 77

 Issuer BIN Response Data 78

 Card Issuer Response Data 78

GetStatus..... 80

 Example 82

 Response Data 83

InitializeUpdate..... 85

 Key Diversification 87

 Example 89

 Response Data 90

 Card Cryptogram Calculation 91

Install	93
Input Data for Adding a Load File to the Card	95
Input Data for Installing an Applet Instance	98
Input Data for Installing a Security Domain	101
Input Data for Making an Application Selectable	102
Example	104
Response Data	105
Load	107
Response Data	109
Format of Input Data: No Verification	109
Format of Input Data: First Block Verified	110
Format of Input Data: Final Block Verified	111
PinChange	113
Format of Input Data	115
PutData	116
PutKey	120
Format of Input Data	123
Response Data Returned by the Card	124
Key Encryption Mechanism	125
Key Check Value Processing	125
SelectApplication	127
Example	129
Response Data (Card Manager or Other Security Domain Selected)	130
Response Data (Applet Instance Selected)	131

SetStatus	133
Example	135

4 Working with Card Applets

Guidelines for Developing Applets.	137
Card Resource Limitations.	137
Java Card Development Requirements.	138
General Programming Guidelines	140
Smart Card Development in a Multi-Application Environment.	141
Roadmap for Developing a Card Applet	143
Writing a Sample Card Applet.	146
Overview of the Sample Applet	146
Creating the Sample Applet	147
Converting Source Code to a Card Application	160
Step 1: Compile the Source Code	160
Step 2: Convert the Class File to a Program File	160
Step 3: Download the Program File as a Load File	166
Step 4: Instantiate the Applet	168
Sending APDU Commands to the Sample Application	171

5 Java Card 2.1.1 API Support

 Support for the javacard.framework Package 177

 Support for the javacard.security Package..... 177

 Support Status for the javacardx.crypto Package 178

A Command Conventions and APDU Basics 179

 Command Description Table Components..... 180

 TPDU Protocol 182

 ISO Protocol Basics 182

 Case 1: No Input or Output..... 182

 Case 2: Receive Mode 183

 Case 3: Send Mode..... 183

 Case 4: Send/Receive Mode 184

B Status Words..... 185

Glossary 187

Index..... 201



THE *Cyberflex Access Cards Programmer's Guide* introduces the Cyberflex Access smart cards—information security smart cards from SchlumbergerSema that run programs written in Java, the programming language from Sun Microsystems.

This guide contains information about developing a Java smart card program for Cyberflex Access cards, and includes a sample program to get you started.

Who Should Use This Guide

This guide is intended for experienced programmers who are interested in developing programs to run on Open Platform Cyberflex Access smart cards.

Supported Cards

The SchlumbergerSema Cyberflex Access smart cards that are supported are listed in this table:

Open Platform Cyberflex Access Card	supports T=0 protocol	supports T=1 protocol
Cyberflex Access 32K card	Y	
Cyberflex Access Developer 32K card	Y	
Cyberflex Access 32K v4 card	Y	Y
Cyberflex Access e-gate 32K card	Y	
Cyberflex Access 64K v1 card	Y	Y

The Cyberflex Access e-gate 32K card adds bytecode verification (Codeshield feature) for applets downloaded to the smart card, supports USB communication protocol, and has some differences in the implementation of the PutKey command.

The Cyberflex Access 64K v1 card additionally provides 48K of available EEPROM for applets and data, and meets FIPS security standard 140-1 level 2.

NOTE *The Cyberflex Access 16K card is no longer available, although the commands are still supported for anyone who is using this older card.*

What Is in This Guide

This guide contains information to help you develop a Cyberflex Access smart card program as you write, test, prepare, load, and run the program on the card.

- “Background Concepts,” on page 1, contains detailed basic concept information about the Open Platform aspects of the Cyberflex Access cards.
- “Card Commands,” on page 61, describes the card commands.
- “Working with Card Applets,” on page 137, takes you step-by-step through the process of writing an example applet, from developing code through loading and executing the applet instance on the card.
- “Java Card 2.1.1 API Support,” on page 177, lists the few interfaces and classes in the Java Card 2.1.1 API specification that the Cyberflex Access cards do not currently implement. Algorithms that are not currently included are also listed.
- “Command Conventions and APDU Basics,” on page 179, describes the APDU components, ISO-specified input and output command formats, and the command conventions used in this document.
- “Status Words,” on page 185, is an overview of the status words (error codes) the Cyberflex Access cards can return in response to APDU commands.

Document Conventions

This guide uses the following conventions:

Bold

Highlights the names of buttons and menu options you select to perform a task.

Italic

Emphasizes text, and distinguishes a new term, file name, or library name.

`monospaced font`

Identifies command names, code examples, terminal output, method names, class names, and field entries.

NOTE

Points out supplementary information about the current topic.



Marks critical information that can help you avoid potential problems.



ACL – Access Control Level privilege. The command can succeed only if the specified ACL privilege is enabled for the currently logged-in card user identity.

Acronyms

The following acronyms are used in the documentation. For more information about these and other terms, see the glossary.

- 0000** — Reserved ID for CHV1 key file
- 0001** — Reserved ID for internal key file (EF Key Int)
- 0011** — Reserved ID for external key file (EF Key Ext)
- 0012** — Reserved ID for private key file (EF RSA-PRI)
- 0100** — Reserved ID for CHV2 key file
- 1012** — Reserved ID for public key file (EF RSA-PUB)
- 0012** — Reserved ID for private key file (EF RSA-PRI)
- 3DES** — Triple-DES
- 3F00** — Reserved ID for the card's master file (root directory)
- ACL** — Access Control Level
- AID** — Application IDentifier
- APDU** — Application Protocol Data Unit
- ATR** — Answer To Reset
- CAD** — card acceptance device
- CAP** — Converted applet file
- CBC** — Cipher Block Chaining mode of DES encryption/decryption
- CHV** — CardHolder Verification
- CLA** — CLAss (First byte of an APDU)
- COVE** — Cryptographic Object Viewer and Editor (Cyberflex Access application)
- CY** — CYclic elementary file
- DES** — Data Encryption Standard
- DF** — Dedicated File (card directory)
- EBC** — Electronic Book Code

EEPROM — Electrically Erasable Programmable Read-Only Memory

EF — Elementary File

FCI — file control information

GPOS — General Purpose Operating System (the Cyberflex Access default operating system)

INS — INstruction (Second byte of an APDU)

ICV — initial chaining vector (used in CBC operations)

IJC — interoperable javacard CAP file (program file)

JCRE — Java Card Runtime Environment

LSB — Least Significant Byte

LSN — Least Significant Nibble

MAC — Message Authentication Code

MF — Master File

mod — Modulus

MSB — Most Significant Bit

MSN — Most Significant Nibble

P1, P2, P3 — Parameters 1, 2, 3 (third, fourth, and fifth bytes of an APDU)

PIN — Personal Identification Number

RFU — Reserved for Future Use

SDK — Software Development Kit

SFI — Short File Identifier

SHA — Secure Hash Algorithm

TLV — Type Length Value

Other Sources of Information

For more information about the Cyberflex Access Software Development Kit, see these additional documents:

- *Cyberflex Access Software Development Kit Guide* – Information about installation, background concepts, instructions for using the development tools (the Smart Card Toolkit and COVE, the Cryptographic Object Viewer and Editor), and how to get certificates, and sign and encrypt email.
- *Guide to SchlumbergerSema Smart Card Middleware* – Description of the interoperability layers that support higher-level functions in smart card programs for all the cards supported by the Cyberflex Access Software Development Kit.
- *Cryptoflex Cards Programmer's Guide* – Descriptions of Cryptoflex™ card commands, error code information, and a tutorial for building a Cryptoflex card program.

If you decide to install the documentation, the default installation directory for the manuals (.pdf format) is here:

```
C:\Program Files\Schlumberger\Smart Cards and Terminals\  
Cyberflex Access Kits\v4\Documentation
```

The manuals are also available on the distribution CD-ROM. In addition, current and previous manuals are linked from the Cyberflex Access Support website and the Cryptoflex Support website (see table on following page).

The websites listed in the following table have useful information about smart cards and cryptography:

Web Site / Email Address	Contents
<i>www.cyberflex.com</i>	SchlumbergerSema Cyberflex Access home page
<i>www.cyberflex.com/Support/support.html</i>	The SchlumbergerSema Cyberflex Access Support page: links to technical help, software updates, and current documentation
<i>www.cryptoflex.com</i>	The SchlumbergerSema Cryptoflex home page
<i>www.cryptoflex.com/Support/support.html</i>	The SchlumbergerSema Cryptoflex Support page: links to a user's discussion forum, FAQ, technical support, the most current documentation, and an email link (<i>cryptoflex@slb.com</i>)
<i>www1.slb.com/smartcards</i>	Information about SchlumbergerSema smart card products
<i>www.reflexreaders.com</i>	Information about the SchlumbergerSema Reflex series smart card readers
<i>www.reflexreaders.com/Support/support.html</i>	The SchlumbergerSema card reader support page: links to drivers, technical help, and documentation
<i>www.scmegastore.com</i>	The SchlumbergerSema smart card marketplace
<i>www.pcscworkgroup.com/</i>	Introductory page for the PC/SC Workgroup, which develops the specifications for the Personal Computer/Smart Card (PC/SC) standard
<i>www.microsoft.com/smartcard</i>	Microsoft site for information about developing CryptoAPI-compliant smart card programs
<i>www.rsasecurity.com</i>	Introductory page for RSA Security, which sets standards for Secure Electronic Transactions (SET), Data Encryption Standards (DES), and Public Key Cryptography Standards (PKCS).

Web Site / Email Address	Contents
<i>www.rsasecurity.com/rsalabs/faq/index.html</i>	RSA Security's cryptography FAQ
<i>www.emvco.com/</i>	Front page for the EMVCo, which develops EMV Integrated Circuit Card Specifications
<i>www.iso.ch</i>	Front page for the International Standardization Organization (ISO), which develops smart card specification #7816 (parts 1–8)



Background Concepts

This section describes components on the Open Platform Cyberflex Access cards and basic concepts about the Open Platform aspects of the cards. The major topics covered are:

- Elements Found by Default on a New Card (page 2)
- Secure Channels (page 5), which includes these topics
 - Stages in a Secure Channel Session (page 5)
 - Security Levels for Secure Channels (page 7)
- Keys and Key Sets (page 8)
- Application Privileges (page 10)
- DAP Verification (page 13), which includes
 - MAC Verification (page 14)
 - MAC+Enc Verification (page 15)
- Life Cycle States and Transitions (page 16) for the following card elements
 - The Card Manager (page 16)
 - Applet instances (page 18)
 - Load files (page 21)
 - Security domains (page 21)
- Other Card Elements (page 25), which covers
 - Global PIN (page 25)
 - Card Production Life Cycle (CPLC) Data (page 25)
 - Card Issuer and BIN Data (page 26)

Elements Found by Default on a New Card

A new Cyberflex Access card contains the following elements:

- Card Manager application (page 4)
- Default key set for the Card Manager (page 9)
- Card Production Life Cycle (CPLC) data (page 25)
- Global Platform compatibility for Java Card Virtual Machine v2.1

You can use the Card Manager's APDU commands to add these elements to a new card:

- Additional key sets for the Card Manager
- Load files
- Applications other than the Card Manager:
 - Applet instances
 - Additional security domains
- A global PIN
- Two types of card identification data in addition to the default CPLC data: card issuer data and the card issuer BIN

NOTES

- *The answer to reset code (ATR) of a Cyberflex Access Developer 32K card begins with: 3B 17 13 9C 12*

The ATR mask is: FF FF 00 FF 00 00 00 00 00 00

- *The answer to reset code (ATR) of a Cyberflex Access e-gate 32K card is:*

3B 75 94 00 00 62 02 02 00 80

The ATR mask is: FF FF FF 00 00 FF FF FF 00 00

- *The answer to reset code (ATR) of a Cyberflex Access 32K v4 card is:*

3b 76 00 00 00 00 9c 11 01 00 00

- *The answer to reset code (ATR) of a Cyberflex Access 64K v1 card is:*

3b 75 00 00 00 29 05 01 01 01 (for the non-FIPS-compliant card, softmask 1.1)

3b 75 00 00 00 29 05 01 02 01 (for the FIPS-compliant card, softmask 2.1)

The ATR mask is: FF FF 00 00 00 FF FF FF 00 00

Applications

An *application* is an agent on a Cyberflex Access card that can be selected to process commands. Any of the following card elements can be an application:

- The Card Manager
- A security domain other than the Card Manager
- An applet instance

- ALSO SEE**
- *Changing an application's life cycle state to selectable* — page 19
 - *Privilege for making an application selected by default* — page 11

Security Domains

A *security domain* is an environment established on a card to protect card contents that are loaded in the security domain, such as an applet and its collateral data. Security domains make it possible for multiple providers to place applets on a smart card without compromising each other's security.

A new Cyberflex Access card contains only one security domain—the Card Manager. You can add other security domains at any point in a card's life, provided the card is not terminated.

The Card Manager and the card's subsidiary security domains provide identical cryptographic services through the Java Card API. You could think of a security domain as a key repository with cryptographic services. The subsidiary security domains process almost the same set of APDU commands the Card Manager does. A few commands are exclusive to the Card Manager, which handle downloading load files, installing applet instances, and deleting card elements.

- ALSO SEE**
- *Secure channels* — page 5
 - *Key sets* — page 8
 - *Adding a Key Set to the Card* — page 33
 - *Changing Key Values* — page 35

Card Manager

The *Card Manager* is the application that acts as the card's central administrator. The Card Manager handles incoming APDU commands by default, unless another application is currently *selected* or is specified as the *default selected application*.

In addition to being an application, the Card Manager is also a security domain. In this capacity, the Card Manager provides services through the Java Card API. Like all security domains, the Card Manager uses one of its key sets to establish a secure channel, which has a defined security level that protects data transmission and data access. (A new card uses its default key set to establish a secure channel.)

Unique among the card's security domains, the Card Manager can execute all of the card APDU commands, including those used for:

- Adding and deleting load files
- Installing and deleting applet instances and subsidiary security domains
- Managing issuer security domains and keys
- Providing information about the contents of the card



Open Platform Implementation— *The current implementation of the Open Platform Cyberflex Access cards does not support delegated tasks. Therefore, the Card Manager is always responsible for the tasks listed above.*

- ALSO SEE**
- *Making an application selectable — page 18 and page 21*
 - *Privilege for making an application selected by default — page 11*

Secure Channels

A *secure channel* is a protocol you establish to transfer commands and data securely between a host-side application and an Open Platform-compliant smart card.

Stages in a Secure Channel Session

A secure channel session is divided into three sequential phases:

- Initiation (described in the following text)
- Operation (page 6)
- Termination (page 6)

Initiating a Secure Channel

To establish a secure channel, the host-side application and currently selected security domain must authenticate each other—using keys in a security domain key set and defining a security level for all the protected commands issued during the secure channel session.

Establishing a secure channel is a three-step process:

- 1** You select the security domain that will receive commands over the secure channel (as described on page 41).
- 2** You call an `InitializeUpdate` command (page 85), which initializes the secure channel. If the command succeeds, it authenticates the current security domain and generates the *session keys* that will be used for the duration of the secure channel session.
- 3** The secure channel is established and becomes usable only if an `ExternalAuthenticate` command (page 70) follows and succeeds in authenticating the host-side application.

Using a Secure Channel

The host-side application uses the secure channel to send commands and data to the security domain, and uses the session keys and static KEK key to implement the type of verification defined by the secure channel's security level. The secure channel uses the same set of session keys for the life of the secure channel session.

Most commands are available only when a secure channel is active, with only the exceptions noted on page 30.



Open Platform Implementation — *The current implementation of the Open Platform Cyberflex Access cards supports secure transmission of commands and data to the card, but not secure transmission of response data.*

Terminating a Secure Channel

After you establish a secure channel, it remains open until one of these events occurs:

- You issue an `InitializeUpdate` command — either a successful or unsuccessful one.
- You deselect the currently selected application by calling the `SelectApplication` command. Any attempt you make to select an application terminates the current secure channel, even if you reselect the current application or the command fails.
- A MAC verification fails in a secure channel that requires DAP verification.
- The card is powered off or reset.

ALSO SEE

- *Security levels* — page 7
- *DAP verification* — page 13
- *Key sets and keys* — page 8
- *Session keys* — page 10

Security Levels for Secure Channels

A *security level* is the attribute of a secure channel that determines the type of verification needed to issue a command in the secure channel. Secure channels can have any of three levels of security:

- **Clear** — *No security*: Information is sent to the card in cleartext with no integrity check. Use this security level only for a test card in a physically secure environment. You cannot use the clear security level if the Card Manager is in a secured or locked *life cycle state*. The card enforces this policy by rejecting the `ExternalAuthenticate` command if it does not include *DAP verification* data.
- **MAC** — Integrity at the command level is ensured by signing the command with a MAC. Command sequence integrity is enforced by chaining the command MACs. For example, the MAC of one command becomes the initialization chaining vector (ICV) in the MAC calculation for the next command.
- **MAC+Enc** — Integrity is ensured by signing the command with a MAC. Confidentiality is ensured by encrypting the command input data.

The host-side application specifies the security level for the secure channel in the `ExternalAuthenticate` command. The security level remains unchanged throughout a secure channel session. If a new security level is needed, terminate the current secure channel and establish a new one.

Class Byte Values Associated with Security Levels

- **Clear** — If the security level for the secure channel is *no security*, the supported class (CLA) byte value is 80h (Open Platform command set) or 00h (ISO command). If you specify the CLA byte value as 84h for a *no security* command, the command fails.
- **MAC or MAC+Enc** — If the security level for the secure channel is MAC or MAC+Enc, the supported class (CLA) byte value is 84h. If you specify the CLA byte value as 80h or 00h for a command that requires DAP verification, the command fails.

ALSO SEE

- *DAP verification* — page 13
- *MAC verification* — page 14
- *MAC+Enc verification* — page 15
- *Life cycle states* — page 16
- *Determining the Security Level of a Secure Channel* — page 30

Keys and Key Sets

When you receive a new Open Platform Cyberflex Access card, it comes with a default *key set* for the Card Manager security domain, which contains three static keys:

- *AUTH key* — Authentication and general encryption key ($K_{\text{ENC, AUTH}}$), key 1 in the key set
- *MAC key* — MAC computation key (K_{MAC}), key 2 in the key set
- *KEK key* — Key encryption key (K_{KEK}), key 3 in the key set

NOTE *Keys are relevant to a security domain.*

Uses for Keys

You can use these keys to:

- Authenticate both parties (on-card and off-card entities) when you establish a secure channel to protect commands and data you send to the card (page 5).
- Ensure command integrity and valid command sequence by computing a MAC of APDU commands (page 14) you send in any secure channel that requires DAP verification (page 13).
- Ensure confidentiality by encrypting the command data you send in a secure channel that requires heightened security (a MAC+Enc secure channel, page 15).
- Protect command data by encrypting keys or PINs you write to the card (PutKey command on page 120 and PinChange command on page 113).
- Provide extra protection by double-encrypting keys or PINs you write to the card over a MAC+Enc secure channel (page 15).

Each key in a key set is a double-length (16-byte) 3DES key. When you perform a cryptographic operation with a specified key, you identify the key by its key set version number and its position (index) in the key set.

Default Key Set

The first key set loaded for any security domain is its *default key set*. When you receive a new Open Platform Cyberflex Access card, it comes with only one key set—the Card Manager’s default key set. You use this key set the first time you establish a secure channel with the card. This is the key set you use for MAC and MAC+Enc operations you conduct in the secure channel.

You will continue to use the default key set to establish secure channels unless you load another key set and explicitly select it at the initiation of mutual authentication (in the `InitiateUpdate` command).

You can perform these default key set operations:

- Change the key values in the default key set and the key set version number.
- Establish a secure channel with a security domain other than the Card Manager (if you have added one to the card), and use either its default key set or other key sets in the security domain to establish a secure channel.
- Add other key sets in the Card Manager domain or in other security domains you add to the card.



If you add or change a key set, keep records about the changes. The GlobalPlatform specification does not currently support tracking the card’s key sets or tracking key set contents and version numbers. You cannot retrieve key set data from the card, so you must maintain accurate records.

NOTE *The default Card Manager key set on a new Open Platform Cyberflex Access card has the key set version number 01h. The default key values are:*

- **AUTH key (key 1)** – 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4Fh
- **MAC key (key 2)** – 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4Fh
- **KEK key (key 3)** – 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4Fh

ALSO SEE

- *Secure channels* — page 5
- *Key types* — see the following topic
- *DAP verification* — page 13

Session Keys and Static Keys

A security domain never uses the static AUTH and MAC keys directly—it uses the corresponding *session keys*. Session keys are keys the Card Manager or other security domain generates from static AUTH and MAC keys at the beginning of a secure channel session. The security domain uses these session keys for the duration of the secure channel session, then discards them.

Like the corresponding static keys, the session keys are 3DES keys (16 bytes long). The list below shows the representations for static and session keys:

$$\begin{array}{ll} K_{\text{ENC, AUTH}} \text{ (static AUTH key)} & \rightarrow K_{\text{enc, auth}} \text{ (session AUTH key)} \\ K_{\text{MAC}} \text{ (static MAC key)} & \rightarrow K_{\text{mac}} \text{ (session AUTH key)} \\ K_{\text{KEK}} \text{ (static KEK key)} & \end{array}$$

The security domain uses the KEK key directly to decrypt these types of encrypted data:

- Key data sent with a PutKey command, or
- The global PIN data sent with a PinChange command.

For examples of keys and the process of session key diversification, see “InitializeUpdate,” on page 85, and “Key Diversification,” on page 87.

Application Privileges

An application on the card (an applet instance or security domain) can execute some card commands only if it has the appropriate application privilege. Application privileges are defined when you install the application by the value you specify for the *application privileges byte*.

In addition to enabling or disabling access to particular commands, the application privileges byte specifies whether the installed application is an applet instance or a security domain.

Privileges and Attributes Specified in the Application Privileges Byte

You calculate the value of the application privileges byte by concatenating the binary values from an 8-bit matrix and converting that value to hexadecimal format. Each binary value sets (value = 1) or clears (value = 0) a particular property. The matrix of properties is shown in the illustration below.

bit 8: security domain	bit 7: DAP DES verification	bit 6: RFU (always 0)	bit 5: card locking	bit 4: card termination	bit 3: default selection	bit 2: global PIN modification	bit 1: RFU (always 0)
------------------------------	-----------------------------------	-----------------------------	---------------------------	-------------------------------	--------------------------------	--------------------------------------	-----------------------------

Bit Descriptions

- **Bit 8: Security Domain** — Specifies that the installed object is a security domain.
- **Bit 7: DAP DES Verification** — Activates optional verification for a security domain. *Enable bit 7 only for a security domain.*
- **Bit 6: RFU** — Always set bit 6 to a value of 0.
- **Bit 5: Card Locking** — Enables a privileged application to lock the Card Manager (with the `SetStatus` command).
- **Bit 4: Card Termination** — Enables a privileged application to terminate the card (with the `SetStatus` command).
- **Bit 3: Default Selection** — Sets an applet instance or security domain to be the default selected application.

The default selected application is the application that receives commands by default, unless another application is explicitly selected with a `SelectApplication` command. On a new card, the default selected application is the Card Manager.

- **Bit 2: Global PIN Modification** — Enables an applet instance or security domain to add, unblock, or update a global PIN for the card (`PinChange` command).
- **Bit 1: RFU** — Always set bit 1 to a value of 0. The GlobalPlatform specification v2.0.1 designates bit 1 for mandatory DAP verification, which the card does not currently enforce.

Examples of Application Privileges Byte Values

The following table shows examples of settings for a security domain and applet instance:

	Bit 8 <i>sec domain</i>	Bit 7 <i>DAP</i>	Bit 6 <i>RFU</i>	Bit 5 <i>lock</i>	Bit 4 <i>terminate</i>	Bit 3 <i>default select</i>	Bit 2 <i>PIN change</i>	Bit 1 <i>RFU</i>	Binary Value	Hex Value
1	1	1	0	0	0	1	0	0	11000100 =	C4
2	0	0	0	1	0	1	1	0	00010110 =	16

C4 = *Example 1:* The installed object is a security domain that is selected by default and has optional DAP verification.

16 = *Example 2:* The installed object is an applet instance that is selected by default, can change the card's global PIN, and can lock the card.

Working with the Application Privileges Byte

You can manage application privileges in these ways:

- Specify the application privileges byte for an new applet instance or security domain when you the application on the card with an `Install: Install` or `Install: Install/Make Selectable` command.
- Modify bit 3 of the application privileges byte by calling an `Install: Make Selectable` command. You can make the application selected by default or make it selectable.
- Retrieve the value of the application privileges byte by calling a `GetStatus` command.



Open Platform Implementation — *The current release of the Cyberflex Access Developer 32K card implements the GlobalPlatform specification v2.0.1 for the application privileges byte with these options or limitations:*

- *Bit 6, the delegated management bit, is not implemented.*
- *Bit 1, the mandatory DAP verification bit, is not implemented. Use bit 7 for DAP verification.*

DAP Verification

Data authentication pattern (*DAP*) is a generic Open Platform term for a cryptographic value added to an APDU command (and its input data) to authenticate the command's integrity, confidentiality, or both.

How Verification Requirements Are Set

When you establish a secure channel by executing an `ExternalAuthenticate` command, you specify a security level. The *security level* defines which, if any, cryptographic operations are required to verify the commands you send the card.

A secure channel, as it is currently implemented in the Open Platform Cyberflex Access cards, has three possible security levels (described more fully on page 31):

- Clear — No security
- MAC — Protected commands are sent signed with an ICV-chained MAC
- MAC+Enc — Protected commands have encrypted input data, and the command header and input data are signed with an ICV-chained MAC

NOTE *For more information about security levels, see page 7. For a list of protected commands, see page 33.*



Open Platform Implementation — *The current implementation of the Cyberflex Access Developer 32K card supports security only for commands and data sent to the card (SMAC), not for data the card returns (RMAC).*

Types of DAP Verification Supported

The Open Platform Cyberflex Access cards support the following types of DAP verification. (For a listing of circumstances for each type of use, see page 32.)

- MAC — 3DES cipher block chaining (CBC) mode used with the session MAC key (K_{mac}).
- MAC+Enc — 3DES encryption in CBC mode used with the session MAC key (K_{mac}), along with the session AUTH key ($K_{\text{enc auth}}$). Both keys are sometimes used with the static KEK key (K_{KEK}).
- SHA-1 hashes



Open Platform Implementation — *The current implementation of the Cyberflex Access Developer 32K card does not support delegated management, so it does not generate load or install tokens, and does not return receipts.*

ALSO SEE

- *MAC verification — next topic*
- *MAC+Enc verification — page 15*

MAC Verification

You calculate and include a Message Authentication Code (MAC) of any protected command you send to the card in a MAC secure channel session. (A MAC secure session is one whose security level requires signing commands with a MAC.)

The MAC is calculated with the MAC key in the current secure channel's specified key set. The MAC key is the second key in the key set, and—like the other key set keys—is a 16-byte 3DES key.

During the initialization of a secure channel (in an `InitializeUpdate` operation), the static MAC key (K_{MAC}) is used to generate a MAC session key (K_{mac}). Both you and the currently selected security domain (such as the Card Manager) use the K_{mac} key in CBC mode to:

- Establish a secure channel (`InitializeUpdate` and `ExternalAuthenticate` commands).
- Sign protected commands transmitted over a MAC or MAC+Enc secure channel. Any card command can be protected in a MAC or MAC+Enc secure channel, except: `SelectApplication`, `InitializeUpdate`, and (optionally) a `GetData` command.

NOTES

- *For more information about MAC operations, see page 53.*
- *For information about the format of MAC commands, see page 54.*
- *For information about MAC calculation, see page 56.*

MAC+Enc Verification

If the current secure channel has a security level that requires MAC+Enc, you must sign the command with a MAC and encrypt the command input data.

- *MAC* — Calculate the MAC as described in the previous topic on page 56.
- *Encryption* — Encrypt the input data by using the AUTH session key (K_{auth}) in the current secure channel's specified key set. K_{auth} is a 16-byte 3DES key, and is the first key in the key set.

During the initialization of a secure channel (in an `InitializeUpdate` operation), the static AUTH key ($K_{ENC, AUTH}$) is used to generate the $K_{enc, auth}$. Both you and the currently selected security domain (such as the Card Manager) use the $K_{enc, auth}$ key in CBC mode to:

- Encrypt cleartext input data you send to the card as part of a protected command over a MAC+Enc secure channel.
- Encrypt the SHA-1 hash of load file blocks you include with `Load` commands.
- Double-encrypt key data or global PIN data you send to the card with a `PutKey` or `PinChange` command.

- NOTES**
- For more information about MAC+Enc operations, see page 57.
 - For information about the format of MAC+Enc commands, see page 58.
 - For information about MAC+Enc calculation, see page 60.

Life Cycle States and Transitions

The card elements have *life cycle states* that affect the element's functionality and the security measures required to execute most card commands. The life cycle states and possible transitions are described in the following text.

To retrieve the life cycle state for the Card Manager, other security domains, applet instances, and load files, call the `GetStatus` command (page 80).

To modify the life cycle state for the Card Manager, other security domains, applet instances, and load files, call the `SetStatus` command (page 133).

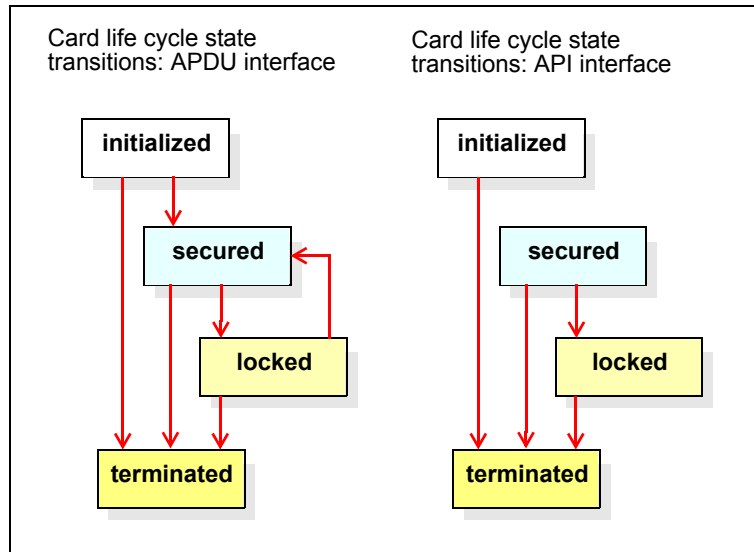
Card Life Cycle States and Transitions

The Card Manager can have any of the following life cycle states:

- **OP_Ready** — The card is in *OP_Ready* state during manufacturing.
- **Initialized** — When you receive a new Open Platform Cyberflex Access card, it is in the *initialized* state. The default key set for the Card Manager has been added, as well as card identification data. The initialized state is irreversible.
- **Secured** — You put the card in *secured* state when you have added the key sets and security elements needed to enforce the card issuer's security policies. The transition to the secured state is irreversible.
- **Locked** — In the *locked* state, all card applications are disabled except for the Card Manager. You can use the locked state to suspend functionality while a security threat is analyzed. You then have the option to return the card to its pre-locked state.
- **Terminated** — If you *terminate* the card, it is permanently disabled. The card no longer responds to attempts to communicate with it. You typically terminate a card in response to a security threat or when the card expires.

Changes You Can Make in the Card Manager Life Cycle State

The following illustration shows the changes you can make to the life cycle state of the Card Manager.



Rules for Changes in Card Manager Life Cycle States

APDU Interface — To change the card state, call a `SetStatus` command. (Note the requirement for locking and terminating privileges described below.)

Locking the card in the API interface — To lock the card through the API interface, call the `org.GlobalPlatform.System.lockCard` method. (This action is available only to a privileged application.)

Terminating the card in the API interface — To terminate the card through the API interface, call the `org.GlobalPlatform.System.terminateCard` method. (This action is available only to a privileged application.)

Privileged Applications — To lock or terminate the card with calls to either the APDU or API interface, you must first establish a secure channel with a security domain that has the appropriate card locking or card termination privilege (as described on page 10).

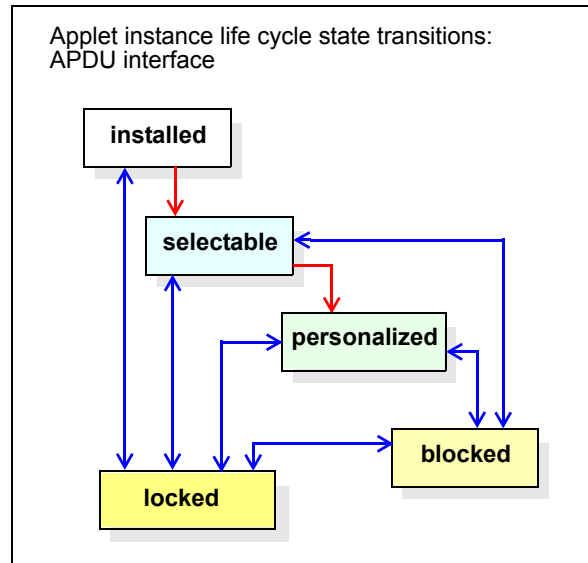
Applet Instance Life Cycle States and Transitions

An applet instance can have any of the following life cycle states:

- **Installed** — An applet instance's state is set to *installed* during the successful execution of an `Install` command, which must be performed with DAP verification by the Card Manager. An installed applet instance has the necessary links and EEPROM allocated for execution. When it is installed, the applet instance AID is entered in the card registry, and the applet instance becomes detectable by an authenticated host-side application.
- **Selectable** — The *selectable* life cycle state enables the applet instance to be selected, so it can receive incoming APDU commands and run a Java Card program. Optionally, you can make the instance selectable as part of the installation process. The Card Manager handles changing the state transition to selectable.
- **Personalized** — Change the applet instance state to *personalized* when it has all the keys and data it needs to run. Personalization requirements are program-dependent.
- **Blocked** — You typically instruct the Card Manager or other current security domain to change an applet instance state to *blocked* when you want to deactivate it in response to a possible security threat. Once an applet instance has reached the selectable or personalized state, it can set its own state to blocked and reverse the transition to blocked, returning the applet instance to its previous state.
- **Locked** — The Card Manager can lock an applet instance at any time for security protection or for business reasons. In the *locked* state, the applet instance is not selectable and is completely inactive. Only the Card Manager can unlock an applet instance.
- **Deleted** — The current implementation of the Open Platform Cyberflex Access cards does not support logical deletion. To delete an applet instance, call a `Delete` command and delete the instance physically. (You can also call the `Delete` command through the Smart Card Toolkit interface, as described in the *Cyberflex Access Software Development Kit User's Guide*.)

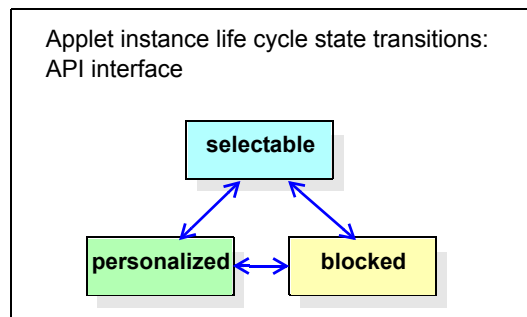
Changing the Life Cycle State of an Applet Instance with APDU Commands

The following illustration shows the changes you can make in the life cycle state of an applet instance by sending a `SetStatus` command to the Card Manager or other currently selected security domain. (For information about the rules that apply to life cycle state transitions, see page 20.)



Changing the Life Cycle State of an Applet Instance Through API Calls

The following illustration shows the life cycle state changes you can make to an applet instance through the Open Platform API—by calling the `org.GlobalPlatform.System.setCardContentState` method. (For information about the rules for life cycle state transitions, see page 20.)



Rules for Changing Applet Instance Life Cycle States

Installation — The applet instance state is set to *installed* when you instantiate it by calling an `Install` command (APDU).

Selectability — In addition to using the `SetStatus` command (APDU), you can use the following APDU commands to make an applet instance *selectable*:

- Call an `Install: Install/Make Selectable` command to change the instance state to selectable at the same time you install it.
- Call an `Install: Make Selectable` command to change an installed applet instance's state to selectable.

Deletion — The current implementation of the Open Platform Cyberflex Access cards does not support logical deletion. To delete an applet instance, call a `Delete` command and delete the instance physically. (You can also call the `Delete` command through the Smart Card Toolkit interface, as described in the *Cyberflex Access Software Development Kit User's Guide*.)

Irreversible and Reversible Transitions — Some life cycle states cannot be reversed. Specifically, this means that once you make an applet instance selectable, you cannot return it to an installed state. Once you make an applet instance personalized, you cannot return it to a selectable (or installed) state. You can reverse transitions to the blocked or locked state, but only by returning the instance to its previous state.

Load File Life Cycle States and Transitions

A load file can have either of the following life cycle states:

- **Loaded** — If you add a load file to the card, it is in the life cycle state *loaded*. The load file AID is in the card registry, and the load file is detectable by an authenticated host-side application.
- **Deleted** — The current implementation of the Open Platform Cyberflex Access cards does not support logical deletion. To delete a load file, call a Delete command and delete the load file physically. (You can also call the Delete command through the Smart Card Toolkit interface, as described in the *Cyberflex Access Software Development Kit User's Guide*.)

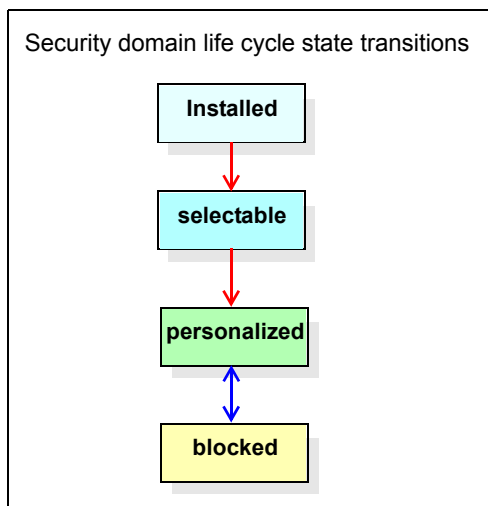
Security Domain Life Cycle States and Transitions

A security domain can have any of the following life cycle states:

- **Installed** — A security domain's state is set to *installed* during the successful execution of an Install command, which only the Card Manager can perform. The security domain's AID is entered in the card registry, and domain data is retrievable by a GetStatus command.
- **Selectable** — The *selectable* life cycle state enables a security domain to be selected, so it can receive incoming APDU commands. Optionally, the security domain can be made selectable as part of the installation process. The Card Manager handles changing the state transition to selectable.
- **Personalized** — Change a security domain's state to *personalized* when it has all the keys and data it needs to run the card programs that will be added in the security domain. You also typically update the card production life cycle (CPLC) data before you personalize a security domain.
- **Blocked** — You typically instruct the Card Manager to *block* a security domain when you want to deactivate it in response to a possible security threat.

Changing the Life Cycle State of a Security Domain with APDU Commands

The following illustration shows the security domain life cycle state changes you can make.



Rules for Changing Security Domain Life Cycle States

Installation — The security domain state becomes *installed* when you instantiate it by calling an `Install` command.

Selectability — You can make a security domain *selectable* by calling:

- `SetStatus` command
- `Install: Install/Make Selectable` command, in which you choose to change the security domain state to selectable during installation
- `Install: Make Selectable` command, in which you choose to change an installed security domain's state to selectable

Personalization — You can change a security domain's life cycle state from selectable to *personalized* by calling a `SetStatus` command, but only if the security domain has a valid key set loaded.

Blocking — You can *block* a security domain by calling a `SetStatus` command.

Irreversible and Reversible Transitions — Some life cycle states cannot be reversed. This means that once you make a security domain selectable, you cannot return it to an installed state. Once you make a security domain personalized, you cannot return it to a selectable or installed state. You can reverse a state transition from personalized to blocked, but only by returning the security domain to the personalized state.

Terminology

The following list contains a few useful terms related to Java Card programs in the context of Open Platform Cyberflex Access cards.

- **applet** — The word *applet* is included in several terms:
 - **applet instance** — An executable Java Card application you generate on the card from a load file.
 - **program file applet** — A component of a program file that corresponds to a class file incorporated into the program file. If a load file is created on a card from the program file and instantiated, the applet instance refers to a specific program file applet.
 - **card applet** — A generic term for Java Card program code or any of the subsequent forms the code can take.

- **application** — On an Open Platform Cyberflex Access card, an application is an agent on the card that can be selected to process commands: An applet instance, the Card Manager, or a security domain other than the Card Manager.
- **program file** — A converted applet (.ijc, or “interoperable javacard CAP”) file, which contains all of the classes and interfaces in a Java package. On a Cyberflex Access card, you create the program file by using the Program File Generator utility, as described on page 37. (A program file corresponds to the .bin file generated for Cyberflex Access 16K cards.)
- **Program File Generator** — The Cyberflex Access SDK utility that converts the class files in a Java package into a program file and an export file for the converted package. (The Program File Generator is discussed in greater detail on page 37.)
- **class file** — Depending on its context, the term class file can refer to:
 - A compiled Java Card source file, or
 - A component of a program file (derived from a compiled .class file).
- **load file** — The on-card form of an card program, which is downloaded to the card from a program file and which has the potential to create applet instances.
- **package** — Depending on its context, the term package can refer to:
 - The collection of class files used to create a program file, or
 - The program file derived from a package of class files.
- **program file** — Alternate term for a load file.

Other Card Elements

Global PIN

The card can have a global Personal Identification Number (PIN)—an alphanumeric string, which any application on the card can use to verify the identity of a user or card administrator. For example, the global PIN could provide access to a secure channel, which an off-card entity uses for protected commands, such as loading keys.

When you receive a new Open Platform Cyberflex Access card, it does not have a global PIN, but you can add one by calling a `PinChange` command. The card can have only one global PIN, but applet instances can have PINs of their own, which are not globally available on the card.

NOTE *If you personalize an Open Platform Cyberflex Access card in the COVE application, a PIN is included in the PKI applet that COVE adds to the card. This PIN is not a global PIN—it is available for use by the PKI applet and for secure logon with GINA.*

ALSO SEE

- *PinChange (page 113)*
- *Adding and Updating the Global PIN (page 52)*

Card Production Life Cycle (CPLC) Data

A new card comes with certain types of stored information known as card production life cycle (CPLC) data. CPLC data consists of microprocessor identification data stored in read-only memory (ROM) and card identification data stored in EEPROM. CPLC data includes details about the life cycle of the card during the production phases the card undergoes before it is shipped to you (fabrication, pre-personalization, and personalization).

You can use CPLC data to verify a card's authenticity or reliability by tracing the trail of processes the card has undergone—to determine which events occurred and when they occurred. CPLC data includes such information as the provider, release level, and release date of the card's operating system; the fabricator of the microprocessor and microprocessor module; the card manufacturer; and the card personalizer.

- ALSO SEE**
- *GetData (page 75)*
 - *PutData (page 116)*
 - *Retrieving Card Production Life Cycle Data (page 50)*

Card Issuer and BIN Data

You have the option to store, update, and retrieve card issuer data and card BIN data on the card by calling a command (as described on page 75: *GetData*, and page 116: *PutData*).

Card issuer data — Consists of user-defined content, which you can use for a variety of purposes—such as an identifier the issuer uses to derive static keys for the Card Manager.

Card BIN data — Typically contains the card issuer's serial number (the issuer's ISO 7812-defined identification number). You could use this data, for example, for associating the card with a specific card management system.

- ALSO SEE**
- *GetData (page 75)*
 - *PutData (page 116)*
 - *Retrieving Card Issuer Data or Card Issuer BIN Data (page 51)*



2

Guide to Using the Card Commands

This section describes how to perform the following tasks:

- Personalize a card (overview) — next topic

Secure channels

- Establish a secure channel — page 29
- Terminate a secure channel — page 30
- Set the security level for a secure channel — page 31

Keys

- Add a key set to the card — page 33
- Retrieve the version number and key set index of the key set used to generate the session keys — page 52
- Change the value of keys or key set version numbers — page 35

Applications

- Create an applet instance — page 36
- Delete a load or applet instance — page 39
- Select an application to receive commands — page 41
- Make an application selectable or selected by default — page 42

Application Life Cycle States

- Lock, unlock, or terminate a card — page 44
- Block or unblock a security domain — page 45
- Block, unblock, lock, or unlock an applet instance — page 46
- Retrieve the life cycle state of an application (the card, a security domain, or an applet instance) — page 47
- Modify the life cycle state of an application — page 48

Miscellaneous Card Data

- Retrieve card production life cycle data — page 50
- Add or update card production life cycle data — page 50
- Retrieve card BIN data or card issuer data — page 51
- Add or update card BIN data or card issuer data — page 51
- Update the AID value of the Card Manager or another security domain — page 51
- Add or update the global PIN — page 52
- Retrieve other data from the card — page 52

Encrypting and Decrypting Data

- Compute and include a MAC with a command — page 53
- Calculate and include encryption for a MAC+Enc command — page 57

Personalizing the Card

Roadmap for Personalizing a Card

To personalize a card, complete these general steps:

- 1** Establish a secure channel (as described in the following topic).
- 2** Initialize and populate one or more key sets on the card (page 31).
- 3** Add one or more applications to the card (page 36).
- 4** Update the card production life cycle data (page 50).
- 5** Change the card state to personalized (page 48).

Using Secure Channels

Establishing a Secure Channel

A secure channel session is divided into three sequential phases:

- Initiation (described in the following text and on page 5)
- Operation (page 30)
- Termination (page 30)

Initiating a Secure Channel

To establish a secure channel, the host-side application and currently selected security domain must authenticate each other—by using keys in a security domain key set and defining a security level for all the protected commands issued during the secure channel session.

Establishing a secure channel is a three step process:

- 1** Make sure the target security domain is selected (as described on page 41).

If the target security domain is selected by default and you have not attempted to select another application, you do not need to call a `SelectApplication` command explicitly.

- 2** Call an `InitializeUpdate` command (page 85), which initializes the secure channel. The `InitializeUpdate` command specifies a key to use for the authentication—from a key set in the current security domain. In the current implementation of the Open Platform Cyberflex Access cards, you can use the MAC key either in the default key set or in an explicitly specified key set.

(For details about the `InitializeUpdate` command, see page 85. For information about the session keys the card generates in an `InitializeUpdate` command, see page 10.)

- 3** Call an `ExternalAuthenticate` command (page 70) immediately after the `InitializeUpdate` command. This action authenticates the host-side application and specifies the security level required for protected commands sent to the card over the secure channel.

Using a Secure Channel

The host-side application uses the secure channel to send commands and data to the security domain. Most commands are available only when a secure channel is active, but there are a few exceptions:

- `InitializeUpdate` command — Available at any time
- `SelectApplication` command — Available at any time
- `GetData` command— Available either in or outside of a secure channel

Terminating a Secure Channel

Once you establish a secure channel, it remains open until you terminate it by selecting an application, initiating a new secure channel, or when one of the events listed on page 6 occurs.

Determining the Security Level of a Secure Channel

To determine the security level for a secure channel, use the `org.GlobalPlatform.SecureChannel.getSecurityLevel` method.

Working with Keys and Key Sets

Setting the Security Level for a Secure Channel Session

When you finish establishing a secure channel by executing an `ExternalAuthenticate` command, you specify a security level. The security level defines which, if any, cryptographic operations are required to verify the protected commands you send to the card.

A secure channel, as it is currently implemented in the Open Platform Cyberflex Access cards, has three possible security levels, described in the following table.

Security Level	Protection Provided	Mechanism
<i>Clear</i>	None	All commands and their input data (if any) are sent to the card in cleartext.
<i>MAC</i>	Command and command sequence integrity	A protected command sent to the card is signed with an ICV-chained MAC.
<i>MAC+Enc</i>	Command integrity and confidentiality, command sequence integrity	A protected command sent to the card has encrypted input data. The command header and input data are signed with an ICV-chained MAC.

The card enforces the security level during card communication.

NOTE *For more information about security levels, see page 7.*

Data Authentication Patterns Supported

Commands are verified by the use of a data authentication pattern (DAP). The Open Platform Cyberflex Access cards support these DAP types:

- MAC — 3DES cipher block chaining (CBC) mode used with the session MAC key (K_{mac}) to:
 - Establish a secure channel (`InitializeUpdate` and `ExternalAuthenticate` commands).
 - Sign protected commands transmitted over a MAC or MAC+Enc secure channel.
- MAC+Enc — 3DES encryption in CBC mode used with the session MAC key (K_{mac}) and the session AUTH key ($K_{\text{enc auth}}$) to:
 - Encrypt command data transmitted over a MAC+Enc secure channel.
 - Double-encrypt key or PIN data with the static KEK key, K_{KEK} . (You can double-encrypt key data sent in an `PutKey` command and PIN data sent in an `PinChange` command.)
- SHA-1 hashes — Used to hash load file blocks added to the card (`Load` command).

- ALSO SEE**
- *MAC verification* — page 53
 - *MAC+Enc verification* — page 57

Protected Commands

The following table shows the card commands that are protected (require DAP verification for security), and shows whether the command is available only in the Card Manager or is also available in subsidiary security domains.

Command	Security	Availability
Delete	<i>required</i>	Card Manager
ExternalAuthenticate	<i>required</i>	Card Manager / other security domain
GetData	<i>optional</i>	Card Manager / other security domain
GetStatus	<i>required</i>	Card Manager / other security domain
InitializeUpdate	<i>none</i>	Card Manager / other security domain
Install	<i>required</i>	Card Manager
Load	<i>required</i>	Card Manager
PinChange	<i>required</i>	Card Manager / other security domain
PutData	<i>required</i>	Card Manager / other security domain
PutKey	<i>required</i>	Card Manager / other security domain
SelectApplication	<i>none</i>	Card Manager / other security domain
SetStatus	<i>required</i>	Card Manager / other security domain

Adding a Key Set to the Card

A new Open Platform Cyberflex Access card contains a single default key set for the Card Manager (as described on page 9). You use this key set the first time you establish a secure channel with the card. You will continue to use the default key set to establish secure channels unless you load another key set and explicitly select it when you initiate a secure channel (in the InitiateUpdate command).

You can add key sets to the card for the Card Manager or for other security domains. In the current implementation of the Open Platform Cyberflex Access card, the Card Manager can have a maximum of eight key sets.

Reasons for Adding Key Sets

Card Manager — You typically put the card in a secured state only after you have added the key sets and security elements needed to enforce the card issuer's security policies.

Subsidiary Security Domains — If you add another security domains to the card, you typically load a domain-specific key set for the new security domain, which you use to establish a secure channel. This enables you to add applications and data to the card so that they are protected from manipulation outside their own security domain.

Until you add a domain-specific key set, you cannot personalize or block a security domain.

If you add a new security domain and default key set to the card, you use a Card Manager key set for the loading operation.

ALSO SEE • *Default Key Set* — page 9

Command for Adding a Key Set

To add key sets or update key set values, call a PutKey command (as described on page 120). You can make these types of changes:

- Add a new key set
- Update part or all of the key values in a key set
- Update the key set version number



If you add a key set or change key set values, keep records about the changes you make. The GlobalPlatform specification does not currently support tracking this type of data, so you must maintain accurate records.

Retrieving Key Set Data

The only key set data you can retrieve from the card with an APDU command is the key set information the InitializeUpdate command returns (as described on page 85). This information consists of the key set version used to generate the session keys and the key index of $K_{ENC, AUTH}$.

Changing Key Values and Key Set Version Numbers

You can also update the key values or the version number for key sets. You should always update the key values in the Card Manager before you personalize or secure the Card Manager.

You cannot change which key set is designated as the default key set, although you can change the default key set's version number and key values. The first key set loaded in any security domain becomes its default key set for the life of the security domain.

Ways of Updating a Key Set

To change key set values, use one of these methods:

- **PutKey command** — Use this command (as described on page 120) to update the values of keys or change the version number for a key set.
- **Smart Card Toolkit** — In the Card Manager pane of the Smart Card Toolkit window, right-click the Card Domain icon and select the **Change Keys** option from the pop-up menu, as shown in the following example.



Use the dialog box that appears to change key values in the default Card Manager key set. You cannot use this dialog box to create key sets or change the version number of key sets. *(For more information, see the Cyberflex Access Software Development Kit User's Guide.)*



If you change a key set, keep records about the changes. The GlobalPlatform specification does not currently support tracking key set contents and version numbers. You cannot retrieve key set data from the card, so you must maintain accurate records.

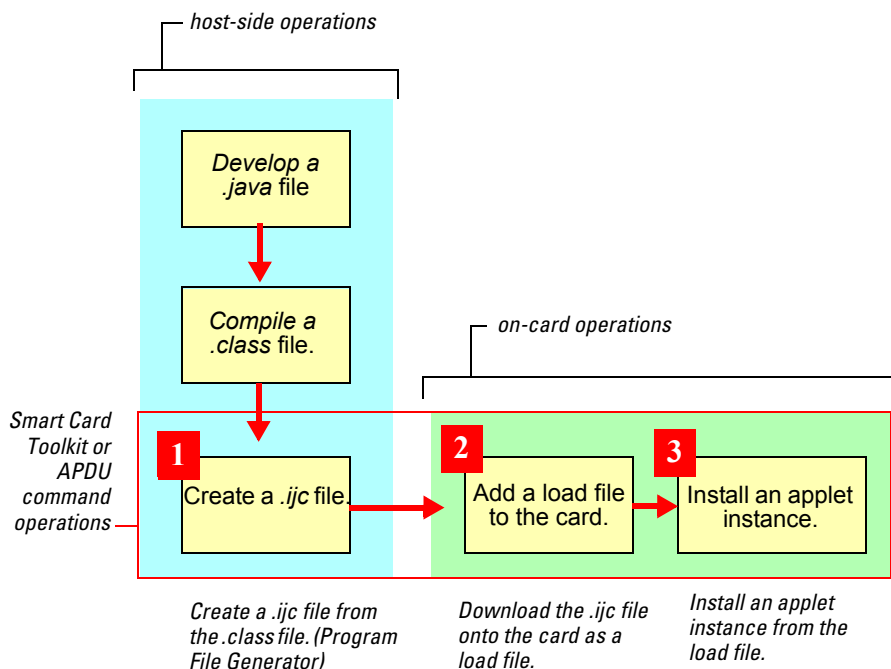
Adding and Working with Card Applications

Creating an Applet Instance

Java Card programs must go through preparatory steps before they can run on an Open Platform Cyberflex Access card. To create an applet instance on a card, compile class files from your source code (as discussed on page 160), then complete these tasks:

- 1** Create a program file from the class file (as described in the following text and on page 160).
- 2** Add a load file (or program file) to the card (as described on page 38 and in more detail on page 166).
- 3** Create applet instances on the card from the load file (as described on page 38 and in more detail on page 168).

The following illustration shows these three steps.



Step 1: Creating a program file

Once you have compiled a package of one or more class files for a card program, create a program file from the class files. Use the Program File Generator utility for the conversion.

Start the Program File Generator from the Smart Card Toolkit (as described on page 160). The conversion process involves three utilities, which the Cyberflex Access SDK installation program adds to your host system:

- Program File Generator — Cyberflex Access conversion utility designed to create program files capable of being downloaded to Open Platform Cyberflex Access cards.
- Sun Converter (Java Card 2.2 Class File Converter, version 1.3) — Utility from Sun Microsystems, which the Program File Generator calls to produce a *.jar* file from the package of class files you specified.
- MakeIJC utility (version 1.5) — A utility the Program File Generator calls to convert the *.jar* file into a *.ijc* file— a file that is formatted so the card can interpret it.

NOTE *The conversion file you use as the source for the load file must be created with utilities that are compatible with the target smart card. This document discusses the conversion process for a Java Card program that will be added to a Cyberflex Access Developer 32K card or a Cyberflex Access e-gate 32K card. For information about creating a conversion file that is compatible with a Cyberflex Access 16K card, see the topic titled “Converting a Program to a Cardlet,” in the Cyberflex Access Card Programmer’s Guide, which was released with the Cyberflex Access SDK 3C. The programmer’s guide for the Cyberflex Access 16K card is available at the Schlumberger Cyberflex Access support web page: www.cyberflex.com/Support/support.html.)*

Step 2: Downloading the program file data to the card as a load file

The next step is to add a load file to the card. The load file is a stream of bytecodes derived from the program file. The load file is the manifestation of the program code that is capable of being instantiated. It is also sometimes referred to as a program file.

Download a load file to the card in either of these ways:

- Use the Create Program dialog box in the Smart Card Toolkit (page 166), or
- Call the APDU commands directly: Select the program file you want to download by calling an `Install: Load` command (page 93), then call a series of `Load` commands (page 107).

Step 3: Instantiate the load file

Install an applet instance on the card from the load file in either of these ways:

- Use the Smart Card Toolkit's Create Instance dialog box (page 168), or
- Call the `Install` command directly (page 93).
 - To install the applet instance with a life cycle state of *installed*, use an `Install: Install` command.
 - To install the applet instance with a life cycle state of *selectable*, use an `Install: Install/Make Selectable` command. (You can also use this command mode to install an applet instance that has a life cycle state of *selectable* and has the default selected application privilege enabled.)

Specifying AID Values

When you create a program file, you specify a unique AID for it (called the package AID), and you specify a unique AID for each applet in the program file (with each program file applet corresponding to a class file added to the program file). These AIDs are unique—that is, no applet AID can have the same value as another applet AID or package AID.

These two types of program file AIDs are used in other elements created subsequently on the card:

- If you create a load file from a program file, its AID matches the package AID specified for the program file.
- If you create an applet instance from a program file applet (class in a program file), the default AID value for the applet instance is the AID value of the program file applet.

Deleting Load Files and Applet Instances

Rules for Deletions

Load File – To delete a load file, you must first delete any objects that refer to it or are derived from it. The Card Manager cannot delete a load file that is referenced by another load file or that was used to create an instance that remains on the card.

Applet Instance – If you delete an applet instance, you delete only the instance and its associated objects. The applet code and package are not affected.

Ways to Delete Load Files and Applet Instances

To delete a load file or applet instance, use one of these methods:

- Delete command (page 66) — To delete a single load file (package) or applet instance from the card.
- Smart Card Toolkit — In the Card Manager pane of the Smart Card Toolkit window, right-click the name of the load file or applet instance you want to delete, and select the **Delete** option from the pop-up menu as shown in the following example.



*For more information, see the *Cyberflex Access Software Development Kit User's Guide*.*

Selecting an Application

To process incoming APDU commands, an application must first be selected. On a new Open Platform Cyberflex Access card, the Card Manager is the default selected application. This means that the Card Manager is selected implicitly when a card session begins. You can also explicitly select an application (a security domain or applet instance).

Command for Selecting an Application

To select an application explicitly, use one of these methods:

- Call a `SelectApplication` command (as described on page 127).
- Smart Card Toolkit — Select an applet instance by using the Card Manager pane in the Smart Card Toolkit window. Right-click the application's icon, and select the option **Select Application** from the pop-up menu, as shown in the example below.



If a secure channel is currently established when you call a `SelectApplication` command, the secure channel session is terminated, even if the selection fails.

Rules for Selecting an Application

Security Requirement — Most commands are available only when a secure channel is active, but the `SelectApplication` command is available at any time.

Life Cycle State of the Target Application — The application you select must be in one of these life cycle states:

- Selectable
- Personalized
- Blocked

You cannot select an application whose state is installed, locked, or deleted. (The life cycle states of applet instances are described on page 18. The life cycle states of security domains are described on page 21.)

Card State — You can select an applet instance or security domain other than the Card Manager only if the card is not locked or terminated. If the card is locked, the only application you can select is the Card Manager. If the card is terminated, you cannot select any application.

When to Select an Application

Explicitly select an application when you want to:

- End the current secure channel session and establish a secure channel with a different security domain or with a different security level
- Send commands to an application that is not currently selected

Making an Application Selectable or Selected By Default

Making an Application's Life Cycle State Selectable

You can make an application's life cycle state selectable in one of these ways:

- **SetStatus** command (page 133) — Change the life cycle state of an application from *installed* to *selectable* or to reverse blocking or locking an application, so that it returns to its former state of selectable.
- **Install: Make Selectable** command (page 93) — Change the life cycle state of an application from *installed* to *selectable*.
- **Install: Install/Make Selectable** command (page 93) — Make the application's life cycle state selectable during installation.

- Smart Card Toolkit — Install an applet instance by using the Create Instance dialog box in the Smart Card Toolkit. Display this dialog box from the Card Manager pane of the main window, by right-clicking the program file you want to instantiate and using the pop-up menu as shown in the following example.



When you create an applet instance in the Smart Card Toolkit, the Install: Install/Make Selectable command is called and the applet instance's life cycle is specified as selectable. *(For more information, see the Cyberflex Access Software Development Kit User's Guide.)*

Specifying an Application to Be Selected by Default

You can make an application other than the Card Manager selected by default in one of these ways:

- Install: Install/Make Selectable command (page 93) — Make the application's life cycle state selectable and set the application to be selected by default during installation.
- Install: Make Selectable command (page 93) — Change the bit setting in the application privileges byte that determines whether an application is selected by default. You can make the application selected by default or toggle it back to the selectable state.

Retrieving an Application's Life Cycle State or Default Selection Status

You can use a `GetStatus` command (page 80) to retrieve:

- The value of an application's application privileges byte
- The application's life cycle state

Blocking, Locking, and Terminating Card Elements and Cards

Locking, Unlocking, and Terminating a Card

Allowable Life Cycle State Transitions

You can lock or terminate the Card Manager in the following ways (as illustrated on page 17):

- Initialized → Terminated
- Secured → Locked or terminated
- Locked → Terminated

You can **unlock** the Card Manager in the following way: Locked → Secured (if the Card Manager's life cycle state was secured immediately before it was locked)

Ways of Locking, Unlocking, and Terminating the Card Manager

To lock the Card Manager, use one of these methods:

- Call a `SetStatus` command (page 80).
- Call the `org.GlobalPlatform.System.lockCard` method.

An application can lock the Card Manager, but only if it has the appropriate privilege set in its application privileges byte (as described on page 11 and in the `Install` command description on page 93).

To unlock the Card Manager, call a `SetStatus` command (page 80). The Card Manager returns to the secured life cycle state.

To terminate the Card Manager, use one of these methods:

- Call a `SetStatus` command (page 80).
- Call the `org.GlobalPlatform.System.terminateCard` method.

An application can terminate the card, but only if it has the appropriate privilege set in its application privileges byte (as described on page 11 and in the `Install` command description on page 93).

Blocking and Unblocking Security Domains

The rules and methods for blocking and unblocking subsidiary security domains are different from those for blocking and unblocking the Card Manager.

Allowable Life Cycle State Transitions

- **You can block** a security domain in the following way (as illustrated on page 22): Personalized → Blocked
- **You can unblock** a security domain in the following way: Blocked → Personalized (if the security domain's life cycle state was personalized immediately before it was blocked)

Ways of Blocking and Unblocking Security Domains

To block a security domain, use one of these methods:

- Send a `SetStatus` command to the Card Manager (as described on page 80).
- Call the `org.GlobalPlatform.System.setCardContentState` method.

To unblock an applet instance, send a `SetStatus` command to the Card Manager (as described on page 80).

Blocking, Locking, Unblocking, and Unlocking Applet Instances

Allowable Life Cycle State Transitions

You can block or lock an applet instance in the following ways (as illustrated on page 19):

- Installed → Locked
- Selectable → Locked or blocked
- Personalized → Locked or blocked
- Blocked → Locked

You can unblock or unlock an applet instance in the following ways:

- Locked → Selectable, personalized, or blocked (if the transition is a reversal to the instance's life cycle state immediately before it was locked)
- Blocked → Selectable, personalized, or locked (if the transition is a reversal to the instance's life cycle state immediately before it was blocked)

Ways of Blocking, Locking, Unblocking, and Unlocking Applet Instances

To block an applet instance, use one of these methods:

- Send a `SetStatus` command to the Card Manager or other currently selected security domain that contains the applet instance (as described on page 80).
- Call the `org.GlobalPlatform.System.setCardContentState` method.

To unblock an applet instance, send a `SetStatus` command to the Card Manager or other currently selected security domain that contains the applet instance (as described on page 80).

To lock or unlock an applet instance, send a `SetStatus` command to the Card Manager or other currently selected security domain that contains the applet instance (as described on page 80).

Working with Card Status and Other Data

This topic describes types of data you retrieve, add, and update on a Cyberflex Access Developer 32K card. The subtopics are:

- Retrieving Life Cycle State Data (page 47)
- Changing the Life Cycle State of an Application (page 48)
- Retrieving Card Production Life Cycle Data (page 50)
- Adding and Updating Card Production Life Cycle Data (page 50)
- Retrieving Card Issuer Data or Card Issuer BIN Data (page 51)
- Adding and Updating Card Issuer Data and Card Issuer BIN (page 51)
- Updating AID Values (page 51)
- Adding and Updating the Global PIN (page 52)
- Other Data You Can Retrieve from the Card (page 52)

Retrieving Life Cycle State Data

To retrieve life cycle state data, call a `GetStatus` command (as described on page 80). You can retrieve these types of data:

- Load files — Life cycle state
- Card Manager — Life cycle state
- Subsidiary security domains — Life cycle state and application privileges data
- Applet instances — Life cycle state and application privileges data

Changing the Life Cycle State of an Application

To modify the life cycle state of an application, call a `SetStatus` command (as described on page 133). The life cycle state changes you can make depend on whether the Card Manager or a subsidiary security domain is selected.

Card Manager Selected

You can change the life cycle state of the Card Manager or applet instance in these ways:

Irreversible Transitions

- Card Manager: *initialized* → *secured*
- Card Manager: *initialized* → *terminated*
- Card Manager: *secured* → *terminated*
- Card Manager: *locked* → *terminated*
- Applet instance: *installed* → *secured*
- Applet instance: *selectable* → *personalized*

Reversible Transitions

- Card Manager: *secured* ↔ *locked*
- Applet instance: *installed* ↔ *locked*
- Applet instance: *selectable* ↔ *blocked*
- Applet instance: *selectable* ↔ *locked*
- Applet instance: *personalized* ↔ *blocked*
- Applet instance: *personalized* ↔ *locked*
- Applet instance: *blocked* ↔ *locked*

Security Domain Selected

You can change the life cycle state of the Card Manager or security domain in these ways:

Irreversible Transitions

- Card Manager: *initialized* → *terminated* (for a security domain with card termination privilege)
- Card Manager: *secured* → *terminated* (for a security domain with card termination privilege)
- Card Manager: *locked* → *terminated* (for a security domain with card termination privilege)
- Security domain: *installed* → *selectable*
- Security domain: *selectable* → *personalized*

Reversible Transitions

- Card Manager: *secured* ↔ *locked* (for a security domain with card locking privilege)
- Security domain: *personalized* ↔ *blocked*

NOTE *A security domain can lock or terminate the card only if it has the appropriate application privilege enabled during installation. (For information about the application privileges byte, see page 11.)*

Retrieving Card Production Life Cycle Data

GetData	To retrieve card production life cycle (CPLC) data from the card, call a <code>GetData</code> command (as described on page 75). The data you can retrieve includes 45 bytes of CPLC data—microprocessor ROM data, EEPROM manufacturing data, EEPROM pre-personalization data, and EEPROM personalization data.
Select Application	If you call a <code>SelectApplication</code> command (as described on page 127), the response data includes six bytes of CPLC data: the card OS ID, release level, and release date.
Initialize Update	If you call an <code>InitializeUpdate</code> command (as described on page 85), the response data includes the microprocessor's fabrication date, serial number, and batch identifier. This is part of the CPLC data (and is identical to bytes 14–21 of the data returned by a <code>GetData</code> command).

Adding and Updating Card Production Life Cycle Data

A new card comes with card production life cycle (CPLC) data, which consists of microprocessor identification data stored in read-only memory (ROM) and card identification data stored in EEPROM. CPLC data includes details about the life cycle of the card during the production phases the card undergoes before it is shipped to you (fabrication, pre-personalization, and personalization).

You can use CPLC data to verify a card's authenticity or reliability by tracing the trail of processes the card has undergone—to determine which events occurred and when they occurred. CPLC data includes such information as the provider, release level, and release date of the card's operating system; the fabricator of the microprocessor and microprocessor module; the card manufacturer; and the card personaliser.

To add or update card production life cycle (CPLC) data on the card, call a `PutData` command (as described on page 116). The types of tagged data objects you can add or update includes:

- Card Manager AID
- Card production life cycle (CPLC) personalization data
- CPLC pre-personalization data

- Security domain AID (if the currently selected application is a security domain)

Retrieving Card Issuer Data or Card Issuer BIN Data

To retrieve card issuer data or issuer BIN data from the card, call a `GetData` command (as described on page 75). This type of data is retrievable only if the data has been added to the card by calling a `PutData` command. These types of data are globally available on the card, so you can store and retrieve the data without creating or gaining access to an applet instance.

Card issuer data — Consists of user-defined content, which is usable for a variety of purposes. For example, the data can serve as an identifier the issuer uses to derive static keys for the Card Manager. Of course, in this case you add the card issuer data before the Card Manager keys are loaded.

Card BIN data — Typically contains the card issuer's serial number (the issuer's ISO 7812-defined identification number). You could use this data, for example, for associating the card with a specific card management system.

Adding and Updating Card Issuer Data and Card Issuer BIN

To add or update card the issuer BIN or card issuer data, call a `PutData` command (as described on page 116). The actions you can perform include:

- Add or update the card issuer BIN, a data object up to 8 bytes long that you can use to identify a card or its key sets on the platform level (independent of applet instances)
- Add or update the card issuer data, another platform-level identification object up to 8 bytes long

Updating AID Values

To update the AID of the Card Manager or another security domain, call a `PutData` command (as described on page 116). (To update an AID, the target security domain must be selected.)

Adding and Updating the Global PIN

When you receive a new Cyberflex Access Developer 32K card, it does not have a global PIN, but you can add one. The card's global PIN is a personal identification number that any application on the card can use. For example, the global PIN could provide access to a secure channel, which an off-card entity uses for protected commands, such as loading keys.

To add or update the global PIN, call the `PinChange` command (as described on page 113), and perform one or more of these actions:

- Add a global PIN
- Update the global PIN value
- Unblock the global PIN and reset its try counter to the maximum limit

Other Data You Can Retrieve from the Card

When you select an application by calling a `SelectApplication` command, the card returns the selected security domain's AID, six bytes of CPLC data (as described on page 50), and the maximum length of load file blocks you can send to the card.

Encrypting and Decrypting Commands

MAC Verification

You calculate and include a Message Authentication Code (MAC) of any protected command you send to the card in a MAC secure channel session.

To calculate the MAC, use the MAC session key (K_{mac}) in the current secure channel's specified key set. (The K_{mac} is the second key in the key set, and is a 16-byte 3DES key.)

Overview of MAC Operations

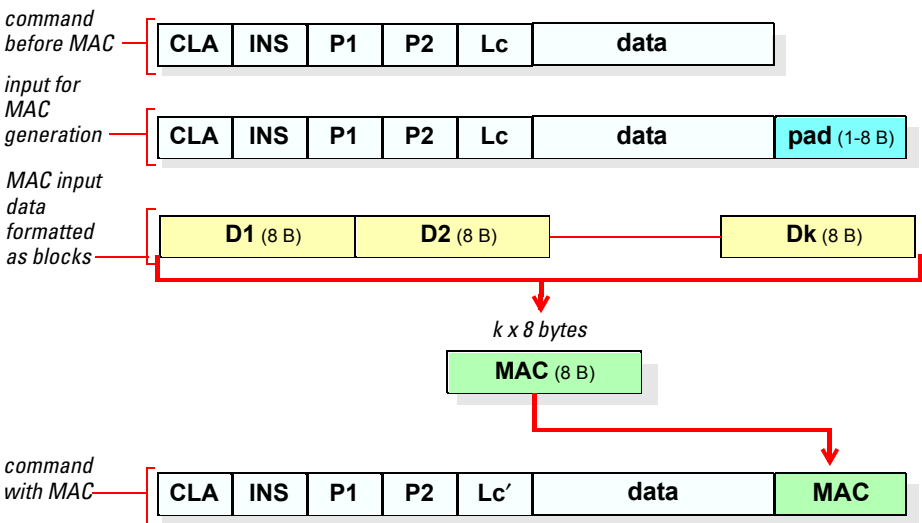
The following table shows the order of events for MAC secure channel operations:

Operation	Encryption Sequence	Command Components
<ul style="list-style-type: none"> Protected APDU command 	<i>MAC</i>	header + input data + MAC
<ul style="list-style-type: none"> Writing key or global PIN data (PutKey or PinChange) 	1) <i>MAC</i> 2) <i>key encryption</i>	header + KEK-encrypted input data + MAC
<ul style="list-style-type: none"> Writing load file (Load) 	1) <i>MAC</i> 2) <i>SHA-1</i>	header + SHA-1 hash of load file block + TLV-formatted MAC included with the first or last load file block

Format of a MAC Secure Channel Command

Compute the MAC on the entire APDU command—both its header and input data.

The following illustration shows the format of a command MAC operation.



LC = length of original data block to send to the card

LC' = length of command data + 8 bytes

NOTE Notice that although you add padding for MAC generation (as described in the next topic), you do not include the padding in the final command input data field.

Padding APDU Command Data for MAC Computation

As shown in the preceding illustration, the data sent to the card is formatted into 8-byte blocks. These rules apply for padding the command data:

- If the last data block is between 1 and 7 bytes long, you add padding after the data to make the block 8 bytes long. The first padding byte's value is 80h. If more padding bytes follow, each additional byte has a value of 00h.
- If the last data block is 8 bytes long, you add 8 bytes of padding with the values 80 00 00 00 00 00 00 00h after the command data.

The result is $[D_1 \parallel D_2 \parallel \dots \parallel D_k]$.

Note that the padding is not transmitted as part of the data block, but is discarded when MAC generation is complete.

Initialization Vector Chaining

You chain the command MACs in a secure channel session by using the MAC of one command as the initialization chaining vector (ICV) for the next MAC. Chaining ensures that the commands in a secure channel session are presented to the card in the order in which you send them.

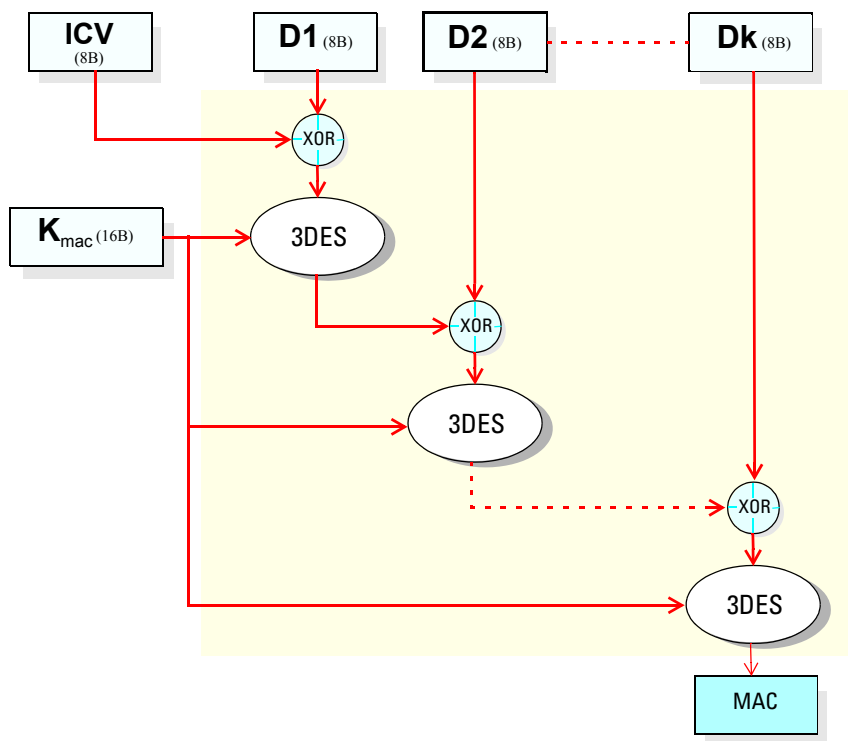
The initialization vector value for the first MAC (the one generated for the ExternalAuthenticate command) is an 8-byte string of null values. For all subsequent commands, the ICV for MAC generation is the MAC value from the previous command.

MAC Calculation

The formula for MAC generation is:

$$\text{MAC} = \text{generateMAC} (K_{\text{mac}}) [D_1 || D_2 || \dots || D_k]$$

The following illustration shows a command MAC calculation, which uses the 3DES MAC key in CBC mode.



where:

D1, D2 = First and second data blocks

Dk = Final data block

ICV = Initialization chaining vector (described on page 55)

MAC+Enc Verification

If the current secure channel has a security level that requires MAC+Enc, you must sign the command with a MAC and encrypt the command input data.

- *MAC* — Calculate the MAC as described in the previous topic (page 53).
- *Encryption* — Encrypt the input data by using the AUTH key in the current secure channel's specified key set. The AUTH key is a 16-byte 3DES key, and is the first key in the key set.

To encrypt command data for a MAC+Enc secure channel session, use the AUTH session key ($K_{enc, auth}$) in CBC mode to:

- Encrypt cleartext input data you send to the card as part of a protected command over a MAC+Enc secure channel.
- Encrypt the SHA-1 hash of load file blocks you include with Load commands.
- Double-encrypt key data or global PIN data you send to the card with a PutKey or PinChange command.

Overview of MAC+Enc Operations

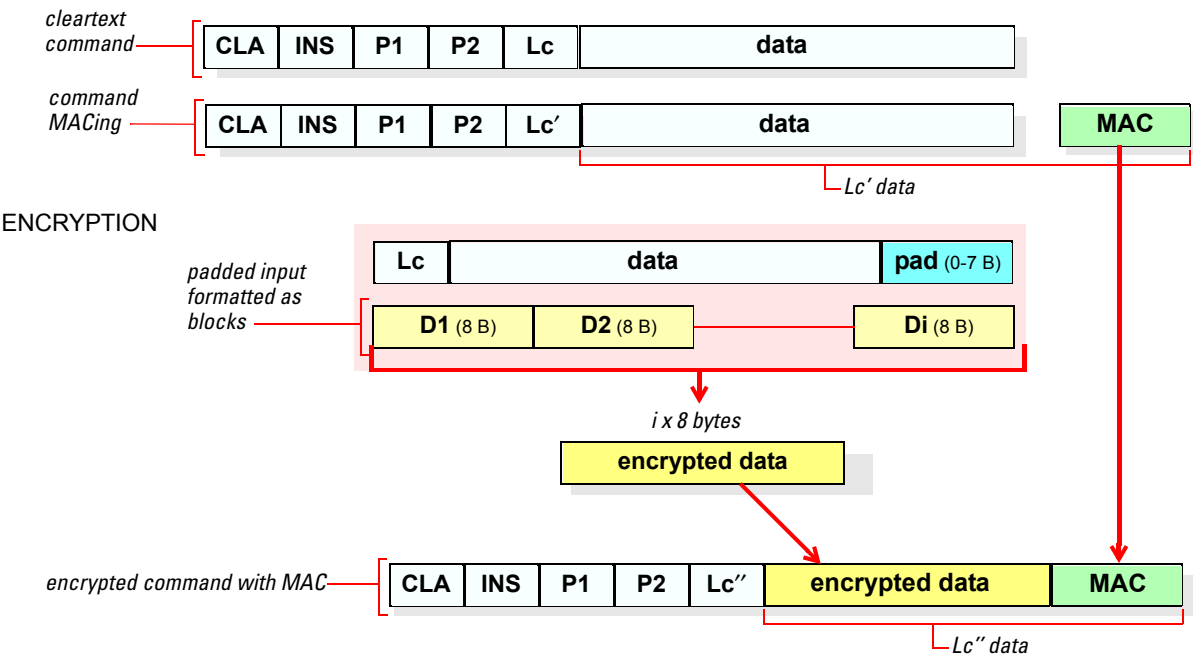
The following table shows the encryption sequence for MAC+Enc secure channel operations:

Operation	Encryption Sequence	Command Components
<ul style="list-style-type: none"> • Protected APDU command 	1) <i>MAC</i> 2) <i>command encryption</i>	header + encrypted input data + MAC
<ul style="list-style-type: none"> • Writing key or global PIN data (PutKey or PinChange) 	1) <i>MAC</i> 2) <i>Key Encryption</i> 3) <i>command encryption</i>	header + input data (encrypted with the KEK key, then with the AUTH key) + MAC.
<ul style="list-style-type: none"> • Writing load file (Load) 	1) <i>MAC</i> 2) <i>SHA-1</i> 3) <i>command encryption</i>	header + encrypted SHA-1 hash of load file block + TLV-formatted MAC included with first or last load file block

Format of a MAC+Enc Secure Channel Command

Encrypt only the APDU command header and input data—not the MAC signature. Compute the MAC first, then perform the encryption.

The following illustration shows the format of a command MAC+Enc operation.



LC = length of original data block to send to the card
LC' = length of command data + 8-byte MAC
LC'' = length of encrypted data (with padding) + 8-byte MAC + 1-byte *Lc* (length of encryption block)

NOTE Notice that (unlike the padding you add for MAC generation) the padding you add for encryption is included in the final command input data field.

Padding Encrypted Data

Before you perform encryption on the data block, pad the data. Unlike MAC padding, encryption padding becomes part of the command data, so you must modify the Lc value to reflect the padded length. Use the following process to pad encrypted data:

- 1 Add the length of the original cleartext input data to the beginning of the input data block. This length byte becomes part of the input data.
- 2 Determine whether the input data block that results is in octet format (is evenly divisible by 8).
If the resulting input data block is in octet format, skip the remaining padding steps. No further padding is needed.
- 3 If the resulting input data block is not in octet format, append a byte with a value of 80h to the end of the input data block (Lc + input data + 80h).
If the resulting input data block is in octet format, skip the remaining padding steps.
- 4 If the resulting input data block is not in octet format, add bytes with a value of 00h to the end of the input data block until the length is a multiple of 8 (Lc + input data + 80h + 00h ...).

Command Checking

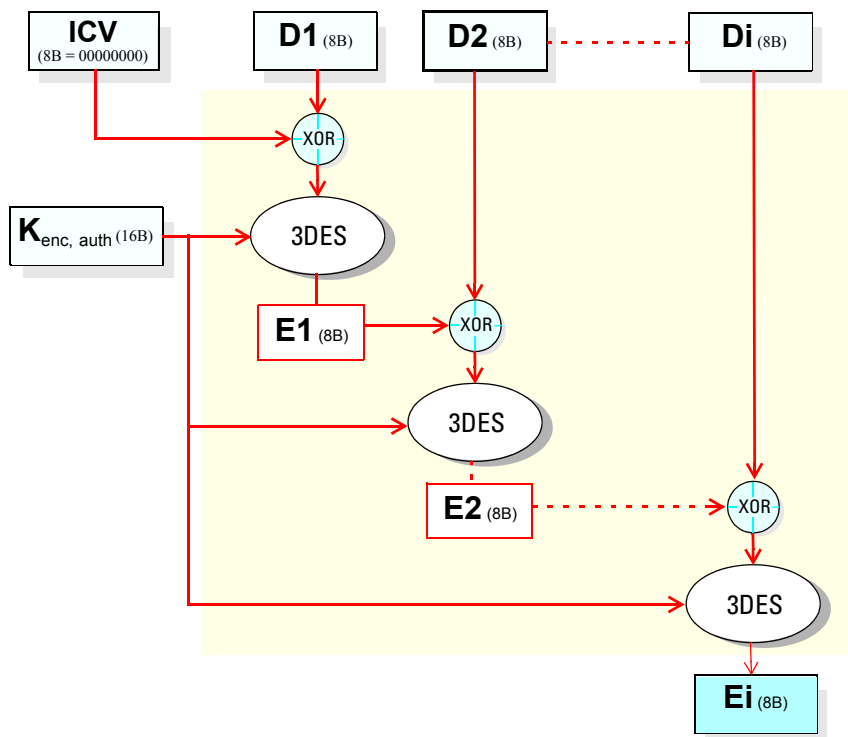
If the card receives data blocks that are encrypted and signed with a MAC, the card checks for the following conditions and returns an error if any one of the conditions is not met:

- Lc" value is a multiple of 8.
If the Lc" value fails the check, the card returns status words 67 00h (wrong length).
- The encryption padding is between 0 and 7 bytes long.
If the encryption padding length fails the check, the card returns status words 6A 80h (incorrect parameter in the data field).
- MAC is correct.
If the MAC fails the verification check, the card returns status words 69 82h (security status not satisfied), and terminates the secure channel.

MAC+Enc Calculation

Encrypt APDU data by using the 3DES $K_{enc, auth}$ key in CBC mode with the 8-byte ICV value set to null values.

The illustration on the next page shows a command encryption calculation.



where:

$D1, D2$ = First and second data blocks

Di = Final data block

$E1, E2$ = First and second encryption blocks

Ei = Final encryption block

ICV = Initialization chaining vector



Card Commands

Introduction

This section describes the Cyberflex Access card commands. The commands handle these main tasks:

- Manage card files, program files, and applet instances
- Conduct the transactions that enable the host application and terminal to gain access to card applications and files
- Handle utilities that manage Java programs on the card
- Provide basic cryptographic functionality

By default, the card calls these commands through the Card Manager application. The Card Manager acts as the card's default operating system — the command handler that runs initially whenever the card is powered up or reset.

If the card has a security domain other than the Card Manager, you can select that application to be the default command processor. In this case, the security domain processes the standard card command set, with these exceptions: Delete, Install, and Load. To execute these commands, the Card Manager must be selected.

The Card Manager and other security domains can pass custom commands to an applet instance, if one is installed and selected.

Getting Additional Information

The Cyberflex Access SDK installation program adds documentation to your system that describes the contents of the card's class libraries. The class libraries contain all the class files necessary for running the Cyberflex Access Developer 32K card's operating system — the General Purpose Operating System (GPOS). The libraries also contain other reference and constant files required by the GPOS and by ISO standards.

NOTE *For more information about the support Cyberflex Access Developer 32K card offers for Java card programs compliant with the Java Card 2.1.1 API specification, see page 177.*

Command Overview

The following table summarizes the commands available in the Card Manager application on a Cyberflex Access Developer 32K card. *(The numbers in the table are in hexadecimal format.)*

Command	CLA	INS	P1	P2	Lc	Le	Mode
Delete	80/84	E4	00	00	<i>lgth</i>	00	S
ExternalAuthenticate	84	82	00/01/03	00	10	—	S
GetData							
<i>CPLC data</i>	80/84	CA	9F	7F	00/08	2D	R or S/R
<i>card issuer BIN</i>	80/84	CA	00	42	00/08	03-0A	R or S/R
<i>card issuer data</i>	80/84	CA	00	45	00/08	0A	R or S/R
GetStatus							
<i>load file</i>	80/84	F2	20	00	00/08	<i>var</i>	S/R
<i>instance/sec. dom. (app)</i>	80/84	F2	40	00	00/08	<i>var</i>	S/R
<i>app., load file</i>	80/84	F2	60	00	00/08	<i>var</i>	S/R
<i>Card Manager (CM)</i>	80/84	F2	80	00	00/08	08-13	S/R
<i>CM, load file</i>	80/84	F2	A0	00	00/08	<i>var</i>	S/R
<i>CM, app</i>	80/84	F2	C0	00	00/08	<i>var</i>	S/R
<i>CM, app, load file</i>	80/84	F2	E0	00	00/08	<i>var</i>	S/R
InitializeUpdate	80	50	<i>keyset vers</i>	00/01	08	1C	S/R

Command	CLA	INS	P1	P2	Lc	Le	Mode
Install							
<i>load</i>	80/84	E6	02	00	<i>lgth</i>	00	S/R
<i>install</i>	80/84	E6	04	00	<i>lgth</i>	00	S/R
<i>make selectable</i>	80/84	E6	08	00	<i>lgth</i>	00	S/R
<i>install/make selectable</i>	80/84	E6	0C	00	<i>lgth</i>	00	S/R
Load							
<i>first/intermed. blocks</i>	80/84	E8	00	00-FF	<i>lgth (+)</i>	—	S
<i>final block</i>	80/84	E8	80	00-FF	<i>lgth (+)</i>	00	S/R
PinChange							
<i>Unblock (no MAC)</i>	80/84	24	00	00	00	—	—
<i>Unblock + MAC</i>	80/84	24	00	00	08	—	S
<i>Update/Load (no MAC)</i>	80/84	24	00	03-0F	08	—	S
<i>Update/Load + MAC</i>	80/84	24	00	03-0F	10	—	S
PutData, w/ CM selected:							
<i>Card issuer BIN</i>	80/84	CA	00	42	01-08+	01-08+	S
<i>Card issuer data</i>	80/84	CA	00	45	08/10	08+	S
<i>Card Manager AID</i>	80/84	CA	00	4F	05-10+	05-10+	S
<i>CPLC perso data</i>	80/84	CA	9F	66	08/10	08+	S
<i>CPLC pre-perso data</i>	80/84	CA	9F	67	08/10	08+	S
w/ sec. domain selected:							
<i>Sec. domain AID</i>	80/84	CA	00	4F	05-10+	01-08+	S
PutKey							
				<i>key index:</i>			
<i>add key set</i>	80/84	D8	00	81	43	0A	S/R
<i>update all 3 keys</i>	80/84	D8	<i>keyset vers</i>	01	43	0A	S/R
<i>update 2 keys (1-2 / 2-3)</i>	80/84	D8	<i>keyset vers</i>	01/02	2D	07	S/R
<i>update 1 key (1, 2, or 3)</i>	80/84	D8	<i>keyset vers</i>	01/02/03	17	04	S/R
SelectApplication ¹	00	A4	04	00/02	<i>AID lgth</i>	<i>var</i>	S/R
SetStatus							
						—	
<i>Card Manager</i>	80/84	F0	80	<i>status</i>	<i>lgth</i>		S
<i>security domain</i>	80/84	F0	40	<i>status</i>	<i>lgth</i>		S

¹ SelectApplication is the first command to be called in a card session, either implicitly or explicitly. SelectApplication is a JCRE command. All the other commands described here are Card Manager or security domain commands.

Use the Cyberflex Access card commands to perform these tasks:

- **Delete** – Delete an applet instance or a load file that is not referenced by any objects (applet instances or other load files) on the card.
- **ExternalAuthenticate** – Authenticate the host to the card and establish a secure channel with a specified security level. Follows a successful **InitializeUpdate** execution.
- **GetData** – Retrieve card issuer BIN, card issuer data, Card Manager or security domain AID, or card production life cycle data.
- **GetStatus** – Retrieve life cycle status and AID of load files and applications (applet instances, security domains, and the Card Manager). Also retrieve the application privileges settings for applet instances.
- **InitializeUpdate** – Begin card/host mutual authentication required for establishing a secure channel.
- **Install** – Perform any of the following actions:
 - *Load*: Prepare the card for a **Load** command to follow.
 - *Install*: Install an applet instance from a load file on the card, or install a security domain.
 - *Make Selectable*: *Installed application* – Make selectable or selected by default. *Selectable application* – Make selected by default. *Default selected application* – Remove the default selection privilege.
 - *Install/Make Selectable*: Install and simultaneously make an applet instance or security domain selectable or selected by default.
- **Load** – Add a load file block to the card. (Used in sequence.) The first **Load** command must follow an **Install: Load** command.
- **PinChange** – Add a global PIN to the card, unblock the global PIN, update the PIN value, or update and unblock the global PIN.
- **PutData** – Update the AID value of the Card Manager or another security domain. Also add or update the value of any of these data objects: card issuer BIN, card issuer data, card production life cycle (CPLC) personalization data, or CPLC pre-personalization data.
- **PutKey** – Add a key set to the card, or update existing keys' values and/or key set version number.

- `SelectApplication` – Select an application (the Card Manager, other security domain, or applet instance) or a file in an application's file system.
- `SetStatus` – Change the life cycle state of the Card Manager, a security domain, or an applet instance.

Overview of Status Words

Status words are 2-byte values the card returns in response to commands. The status words indicate whether the command succeeded or failed. For a command that failed, the status words typically provide information about the cause of the failure. Cyberflex Access Developer 32K card test cards return status words in ISO format.

NOTE *If you are working on a sizable project that requires cards that return TE9 status words instead of ISO status words, contact SchlumbergerSema at cyberflex@slb.com and ask for a product manager to work with you to arrange a special order.*

Delete

Use the `Delete` command to delete a single load file (package) or applet instance from the card.



Security requirement – A secure channel must be open, and you must satisfy the security level defined for it. (For information about establishing a secure channel, see page 5.)

Rules for Deletions

Applet Instance – If you delete an applet instance, you delete only the instance and its associated objects. The applet code and package are not affected.

Load File – To delete a load file, you must first delete any objects that refer to it or are derived from it. The Card Manager cannot delete a load file that is referenced by another load file or that was used to create an instance that remains on the card.

Card Manager – You cannot delete the Card Manager.

Recovery of EEPROM

If you delete a load file or applet instance, the card recovers the use of the EEPROM the object occupied.

If you delete an applet instance, the card also recovers the memory the instance required for transient object references, unless the card contains another instance that needs the same amount of object reference space. If less object reference space is needed, the card recovers the extra space.

Command Format To delete a load file or applet instance, issue a Delete command with the following APDU format.

CLA	INS	P1	P2	Lc	Input Data	Le	Mode
80/84	E4	00	00	<i>input lgth</i>	<i>TLV AID</i>	00	S

P1 Always set to 00h.

P2 Always set to 00h.

Lc Length of the input data: Number of bytes in hexadecimal format.

Input Data *TLV AID*: The AID for the object to be deleted, entered in TLV format as described below:

- *Tag* – (1 B) Set to 4Fh, the ISO-specified tag used to indicate that an application identifier, or AID, follows.
- *Length* – (1B) The AID's length (number of bytes, in hexadecimal format).
- *Value* – Byte stream – (5–16 B) The AID value.

Le 00h – No response data.



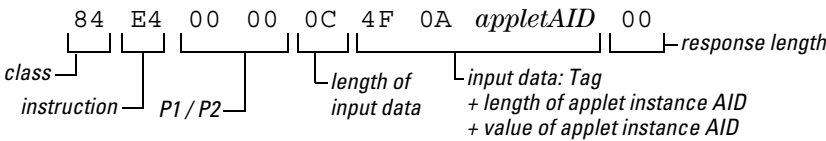
Open Platform Implementation — *The current release of the Cyberflex Access Developer 32K card implements the GlobalPlatform specification v2.0.1 for the Delete command with these options or limitations:*

- *The card does not support delegated management, and so does not return delegated management receipt data for the Delete command.*
- *You cannot delete multiple applet instances or load files with a single Delete command.*
- *The Delete command is available when a card is locked.*

Example

The following example Delete command asks the card to delete an applet instance. (The example does not include MAC or MAC+Enc verification data. For information about including MAC or MAC+Enc verification data, see page 14 and page 15.)

84 E4 00 00 0C 4F 0A *appletAID* 00



where:

- | | | |
|--------------------|---|---|
| 84 | = | CLA – Open Platform-compliant command that requires a secure channel. |
| E4 | = | INS – Instruction in the command set is Delete. |
| 00 | = | P1 – Always set to 00h. |
| 00 | = | P2 – Always set to 00h. |
| 0C | = | Lc – Number of bytes of input data that follow. |
| Input data: | | |
| 4F | = | • Tag that signals an applet instance AID follows. |
| 0A | = | • Length of the applet instance AID (05–10h) |
| <i>appletAID</i> | = | • Value of the applet instance AID (5–16 bytes) |
| 00 | = | Le – Length of the expected response data, none. |

Response Data

If the specified object is deleted successfully, the card returns the receipt length, a value of 00h. No delete receipt is available for return, since the Card Manager deletes the object directly instead of delegating the task. (If the command does not succeed, the card returns only status words.)

Status Words Returned

In addition to the receipt length, the card returns ISO status words to indicate that the command succeeded or failed, as described in the following table.

Status	Description
6400	Technical problem that has no specified diagnosis.
6581	Memory failure.
6700	The specified length of the input data (Lc) is incorrect.
6982	Security status not satisfied. For example, MAC verification failed.
6985	A requirement for using the command is not satisfied. For example, you issued the command outside of a secure channel or the card still contains some applet instances derived from the specified load file.
6A80	Incorrect values in input data. Examples: The length (<i>L</i>) and value (<i>V</i>) in the TLV-formatted input data do not match; the specified AID is invalid; or an incorrect number of padding bytes are included in the input data (if the command is encrypted).
6A86	Incorrect value specified for P1, P2, or both.
6A88	The Card Manager did not find the specified AID in the registry.
6D00	Unsupported value entered for the INS byte.
6E00	Unsupported value entered for the CLA byte.
9000	The command succeeded, and the specified object has been deleted.

Also See: SetStatus command on page 133

ExternalAuthenticate

Use the `ExternalAuthenticate` command to authenticate the host's identity to the card and establish a secure channel — a channel with a specified security level for passing commands to the card.



Security requirement – You must accompany an `ExternalAuthenticate` command with a MAC. (For information about MAC verification, see page 14.)

Previous command – You must execute a successful `InitializeUpdate` command immediately before you call an `ExternalAuthenticate` command.

Card state – You can issue an `ExternalAuthenticate` command to a card that is in any of these states: initialized, secured, or locked.

Mutual authentication must be completed to establish a secure channel between the host and card security domain. The host begins the process by sending the card an `InitializeUpdate` command, passing in a challenge (random number) and key identification data. The card returns its own challenge, cryptogram, and key identification information. The host uses its copy of the specified key to calculate a cryptogram from the original challenge, which it compares to the card cryptogram.

If the card authentication succeeds, the host follows with an `ExternalAuthenticate` command, passing in a host cryptogram and MAC. The P1 value specifies the security requirement that will apply to all the commands passed over the secure channel, if it is established successfully. The card generates a comparison cryptogram to authenticate the host.

If the verification succeeds, a secure channel is opened physically. If the mutual authentication fails for any reason, no secure channel opens. Begin again by executing the `InitializeUpdate` command.

**Command
Format**

Immediately after a successful `InitializeUpdate` command, call the `ExternalAuthenticate` command with the following APDU format.

CLA	INS	P1	P2	Lc	Input Data	Le	Mode
84	82	<i>sec level</i>	00	10	<i>cryptogram + MAC</i>	–	S

- P1** *sec level*: Security level required for all subsequent commands that are issued in the secure channel. P1 can have any of these values:
- 00h = No security (Use this setting only for a card that is not secured.)
 - 01h = MAC (Message Authentication Code, to verify integrity)
 - 03h = MAC+Enc (with encryption added to ensure confidentiality)
- P2** 00h = RFU
- Lc** 10h: Length of the input data, 16 bytes.
- Input Data** A 16-byte block that consists of the host cryptogram and command MAC, formatted as described in the table that follows.
- Le** None – The card returns no response data.

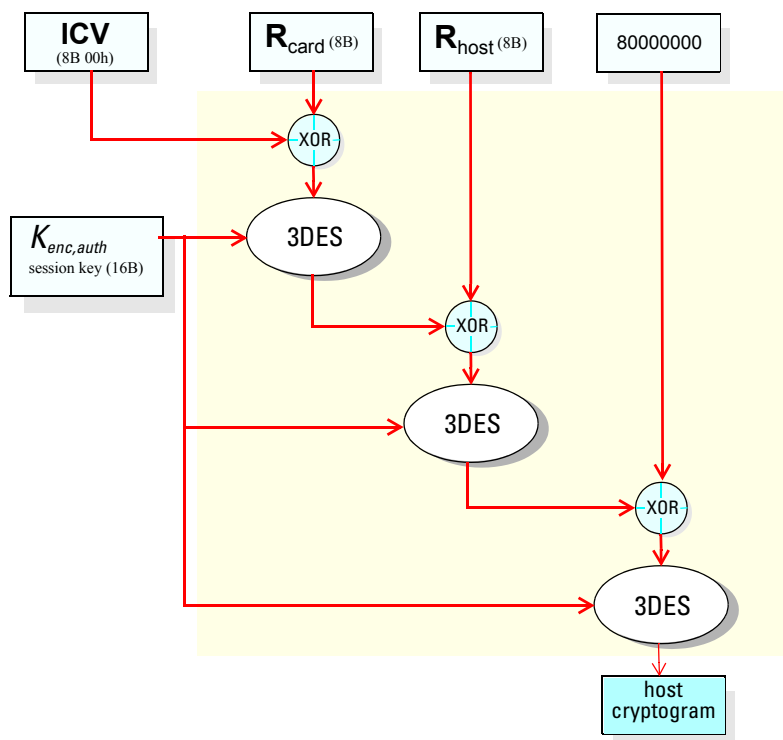
Input Data

Bytes	Description	Length
1–8	Host cryptogram, concatenated from the 8-byte card challenge and 8-byte host challenge and padded for MAC, as shown in the following illustration.	8 B
9–16	MAC, which you generate by using triple DES in CBC mode with 00h as the value for all ICV bytes. <i>Note: If the security level set for the secure channel requires MAC, the next command uses the MAC generated for the ExternalAuthenticate command as its ICV. From this point on, the ICV for each command is the MAC value generated by the previous command.</i>	8 B

Host Cryptogram Calculation

The host cryptogram (MAC generated) is calculated as follows:

$(K_{enc,auth}, ICV = 8 \text{ bytes of } 00h) [R_{host} \parallel R_{card}]$



Calculation for the Host Cryptogram in an ExternalAuthenticate Command

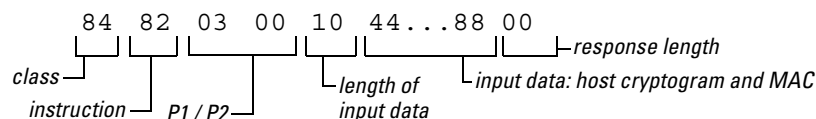
Pad the input data with an 8-byte block (80 00 00 00 00 00 00 00h) to indicate that a MAC is included in the input data.

NOTE For more information about including a MAC with a command, see page 14.

Example

The following example APDU data string sends the card an ExternalAuthenticate command.

84 82 03 00 10 44 44 44 44 44 44 44 44 11 22 33 44 55 66 77 88 00



where:

- 84 = **CLA** – Open Platform-compliant command that requires a secure channel.
- 82 = **INS** – Instruction in the command set is ExternalAuthenticate.
- 03 = **P1** – Security level required for all commands issued in the secure channel established at the successful completion of this command is MAC+Enc (MAC with encryption added).
- 00 = **P2** – RFU
- 10 = **Lc** – Length of input data that follows, 16 bytes.
- 44...88 = **Input data** – Placeholder for the host cryptogram and MAC. The first 8 bytes are the host cryptogram, concatenated from the 8-byte card challenge and 8-byte host challenge, with padding for MAC. The final 8 bytes are the encrypted MAC of the command.
- 00 = **Le** – Length of the expected response data, none.


**Status Words
Returned**

Status	Description
6300	Authentication of host cryptogram failed.
6400	Technical problem that has no specified diagnosis.
6700	The specified Lc value is incorrect. Enter 10h.
6982	Security status not satisfied: MAC verification failed. (For example, $K_{enc, aut}$ is incorrect.)
6985	A requirement for using the command is not satisfied. For example, a successful <code>InitializeUpdate</code> command did not immediately precede the command, or the P1 value is set to 00h (no security) on a card that is in a secured or locked state.
6A80	The lengths of the encrypted message and cleartext are inconsistent. This can occur if the MAC is computed with the wrong key, for example.
6A86	Incorrect value specified for P1, P2, or both.
6D00	Unsupported value entered for the INS byte.
6E00	Unsupported value entered for the CLA byte.
9000	The command succeeded and a secure channel is established.

Also See: `InitializeUpdate` on page 85
 “Key Diversification,” on page 87

GetData

Use the `GetData` command to retrieve a single, specified data object. You can retrieve card production life cycle (CPLC) data, CPLC pre-personalization data, the card issuer BIN, card issuer data, or the AID of the Card Manager or other currently selected security domain. (You can retrieve any Card Manager data added to the card with a `PutData` command.)

 **Security requirement** – You can call the `GetData` command either within or outside of a secure channel. If you call the `GetData` command within a secure channel, you must satisfy the security level defined for the secure channel. (For information about establishing a secure channel, see page 5.)

Command Format Call a `GetData` command with the following APDU format.

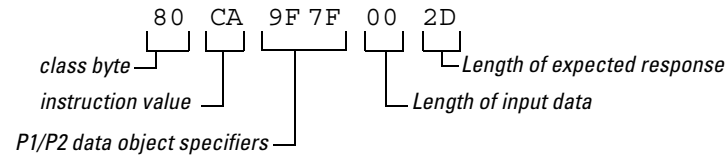
CLA	INS	P1	P2	Lc	Input Data	Le	Mode
80/84	CA	<i>data type</i>	<i>data descr</i>	00/08	—/MAC	<i>ret lgth</i>	R or S/R

- P1 / P2** *data type / data descr* – P1 and P2 together specify the data object tag. You can use any of the following P1 / P2 combinations:
- Card production life cycle (CPLC) data = 9F 7Fh
 - Issuer BIN (Card Manager only) = 00 42h
 - Card issuer data (Card Manager only) = 00 45h
- Lc** *input data length* – The Lc value is either:
- 00h = If you issue the command outside of a secure channel
 - 08h = If you issue the command in a secure channel
- Input Data** Included only if the `GetData` command is issued in a secure channel. In this case the input data is an 8-byte MAC of the command, calculated as described on page 14.
- Le** *ret lgth* – The length of the response data varies with data type:
- Card production life cycle (CPLC) data = 2Dh (45 bytes)
 - Issuer BIN (Card Manager only) = 03–0Ah (3–10 bytes)
 - Card issuer data (Card Manager only) = 0Ah (10 bytes)

Example

The following example GetData command asks the card to return card production life cycle data.

80 CA 9F 7F 00 2D



where:

- 80 = **CLA** – Standard ISO-compliant command.
- CA = **INS** – Instruction in the command set is GetData.
- 9F 7F = **P1/ P2** – Return card production life cycle data.
- 00 = **Lc** – Length of input data that follows is 0 bytes.
- 2D = **Le** – Length of the expected response data is 45 bytes.

CPLC Response Data

If you enter 9F 7Fh for P1 and P2, the card returns the following 45 bytes of CPLC data.

Byte(s)	Description / Value	Length
1–2	CPLC tag = 9F 7Fh	2 B
3	Length = 2Ah	1 B
4–5	Integrated circuit ROM fabricator = 40 90h	2 B
6–7	Integrated circuit ROM type = 00 9Ch	2 B
8–9	ROM OS identifier = 00 02h	2 B
10–11	ROM OS release date = 07 00h	2 B
12–13	ROM OS release level = 01 00h	2 B
14–15	EEPROM fabrication date	2 B
16–19	EEPROM serial number	4 B
20–21	EEPROM batch identifier	2 B
22–23	EEPROM module fabricator	2 B
24–25	EEPROM module packaging date	2 B
26–27	EEPROM integrated circuit card manufacturer	2 B
28–29	EEPROM embedding date	2 B
30–31	EEPROM pre-personalizer	2 B
32–33	EEPROM pre-personalization date	2 B
34–37	EEPROM pre-personalization equipment identifier	4 B
38–39	EEPROM personalizer	2 B
40–41	EEPROM personalization date	2 B
42–45	EEPROM personalization equipment identifier	4 B

Issuer BIN Response Data

If you enter 00 42h for P1 and P2, the card returns the following 3–10 bytes of card issuer BIN data. (If no card issuer BIN exists on the card, the card returns only the status words 6A86h.)

Byte(s)	Description Value	Length
1	Issuer BIN tag = 42h	1 B
2	Length = 01–08h	1 B
3–x	Issuer BIN (value specified in PutData command)	1–8 B

Card Issuer Response Data

If you enter 00 45h for P1 and P2, the card returns the following 10 bytes of card issuer data. (If no card issuer data exists on the card, the card returns only the status words 6A86h.)

Byte(s)	Description Value	Length
1	Card issuer data tag = 45h	1 B
2	Length = 08h	1 B
3–10	Card issuer data (value specified in PutData command)	8 B

NOTE *The format of the response data for a GetData command has changed for the Cyberflex Access Developer 32K card. If you are using an older smart card, refer to the card's documentation.*

**Status Words
Returned**

The card returns status words to indicate the success or failure of the GetData command, as described in the table that follows.

Status	Description
6400	Technical problem that has no specified diagnosis.
6581	Memory failure.
6700	The specified length of the input data (Lc) is incorrect.
6982	Security status not satisfied. For example, MAC verification failed.
6985	A requirement for using the command is not satisfied. For example, you issued the command outside of a secure channel in a security domain that requires secure data transmission.
6A80	Incorrect values in input data. Examples: The length (<i>L</i>) and value (<i>V</i>) in the TLV-formatted input data do not match; the specified AID is invalid or does not match the specification for P1 or P2; or an incorrect number of padding bytes are included in the input data of an encrypted command.
6A86	The value specified for P1, P2, or both is unsupported or the data specified by P1/P2 does not exist on the card.
6A88	Referenced data not found.
6D00	Unsupported value entered for the INS byte.
6E00	Unsupported value entered for the CLA byte.
9000	The command succeeded and the Card Manager or other security domain has returned the requested data.

Also See: PutData command on page 116

GetStatus

Use the `GetStatus` command to retrieve information about the life cycle status, application privileges settings, and AID of the card's load files, applet instances, Card Manager, and security domains.



Security requirement – A secure channel must be open, and you must satisfy the security level defined for it. (For information about establishing a secure channel, see page 5.)

The information you can retrieve varies according to whether the Card Manager or another security domain is selected when you issue the command:

- **Card Manager selected** – Retrieve life cycle status data for load files and applications of all types (applet instances, the Card Manager, or security domains) or restrict the target elements. For applet instances and security domains, the card also returns application privileges data.
- **Other security domain selected** – Retrieve life cycle status data for one or both of these card elements: the Card Manager and the currently selected security domain.

**Command
Format**

Call a `GetStatus` command with the following APDU format.

CLA	INS	P1	P2	Lc	Input Data	Le	Mode
80/84	F2	<i>ref. ctrl.</i>	00	00/08	— / <i>MAC</i>	<i>var</i>	S/R

where:

Element	Value	Length	Description
P1	<i>var</i>	1 B	<i>reference control</i> : Identifies the target card element(s): <ul style="list-style-type: none">• 20h = Load files• 40h = Applications*• 60h = Applications* and load files• 80h = Card Manager• A0h = Card Manager and load files• C0h = Card Manager and applications*• E0h = Card Manager, applications,* and load files <i>* If you retrieve application status information, the card returns data for applet instances and security domains. You distinguish the two by the value of the application privileges byte.</i>

Element	Value	Length	Description
P2	00h	1 B	Retrieve status for all applicable elements (Card Manager, applications, and load files).
Lc	00/08h	1 B	Length of input data: <ul style="list-style-type: none"> • 00h = If no MAC included, or • 08h = If MAC included with the command.
Input data	4F 00h +	2/10 B	4F 00h + MAC, if included with the command. (For information about including and calculating a MAC, see page 14.)
Le	var.	x B	Response data for each application, load file, or Card Manager is 8–19 bytes long. Descriptions of response data follow.



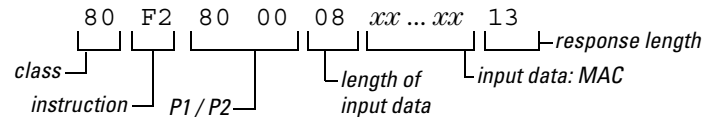
Open Platform Implementation — *The current release of the Cyberflex Access Developer 32K card implements the GlobalPlatform specification v2.0.1 for the GetStatus command with these options or limitations:*

- *You can retrieve status for all applicable occurrences, but not for the “next” occurrence.*
- *You do not include search criteria in the input data.*
- *You use the Le byte to specify the maximum amount of data you want the card to return, rather than automatically returning all available data.*
- *Load files are always physically deleted, so the card never returns a logically deleted life cycle state value.*
- *The GetStatus command is not available if the card is terminated.*

Example

The following example GetStatus command asks the card to return status data for the Card Manager.

```
80 F2 C0 00 08 xx...xx 13
```



where:

- 80 = **CLA** – Standard ISO-compliant command.
- F2 = **INS** – Instruction in the command set is GetStatus.
- 80 = **P1** – Reference control: Return data for the Card Manager, but not other security domains, applet instances, or load files.
- 00 = **P2** – Retrieve data for all occurrences.
- 08 = **Lc** – Number of bytes of input data that follow.
- xx...xx = **Input data** – Placeholder for the command MAC, calculated as described on page 72.
- 13 = **Le** – Length of the expected response data, 19 bytes (including a 16-byte AID).

Response Data

The card returns a stream of response data that consists of an 8–19 byte segment for each targeted card element (such as applet instance, security domain, load file, and Card Manager). Each segment of response data is formatted as described in the following table.

Element	Length	Description
<i>AID lgth</i>	1 B	Length of the AID.
<i>AID</i>	5–16 B	AID of targeted card element (Card Manager, load file, applet instance, or security domain).
<i>life cycle state</i>	1 B	<p>Life cycle state data of targeted card element (Card Manager, load file, applet instance, or security domain). Note that the card returns some values only if the Card Manager is selected.</p> <ul style="list-style-type: none"> • 01h – <i>Load file</i> = Loaded (if Card Manager selected) • 03h – <i>Applet instance / security domain</i> = Installed (if Card Manager selected) • 07h – <i>Card Manager</i> = Initialized; <i>Applet instance / security domain</i> = Selectable • 0Fh – <i>Card Manager</i> = Secured; <i>Applet instance / security domain</i> = Personalized • 7Fh – <i>Card Manager</i> = CM_Locked (if Card Manager selected); <i>Applet instance / security domain</i> = Blocked • FFh – <i>Applet instance</i> = Locked (if Card Manager selected)
<i>application privileges</i>	1 B	<p>Application privileges data (described on page 11).</p> <p><i>The application privileges byte contains meaningful data only for a security domain or applet instance, and only if the Card Manager is currently selected. The application privileges byte of a load file or the Card Manager always has a value of 00h. If a security domain is selected, the application privileges byte of an applet instance also has a value of 00h.</i></p>

NOTE *In a T=0 format `GetStatus` command, if more than 255 bytes of response data are available, the card returns the data serially (using `GetResponse` operations). If the `Le` value (length of expected response data) is less than the length of the available data, the current application returns the status words `6310h`, indicating that additional data is available.*

Status Words Returned The card returns status words to indicate the success or failure of the `GetStatus` command, as described in the following table.

Status	Description
6310	More data is available for return from the card.
6400	Technical problem that has no specified diagnosis.
6700	The specified length of the input data (<code>Lc</code>) is incorrect.
6982	Security status not satisfied. For example, MAC verification failed or the authentication key is locked.
6985	A requirement for using the command is not satisfied. For example, you issued the command outside of a secure channel.
6A80	Incorrect values in input data. Examples: The specified AID is invalid or does not match the P1 specification, or an incorrect number of padding bytes are included in the input data of an encrypted command.
6A86	Incorrect value specified for P1, P2, or both.
6A88	No data found that matches the P1 specification. (For example, no occurrence of the specified type was found in the card registry.)
6D00	Unsupported value entered for the INS byte.
6E00	Unsupported value entered for the CLA byte.
9000	The command succeeded, and the card has returned the status data.

Also See:

- `Install` command on page 93
- “Application Privileges,” on page 10
- `SetStatus` command on page 133
- `GetData` command on page 75

InitializeUpdate

Use the `InitializeUpdate` command to:

- Begin the mutual authentication between the card and host that is needed to establish a secure channel, or
- Terminate a secure channel that is already established, and begin the process of establishing a different secure channel.



Security requirement – None.

Previous command – Call a `SelectApplication` command and select one of the following immediately before you call the `InitializeUpdate` command:

- The Card Manager, if it is not already selected by default
- Another security domain

You can also select an applet instance, if it is designed to use Java API services for mutual authentication.

Card state – The card can be in any of these states: initialized, secured, or locked.

The `InitializeUpdate` command enables the host to authenticate the card. The host sends the card a host challenge (random number) and key identification data. The card generates its own challenge, creates session keys, then uses the host challenge, card challenge, and one of the session keys to generate a card cryptogram. The card returns the card challenge, card cryptogram, key identification information, and card identification data. If the verification succeeds, the host can send an `ExternalAuthenticate` command, which authenticates the host to the card and establishes a secure channel.

If you have not added a security domain with a valid key set to the card, the card uses the default key set to initiate the secure channel. If you add a security domain with a valid key set to the card, the default keys are no longer used.

NOTE *Security domain services are available through the API only if the security domain is in a personalized or blocked state. A security domain can reach these states only if it has been initialized with a valid key set. For this reason, Card Manager keys are never used for the security domain services available through the API.*

Command Format With the Card Manager or other security domain selected, issue an `InitializeUpdate` command with the following APDU format.

CLA	INS	P1	P2	Lc	Input Data	Le	Mode
80	50	<i>keyset vers</i>	<i>key index</i>	08	<i>challenge</i>	1C	S/R

P1 *keyset vers*: Key set version, implicitly or explicitly specified as follows:

- 00h = Implicit key set version (Use this setting if the card does not contain a valid key set other than the default Card Manager key set.)
- *xx* = Explicit key set version number, as specified in the `PutKey` command

P2 *key index*: Index of the key to be used in the specified key set, which can be either of these values:

- 00h = Implicit key index (Use this setting if the card does not contain a valid key set other than the default Card Manager key set.)
- 01h = Explicit key index

Note: The key index identifies the first key in the key set to be used (index of $K_{ENC, AUTH}$). On the Cyberflex Access Developer 32K card, each key set contains three static keys, with $K_{ENC, AUTH}$ as key 1 (index 01h).

Lc 08h: Length of the input data (the host challenge)

Input Data An 8-byte host challenge (random number) that is unique to the current card session.

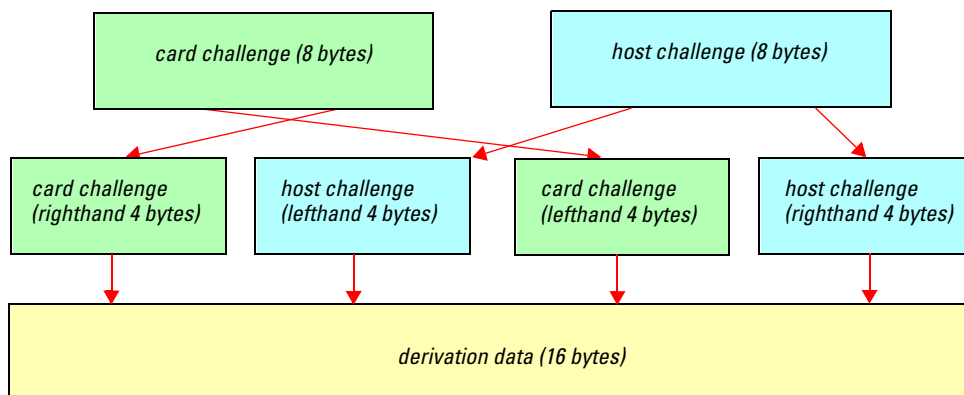
Le 1Ch: Length of the response data the card returns, 28 bytes, formatted as described in the table on page 90.

Key Diversification

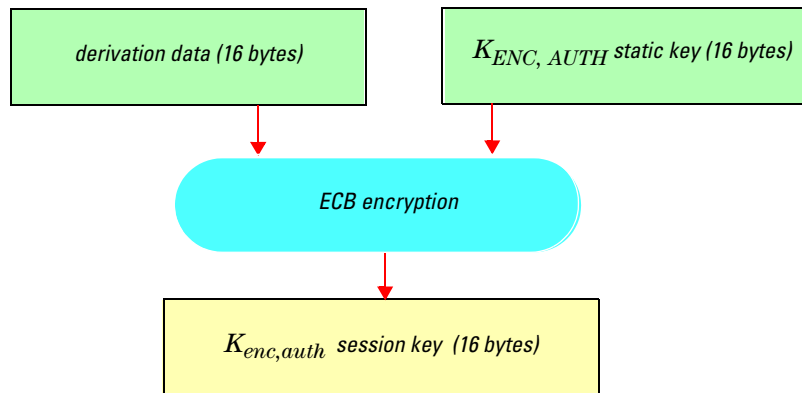
DES session keys are generated each time a secure channel is initiated, and are used in the mutual authentication process. You may use the same session keys for later commands if a security level requires secure messaging. The DES operation used to generate the keys is always triple DES in ECB mode.

Session keys ensure that every secure channel session uses a different set of keys. This is not required for key encryption operations, but it is important for authentication, MAC generation and verification, and command message encryption and decryption. So, you need to create DES session keys from the static secure channel encryption key ($K_{ENC, AUTH}$) and the secure channel MAC key (K_{MAC}), plus the random host and card challenges. To create session keys, follow these steps, illustrated in the subsequent diagrams:

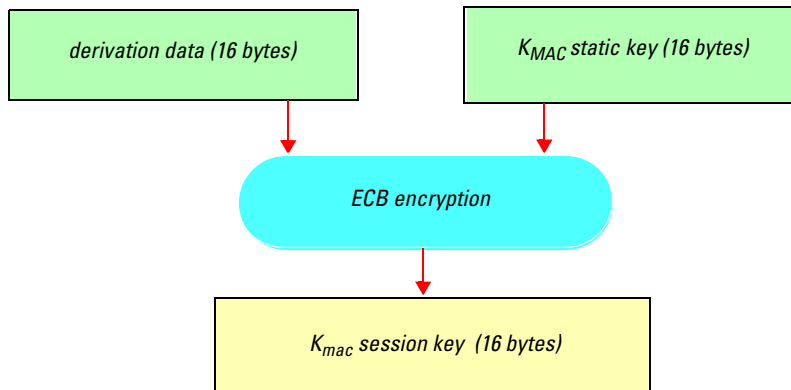
- 1 Generate the session key derivation data (the same data is used to create both the secure channel encryption and MAC session keys).
- 2 Create the secure channel encryption ($K_{enc,auth}$) session key. The $K_{ENC,AUTH}$ session key is a two-key 3DES key, which means that bytes 1-8 and bytes 17-24 are identical.
- 3 Create the secure channel MAC (K_{mac}) session key.



Step 1, Creating Session Keys: Generating the Derivation Data



Step 2, Creating Session Keys: Creating the $K_{enc, auth}$ Session Key

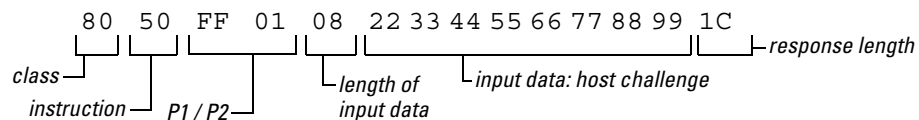


Step 3, Creating Session Keys: Creating the K_{mac} Session Key

Example

The following example APDU data string sends the card an InitializeUpdate command.

80 50 FF 01 08 22 33 44 55 66 77 88 99 1C



where:

- 80 = **CLA** – Standard ISO-compliant command.
- 50 = **INS** – Instruction in the command set is InitializeUpdate.
- FF = **P1** – Key set version to use, #255.
- 01 = **P2** – Index of the key to use in the key set. #1.
- 08 = **Lc** – Number of bytes of input data that follow.
- 22 33 44 55 66 77 88 99 = **Input data** – Placeholder for the host challenge data (an 8-byte random number).
- 1C = **Le** – Length of the expected response data, which is 28 bytes (host cryptogram and command MAC).

Response Data

A 28-byte block that consists of the host cryptogram and command MAC, formatted as described in the following table.

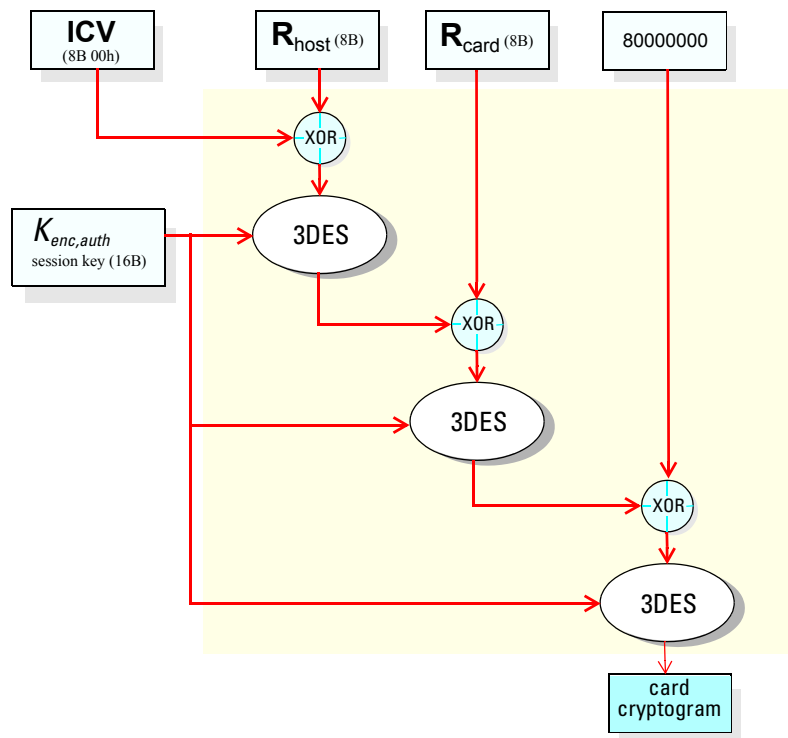
Bytes	Description	Length
1–2	Last two bytes of the Card Manager or security domain AID: <ul style="list-style-type: none"> • The Card Manager AID, if the default key set was used to generate the session keys, or • The security domain AID, if a security domain key set was used to generate the session keys. 	2 B
3–4	Microprocessor fabrication date ¹ .	2 B
5–8	Microprocessor serial number.	4 B
9–10	Microprocessor batch identifier.	2 B
11	Key set version used to generate the session keys, either: <ul style="list-style-type: none"> • The explicit key index, if one was specified in P1. • The default key set version, if P1 = 00h or an explicit value that does not correspond to a valid key set. 	1 B
12	01h = Key index of K _{ENC, AUTH} .	1 B
13–20	Card challenge, which is internally generated.	8 B
21–28	Card cryptogram used for authentication, calculated as shown in the following illustration.	8 B

¹ Bytes 3–10 are identical to each of the following data blocks:
 Bytes 11–18 of card production life cycle (CPLC) data
 Bytes 14–21 of data returned by a GetData command

Card Cryptogram Calculation

The MAC-generated card cryptogram is calculated as follows:

$(K_{enc,auth}, ICV = 8 \text{ bytes of } 00h) [R_{host} || R_{card}]$



Calculation for the Card Cryptogram in the InitializeUpdate Command

Pad the input data with an 8-byte block (80 00 00 00 00 00 00 00h) to indicate that a MAC is included with the input data.

QUESTION or NOTE *does this diagram need to be re-titled or an explanation added that this is showing the CBC mode for calculating a 3DES cipher?*

**Status Words
Returned**

Status	Description
6400	Technical problem that has no specified diagnosis.
6700	The specified length of the input data (Lc) is incorrect. Enter 08h.
6982	Security status not satisfied. For example, MAC verification failed.
6A86	Incorrect value specified for P1, P2, or both.
6A88	Referenced data not found. (For example, the P1 value does not correspond to a valid key set.)
6D00	Unsupported value entered for the INS byte.
6E00	Unsupported value entered for the CLA byte.
9000	The command succeeded. The card is ready to receive an ExternalAuthenticate command.

Also See:

- SelectApplication command on page 127
- ExternalAuthenticate command on page 70

Install

Use one of the `Install` command's four modes to perform these tasks:

- `Install: Load` — Prepare the Card Manager or other security domain for a follow-up `Load` command that will add a load file to the card.
- `Install: Install` — Install an applet instance from a load file that is on the card, or install a security domain.
- `Install: Make Selectable` —
Installed application (applet instance or security domain): Make the application selectable or selected by default whenever the card is reset.
Selectable application: Enable the default selection privilege.
Default selected application: Remove the default selection privilege, and make the application selectable again.
- `Install: Install/Make Selectable` — Simultaneously install and make an application (applet instance or security domain) selectable or selected by default.

NOTE *For an overview of steps for installing a card application, see page page 36.*



Security requirement – A secure channel must be open, and you must satisfy the security level defined for it. (For information about establishing a secure channel, see page 5.)

Target state – *To execute an `Install: Make Selectable` command*, the specified applet instance or security domain must be in an installed state. No other states support the command.

To install an applet instance, the referenced load file must already be on the card, added with a previous sequence of `Install:Load` and `Load` commands.

**Command
Format**

Issue an `Install` command with the following APDU format.

Note: To install a load file or applet instance in a security domain other than the Card Manager, you must select the security domain and establish a secure channel before you issue the `Install` command.

CLA	INS	P1	P2	Lc	Input Data	Le	Mode
80/84	E6	<i>ref control</i>	00	<i>lgth</i>	<i>varies</i>	01	S/R

P1	<i>ref control</i> : Reference control parameter, which you use to set the command mode by entering one of these values: <ul style="list-style-type: none">• 02h = <i>Load</i>: Signals the Card Manager (or other currently selected security domain) that a <code>Load</code> command will follow.• 04h = <i>Install</i>: Installs an applet instance or security domain on the card.• 08h = <i>Make Selectable</i>: Makes an application (applet instance or security domain) selectable, selected by default, or clears the default selection privilege so the application reverts to selectable status.• 0ch = <i>Install/Make Selectable</i>: Installs an application and makes it either selectable or selected by default.
P2	00h: Security control parameter (RFU).
Lc	Length of the input data.
Input Data	The input data is described in the tables found later in this topic.
Le	00h: Receipt length.
Response Data	None: No receipt is available for return, since the Card Manager installs the object directly instead of delegating the task.

Input Data for Adding a Load File to the Card

To add a load file to the card with an `Install: Load` command, format the input data in Length/Value (LV) format as described in the following table.

Length	Description
1 B	<i>Length</i> of the load file AID (a value between 05–10h).
5–16 B	<i>Value</i> of the load file AID (package AID).
1 B	<i>Length</i> of the security domain AID, either: <ul style="list-style-type: none"> • 00h = Current security domain (specified implicitly), or • 05–10h = An explicitly specified security domain.
0 B, or 5–16 B	<i>Value</i> of the security domain AID, either: <ul style="list-style-type: none"> • No data, if the default security domain is used (selected implicitly), or • An explicitly specified security domain.
1 B	<i>Length</i> of the load file hash, either: <ul style="list-style-type: none"> • 00h = If no load file hash is included, or • 14h = If a load file is included.
0 B, or 20 B	<i>Value</i> of the load file hash, either: <ul style="list-style-type: none"> • No data, if no load file hash included, or • SHA-1 load file hash, calculated on the whole load file block, including tags and lengths. (Verification is not mandatory. To verify the command, include load file hash data. Otherwise, no verification is performed.)
1 B	<i>Length</i> of load parameter data, 06h.
6 B	<i>Value</i> of load parameter data in TLV format, described in the following text.
1 B	<i>Length</i> of the load token the card returns, 00h (RFU). The null value indicates that the card returns no token data, since the card does not delegate the task.

Format of the Load Parameters Data Block

Enter a 6-byte block of load parameter data in TLV format as shown in the following example:

EF 04 C6 02 *xx xx*

where:

- EFh *Tag* that signals a system parameters constructed field will follow.
- 04h *Length* of the system parameters constructed field. (This data is 4 bytes long in the example.)
- C6h *Value (Tag)* that signals the following value contains the size of the executable code (program file).
- 02h *Value (Length)* of the card resource value that follows. (This field is 2 bytes long in the example.)
- xx xx* *Value (Value)* – Size of the program file plus the size of the static data.

Installation Buffer

For an `Install: Load` command, the JCRE calls the following method (as defined in the Java Card 2.1 specification):

```
install( byte[] buffer, short offset, byte length )
```

where:

`buffer` A buffer the JCRE creates that contains the following LV-formatted data, which the JCRE retrieves from the `Install: Load` command.

Field	Description
<i>app AID lgth</i>	Length of applet instance AID (1 B), either: <ul style="list-style-type: none"> • 00h – If the applet instance AID = the class AID, or • 05–10h – If an explicitly specified applet instance AID.
<i>app AID</i>	Value of applet instance AID, either: <ul style="list-style-type: none"> • No data – If the applet instance AID = the class AID, or • An explicitly specified applet instance AID (5–16 B).
<i>appl priv lgth</i>	Length of application privileges byte: 01h.
<i>appl priv</i>	Value of application privileges byte. (See page 11.)
<i>appl param lgth</i>	Length of application-specific parameter data (tag C9): 01h.
<i>appl param</i>	Value of application-specific parameter data (tag C9 field), ≤ 32 B for a Cyberflex Access Developer 32K card. To remain fully compliant with Java Card 2.1 specifications, however, limit this data to a maximum of 12–23 bytes, depending on the length of the AID.

`offset` Buffer offset, which points to the length of the instance AID.

`length` Length of all the above defined data. (Maximum length = 32 bytes.)

NOTE *To remain fully compliant with Java Card 2.1 specifications, limit the buffer size to a maximum of 32 bytes. The ISO limit for the buffer size is 255 bytes, which the Cyberflex Access Developer 32K card supports.*

Input Data for Installing an Applet Instance

To install an applet instance with an `Install: Install` or `Install: Make Selectable` command, include an input data block in the LV format described in the following table.

Length	Description
1 B	<i>Length</i> of the load file AID, a value between 05–10h.
5–16 B	<i>Value</i> of the load file AID (package AID) value.
1 B	<i>Length</i> of the applet instance AID in the load file, a value between 05–10h.
5–16 B	<i>Value</i> of the applet instance AID in the load file.
1 B	<i>Length</i> of the applet instance AID, either: <ul style="list-style-type: none"> • 00h = Applet instance AID set to default to the class AID value (AID in the load file), or • 05–10h = Length of an explicitly defined applet instance AID.
0 B or 5–16 B	<i>Value</i> of the applet instance AID, either: <ul style="list-style-type: none"> • No data, to set the applet instance AID to use the default value (the class AID, or AID in the load file), or • An explicitly defined applet instance AID. <p><i>Notes: If you call an <code>Install: Install/Make Selectable</code> command, the card uses the AID you specify in this field (either an explicit value or the default value) to make the applet selectable. The specified AID must match the AID used to register the applet at installation. If the values do not match, the card returns status words 6A80, indicating that the card cannot find the AID in the registry. If you do not explicitly specify an AID, make sure the applet invokes a register method from its install method, as described on page 100.</i></p>
1 B	<i>Length</i> of the application privileges byte: 01h.
1 B	<i>Value</i> of the application privileges byte (described on page 11).
1 B	<i>Length</i> of the install parameter data.
<i>x</i> B	<i>Value</i> of the install parameter data in TLV format, described in the following text.
1 B	<i>Length</i> of the install token the card returns, 00h (RFU). The null value indicates that the card returns no token data, since the card does not delegate the task.
8 B (+)	MAC or MAC+Enc, if needed to satisfy the secure channel requirement.

Format of the Install Parameters Data Block

The install parameter data is in TLV format as shown in the following example:

EF 08 C8 02 || *length(instance data in EEPROM)* || C9 || 02 || *xx xx*

where:

- EFh *Tag* indicating that a system parameters constructed field follows.
- 08h *Length* of the system parameters constructed field. This data is 8 bytes long in the example.
- C8h *Value (Tag)*: Card resource tag 3, which indicates that the size of the instance data in EEPROM follows.
- 02h *Value (Length)*: Number of bytes in the length of the instance data.
- length* *Value (Value)*: Size of the instance data in EEPROM. This data is 2 bytes long in the example.
- C9h *Tag* indicating that an optional field of application-specific parameters follows.
- 02 *Length* of the optional field of application-specific parameters (a value between 0–32 bytes, or 00–20h). This data is 2 bytes long in the example.
- xx xx* *Value* of the optional application-specific parameters data.

Format of the Application Privileges Byte

The application privileges byte is derived from eight binary values in a bit matrix. The following illustration shows the settings in the privilege bit matrix.

bit 8: security domain	bit 7: DAP DES verification	bit 6: RFU (always 0)	bit 5: card locking	bit 4: card termination	bit 3: default selection	bit 2: global PIN modification	bit 1: RFU (always 0)
------------------------------	-----------------------------------	-----------------------------	---------------------------	-------------------------------	--------------------------------	--------------------------------------	-----------------------------

A setting of one (1) enables a privilege; a setting of zero (0) disables it. (Always set RFU bits to 0.) Concatenate the binary values of the eight bits and convert the bit string to a hexadecimal value. The hexadecimal value is the application privileges byte value.

NOTE *For a description of each privilege or attribute that is controlled by the application privileges bit matrix, see page 11. For examples of application privilege byte values, see page 12.*



Open Platform Implementation — *The current release of the Cyberflex Access Developer 32K card implements the GlobalPlatform specification v2.0.1 for the Install command with these options or limitations:*

- *Bit 6, the delegated management bit, is not implemented.*
- *Bit 1, the mandatory DAP verification bit, is not implemented. Use bit 7 for DAP verification.*
- *The Cyberflex Access Developer 32K card does not support delegated management and therefore does not return receipt tokens.*

Notes about Installation Behavior

Including the Register Method — Applet instances typically invoke the register method (as required by Java Card specifications). The form of the register method depends on whether the length of the instance AID byte indicates that the buffer contains an applet instance AID:

- If the buffer contains an applet instance AID, the application invokes:

```
register ( bArray, bOffset, bLength )
```

The bArray parameter contains the AID used for registration.
- If the buffer does not contain an AID, the application invokes:

```
register ( )
```

Mandating Privileges — If you require an applet instance to be installed with specific privileges, set the application to check the install privileges and cancel installation if the privileges are incorrect.

Application-Specific Parameters — Applet instances are responsible for managing any application-specific parameters you include in the `Install` command.

Input Data for Installing a Security Domain

To install a security domain with an `Install: Install` or `Install: Install/Make Selectable` command, include an LV-formatted input data block as shown in the following table.

Length	Description
1 B	<i>Length</i> of the load file AID, 00h (none included).
1 B	<i>Value</i> of the load file AID, 00h (none included).
1 B	<i>Length</i> of the security domain AID, a value between 05–10h.
5–16 B	<i>Value</i> of the security domain AID.
1 B	<i>Length</i> of the application privileges byte, 01h.
1 B	<i>Value</i> of the application privileges byte, defined as described on page 11.
1 B	<i>Length</i> of the install parameter data, 00h (none included).
0 B	<i>Value</i> of the install parameter data included: None.
1 B	<i>Length</i> of the install token the card returns, 00h (RFU). The null value indicates that the card returns no token, since it does not delegate the task.
8 B (+)	MAC or MAC+Enc, if needed to satisfy the secure channel requirement.

NOTE *If you require a security domain to be installed with specific privileges, set the application to check the install privileges and cancel installation if the privileges are incorrect.*

Input Data for Making an Application Selectable

To make an applet instance or security domain selectable with an `Install: Make Selectable` command, include an LV-formatted input data block as shown in the following table.

Length	Description
1 B	<i>Length</i> of the load file AID: 05–10h (5–16 bytes).
5–16 B	<i>Value</i> of the load file AID (package AID).
1 B	<i>Length</i> of the applet instance or security domain AID: 05–10h (5–16 bytes).
5–16 B	<i>Value</i> of the applet instance or security domain AID.
1 B	<i>Length</i> of the application privileges byte: 01h.
1 B	<p><i>Value</i> of the application privileges byte (described on page 11), which is bit-masked to specify the application's selection status. Enter either:</p> <ul style="list-style-type: none"> • 00h = <i>Installed application</i>: Makes the application <i>selectable</i>. • 04h = <i>Installed or selectable application</i>: Makes the application <i>selected by default</i>. Applicable only if the default selection privilege is not already claimed, as described on page 103. <p><i>Application that is selected by default</i>: Makes the application <i>selectable</i>.</p> <p><i>Note</i> – Any value that enables or clears bit 3 has the same effect as 04h (default selection privilege toggled on or off) or 00h (default selection privilege cleared). Only the bit 3 setting takes effect.</p>
1 B	<i>Length</i> of the install parameter field. For a security domain, set this byte to 00h (no install parameter data included).
<i>x</i> B	<i>Value</i> of the install parameter data. Omit this field for a security domain. For an applet instance, include install parameter data in the TLV format described on page 99.
1 B	<i>Length</i> of the install token: 00h (RFU). The null value indicates that the card returns no token data, since the card does not delegate the task.
8 B (+)	MAC or MAC+Enc, if needed to satisfy the secure channel requirement.

Using the Default Selection Privilege

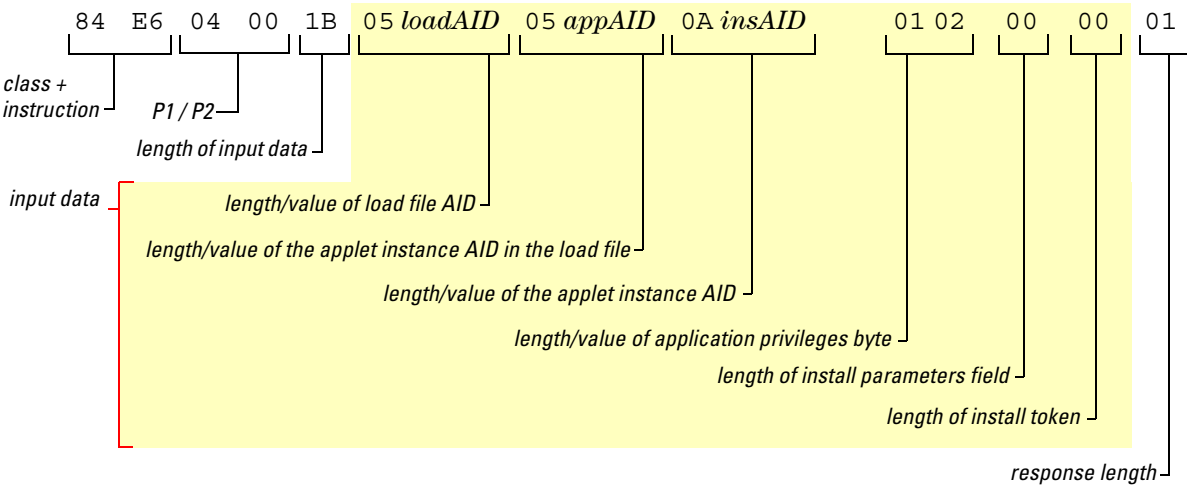
If you issue an `Install` command that gives the default selection privilege to an application (an applet instance or security domain), these conditions apply:

- **Exclusivity** — Only one active card application can have the default selection privilege. If an active application (other than the Card Manager) already has the default selection privilege, you cannot enable the privilege for any other application. (An active application is an applet instance or security domain in an installed, selectable, or personalized state.)
- **Suspension** — If you block or lock an application that has the default selection privilege, another application can claim the privilege. If you unlock or unblock the application that has the suspended privilege, it regains default selection only if no active application (other than the Card Manager) claims the privilege.
- **Special Rules for the Card Manager** — The Card Manager is selected by default if no other application has the default selection privilege. The Card Manager releases the default selection privilege automatically if you call an `Install` command that enables that privilege for another application. The Card Manager becomes selected by default again if you disable either the superseding application or its default selection privilege.

Example

The following example Install command installs an applet instance on the card without making it selectable. (The example does not include MAC or MAC+Enc data.)

84 E6 04 00 1B 05 *loadAID* 05 *appAID* 0A *insAID* 01 02 00 00 00



where:

- 84 = CLA – Open Platform-compliant command that requires a secure channel.
- E6 = INS – Instruction in the command set is SetStatus.
- 04 = P1 – Reference control parameter, which specifies that an applet instance will be installed but not made selectable during installation.
- 00 = P2 – Security control parameter: RFU.
- 1B = Lc – Length of input data that follows is 27 bytes.

- = **Input data:**
- 05 *loadAID* = • Length (1 B) + value (5 B) of the load file AID.
 - 05 *appAID* = • Length (1 B) + value (5 B) of the applet instance AID in the load file.
 - 0A *insAID* = • Length (1 B) + value (10 B) of the applet instance AID.
 - 01 02 = • Length (1 B) + value (1 B) of the application privileges byte, which specifies that the installed object is an applet instance with global PIN change privilege.
 - 00 = • Length (1 B) of the install parameters field (not included in this example).
 - 00 = • Length (1 B) of the install token (not included).
 - 01 = Le – Length (1 B) of the expected response data.

Response Data

If the specified object is installed successfully, the card returns a value of 00h (the length of the install receipt). The null value indicates that no install receipt is available for return, since the Card Manager installs the object directly instead of delegating the task.

Status Words Returned

Status	Description
6400	Technical problem that has no specified diagnosis.
6581	Memory failure.
6700	The specified length of the input data (Lc) is incorrect.
6982	Security status not satisfied. For example, MAC verification failed.
6985	A requirement for using the command is not satisfied. For example, you issued the command outside of a secure channel.
6987	The MAC or other verification data is missing.

Status	Description
6A80	<p>Incorrect parameter value in the input data. Examples:</p> <ul style="list-style-type: none"> • <i>Install: Load</i> — The specified security domain AID is not in the registry, or the load file AID is not in the package. • <i>Install: Install</i> — The specified AID already exists in the registry, or the specified load file AID is not in the package. <i>Installing a security domain:</i> The appropriate application privilege is not enabled for this action. • <i>Install: Install/Make Selectable</i> — The specified AID already exists in the registry, or the specified load file AID is not in the package. <i>Application set to be selected by default:</i> Another application (other than the Card Manager) is already selected by default. <i>Installing a security domain:</i> The appropriate application privilege is not enabled for this action. • <i>Install: Make Selectable</i> — The specified AID does not match the application AID in the registry. <i>Application set to be selected by default:</i> Another application (other than the Card Manager) is already selected by default. • <i>All Install command modes:</i> An incorrect number of padding bytes is included in the input data. <p><i>Note: The term application in this context can refer to a security domain or applet instance.</i></p>
6A84	Insufficient EEPROM memory is available on the card to add the object.
6A86	Incorrect value specified for P1, P2, or both.
6D00	Unsupported value entered for the INS byte.
6E00	Unsupported value entered for the CLA byte.
6F00	Error thrown by the JVM that has no specified diagnosis.
9000	The command succeeded.
9481	The target has an invalid life cycle state for the operation. For example, an <i>Install: Make Selectable</i> operation failed because the specified target is not in an installed state.

- Also See:**
- Load command on page 107
 - “Roadmap for Developing a Card Applet,” on page 143
 - “Converting Source Code to a Card Application,” on page 160

Load

Use Load commands to add a load file to the card from a program file on the host system. A single Load command transfers a limited amount of data, so you call Load commands in a series.



Security requirement – A secure channel must be open, and you must satisfy the security level for the secure channel. If the card security domain requires DAP verification, include a MAC in Cipher Block Chaining (CBC) mode, computed on the program file. You typically include the verification data before the first load file block. Alternatively, you can include the verification data after the final block.

Previous command – Call a successful Install: Load command immediately before you call the first Load command in a series, to specify the program file you want to load. Call a successful Load command with the preceding block number specified as P2 immediately before a follow-up Load command.

Object state – You can install a load file only from a valid program file.

Additional Information to Help You Get Started

- *Establishing a secure channel — page 5 (overview)*
- *Security levels — page 7 (overview) and page 93 (setting during installation)*
- *Computing a MAC — page 53*
- *Creating a program file — page 160*

Identifying the Load Block — Use P2 to number the load file blocks in increments of one (beginning with 00h). Use P1 to specify whether the current load file block is the final one. Use the Lc byte to specify the amount of input data the host system will send the card. The host system uses the P2 and Lc values to identify the appropriate block of program file data to copy to the card.

Length of Input Data — For a Cyberflex Access Developer 32K card, the field of input data is a maximum length of 255 bytes, including the MAC. (When you select the Card Manager with a SelectApplication command, the card response data includes the value of the maximum amount of input data you can send with each Load command.) The minimum data field for a Load command is 1 byte, unless the field includes DAP data. In this case, the minimum amount of input data is 1 byte plus the full 8-byte block of DAP data.

Linking and Checking the Load File — After you send the final load file block, the card links the load file to the appropriate Java libraries and packages. The card also verifies that the referenced packages exist and are the correct version. The card performs these checks at load time rather than at runtime.

Command Format Use the following format to issue a Load command immediately after a successful Install: Load command (to begin loading) or immediately after a successful Load command (to continue or complete loading).

CLA	INS	P1	P2	Lc	Input Data	Le	Mode
80/84	E8	<i>ref control</i>	<i>block num</i>	<i>lgth</i>	<i>var</i>	00	S or S/R
P1	<i>ref control</i> : Reference control parameter, which sets the command mode: <ul style="list-style-type: none">• 00h = <i>Intermediate</i>: More load blocks follow.• 80h = <i>Final/Only</i>: Final load block.						
P2	<i>block num</i> : Number of the load file block. (Number load file blocks sequentially, beginning with 00h and continuing in increments of one. If you have more than 255 load file blocks, FFh, restart the numbering at block 256 with the value 00h.)						
Lc	<i>lgth</i> : Length of the input data: The load file block plus MAC or MAC+Enc, if needed.						
Input Data	Input data = <ul style="list-style-type: none">• The load file block (formatted as per verification type and serial position of the command) +• MAC or MAC+Enc (if included). <i>For information about input data format, see:</i> <ul style="list-style-type: none">• <i>No verification</i> — page 109• <i>Verification data sent with first Load command</i> — page 110• <i>Verification data sent with final Load command</i> — page 111						
Le	Length of the response data the card returns for successful completion of the final Load command: 00h.						
Response Data	None.						

Response Data

If the final load file is added to the card successfully, the card returns a value of 00h—the length of the load receipt. No receipt is available for return, since the Card Manager loads the data directly instead of delegating the task.

- NOTES**
- *To instantiate the load file you have added to the card, call an Install: Install command.*
 - *The loading process is atomic: if loading fails at any point, the card reclaims the EEPROM used so far. To restart loading, begin by issuing another Install: Load command.*

Format of Input Data: No Verification

If you add a load file to the card without verification, send the first load file block formatted in the Tag/Length/Value (TLV) format shown in the following table.

First Load Command

Length	Value	Description
1 B	C4	Tag indicating that the program file follows (in blocks).
1–3 B	01– 82 FFFFh	Length of the program file, a 1- to 3-byte value: <ul style="list-style-type: none"> • 1 B = program file less than 128 bytes long (01–7Fh) • 2 B = program file 128–255 bytes long (81 80–81 FFh), or • 3 B = program file 256–65,535 bytes long (82 0100–82 FFFFh).
<i>x</i> B	<i>varies</i>	Value of the first load file block.

Intermediate and Final Load Commands

To load an intermediate or final load file block, include the value of the load file block as input data without Tag/Length data.

Format of Input Data: First Block Verified

To send a `Load` command series that includes DAP verification at the beginning of the program file, format the input data as described here. (*Note that the card verifies only the first DAP block you send. If you send additional DAP blocks, the card ignores them.*)

First Load Command

Length	Value	Description
1 B	E2h	<i>Tag</i> indicating that a DAP block follows.
1 B	10–1Bh	<i>Length</i> of the DAP block. (+ If MAC+Enc security is used, increase this value to accommodate the encryption.)
1 B	4Fh	<i>Value (Tag)</i> indicating the DAP ID.
1 B	05–10h	<i>Value (Length)</i> of the security domain AID.
5–16 B	<i>varies</i>	<i>Value (Value)</i> of the security domain AID. (For the current Cyberflex Access card, this is the Card Manager's security domain.)
1 B	C3h	<i>Tag</i> indicating that the DAP follows.
1 B	08h	<i>Length</i> of the DAP.
8 B	<i>varies</i>	<i>Value</i> of the DAP—the MAC or MAC+Enc. Compute a MAC on the entire program file, excluding its tag and length. Use Cipher Block Chaining (CBC) mode with the 8-byte MAC session key (K_{MAC}) the card generates during the <code>InitializeUpdate</code> operation. For the initialization vector, use the MAC value the card generated for the previous command.
1 B	C4	<i>Tag</i> indicating that the program file follows, C4h.
1–3 B	01– 82 FFFFh	<i>Length</i> of the program file, a 1- to 3-byte value: <ul style="list-style-type: none"> • 1 B = program file less than 128 bytes long (01–7Fh) • 2 B = program file 128–255 bytes long (81 80–81 FFh), or • 3 B = program file 256–65,535 bytes long (82 0100–82 FFFFh).
<i>x</i> B	<i>varies</i>	<i>Value</i> of the first load file block.

Intermediate and Final Load Commands

To load an intermediate or final load file block with first-block DAP verification, include the load file block value as input data without Tag/Length formatting. (The final block must contain at least 1 byte of data.)

Format of Input Data: Final Block Verified

To send a Load command series that includes DAP verification at the end of the program file, format the input data as described here. *(Note that the card verifies only the DAP block at the end of the program file. If you send DAP blocks earlier in the command sequence, the card ignores them.)*

First Load Command

Send the first load file block with TLV-formatted input data as described in the following table.

Length	Value	Description
1 B	C4h	<i>Tag</i> indicating that the program file follows.
1–3 B	01– 82 FFFFh	<i>Length</i> of the program file, a 1- to 3-byte value: <ul style="list-style-type: none"> • 1 B = program file less than 128 bytes long (01–7Fh) • 2 B = program file 128–255 bytes long (81 80–81 FFh), or • 3 B = program file 256–65,535 bytes long (82 0100–82 FFFFh).
<i>x</i> B	<i>varies</i>	<i>Value</i> of the first load file block.

Intermediate Load Command

For each intermediate load file block, include only the load file block as input data. Do not include TLV formatting.

Final Load Command

Do not precede the final load file block with TLV-formatted elements. Follow the load file block with a DAP block formatted as described in the following table.

Length	Value	Description
<i>x</i> B	<i>varies</i>	Final load file block. (Must contain at least 1 byte of data.)
1 B	E2h	<i>Tag</i> indicating that a DAP block follows.

Length	Value	Description
1 B	10–1Bh	<i>Length</i> of the DAP block.
1 B	4Fh	<i>Tag</i> indicating that the DAP ID follows.
5–16 B	<i>varies</i>	<i>Value</i> of the security domain AID. (For the current Cyberflex Access Developer 32K card, this is the Card Manager AID.)
1 B	C3h	<i>Tag</i> indicating that the DAP follows.
1 B	08h	<i>Length</i> of the DAP.
8 B	<i>varies</i>	<i>Value</i> of the DAP—MAC or MAC+Enc. Compute a MAC on the entire program file, excluding its tag and length. Use CBC mode with the 8-byte MAC session key (K_{MAC}) the card generates during the <code>InitializeUpdate</code> operation. For the initialization vector, use the MAC value the card generated for the previous command.

Status Words Returned

Status	Description
6400	Technical problem that has no specified diagnosis.
6581	Memory failure.
6700	The specified length of the input data (L_c) is incorrect.
6982	Security status not satisfied. For example, the current security domain requires DAP verification, and no verification data was included with the input data. As another example, MAC verification failed
6985	A requirement for using the command is not satisfied. For example, the command was not issued in a secure channel or the preceding command was not the required one.
6A80	Incorrect parameter in the data field. For example, the first block of input data does not begin with C4h and/or valid length; the AID is incorrect; or the load file references a package that is not on the card.
6A84	Insufficient EEPROM memory available on the card to add the specified data. Enough EEPROM must be available to hold the entire program file.
6A86	Incorrect value for P1, P2, or both. (For example, P1 does not have a value of 80h or 84h, or the block number specified in P2 is invalid.)
6D00	Unsupported value entered for the INS byte.
6E00	Unsupported value entered for the CLA byte.
9000	The command succeeded.

- Also See:**
- InitializeUpdate command on page 85
 - Install command on page 93

PinChange

A new Cyberflex Access Developer 32K card does not have a global PIN on it. You can use the PinChange command to add a global PIN to the card. You can also replace or unblock the global PIN, and reset the PIN try counter to the maximum limit. (Each unsuccessful PIN verification attempt decrements the PIN try counter.) You cannot use the PinChange command to add a PIN that is specific to a security domain or applet instance.



Security requirement – A secure channel must be open, and you must satisfy the security level defined for it. (For information about establishing a secure channel, see page 5.)

The Card Manager on a Cyberflex Access Developer 32K card has the necessary privilege to execute a PinChange command. If you can establish a secure channel with the Card Manager selected, you can execute the PinChange command. If you install a security domain other than the Card Manager, then select the security domain, you can execute a PinChange command successfully only if you enabled the PIN change privilege at installation. Similarly, an applet instance you add to the card and select can execute a PinChange command successfully only if you enabled the PIN change privilege when you installed the applet instance.

PIN Encryption — To change the PIN value, you must load it in encrypted form, using the K_{KEK} key, with the security level required by the current secure channel (as described on page 125). If the secure channel requires MAC+Enc, you perform double key encryption by using the diversified key $K_{enc, auth}$ on top of the K_{KEK} key encryption.

Command Format								
	CLA	INS	P1	P2	Lc	Input Data	Le	Mode
	80/84	24	00	00/03-0F	00/08/10 (+)	PIN data	—	—/S
P1	00h							
P2	<i>PIN try limit</i> , specified by either: <ul style="list-style-type: none">00h = <i>Unblock the PIN</i>: The PIN try counter is reset to its maximum limit. Do not include PIN value data with the input data.03-0Fh = <i>Update or add PIN</i>: Specify the PIN try limit by entering a value from 03-0Fh (3-15). Include PIN value data as input.							
Lc	<i>input data length</i> : <ul style="list-style-type: none">00h = <i>Unblock the PIN w/o MAC</i>: (P2 = 00h) No input data.08h = <i>Unblock the PIN + MAC</i>: (P2 = 00h) Include only the MAC as input data. (+) <i>If the secure channel requires MAC+Enc, increase the value as appropriate for the data length.</i>08h = <i>Update or add PIN w/o MAC</i>: Use P2 to set the PIN try limit and include 8-byte encrypted PIN as input data.10h = <i>Update or add PIN + MAC</i>: Use P2 to set the PIN try limit and include 8-byte encrypted PIN + MAC as input data. (+) <i>If the secure channel requires MAC+Enc, increase the value as appropriate for the data length.</i>							
Input Data	Include input data as described in the following table.							
Le	<i>response data length</i> : None							

Format of Input Data

If you set the P2 value to 00h, enter the global PIN block in encrypted form. The PIN block is in big-endian format. The format for a PIN block in cleartext is shown in the following table.

Byte/ Nibble	1	2	3	4	5	6	7	8
	MSB LSN	MSB LSN	MSB LSN	MSB LSN	MSB LSN	MSB LSN	MSB LSN	MSB LSN
Value	2 4–C	0–9 0–9	0–9 0–9	0–9/F 0–9/F	0–9/F 0–9/F	0–9/F 0–9/F	0–9/F 0–9/F	F F
Use	CP L	<i>mand. PIN</i>	<i>mand. PIN</i>	<i>opt. PIN</i>	<i>opt. PIN</i>	<i>opt. PIN</i>	<i>opt. PIN</i>	<i>padding</i>

CP Control parameter: Always set to a value of 2.

L PIN length: Can have a value of 4–C (4–12 nibbles).

mand. PIN *Mandatory PIN digits*: The PIN must be at least 2 bytes long. Set each PIN nibble to a value of 0–9.

opt. PIN *Optional PIN digits*: If the PIN is more than 2 bytes (4 nibbles) long, set a value of 0–9 for each additional nibble needed.

padding Set padding nibbles to a value of F.

Status Words Returned

The card returns status words to indicate the success or failure of the PinChange command, as described in the following table.

Status	Description
6400	Technical problem that has no specified diagnosis.
6581	Memory failure.
6700	The specified length of the input data (Lc) is incorrect.
6982	Security status not satisfied. For example, MAC verification failed.
6985	A requirement for using the command is not satisfied. For example, you issued the command outside of a secure channel, or the currently selected security domain does not have the PIN change privilege.
6A80	Incorrect parameter value in the input data. For example, the lengths of the encrypted message and cleartext are inconsistent, possibly due to a wrong number of padding bytes in the input data; or because the length (L) and value (V) in the LV-formatted input data do not match.

Status	Description
6A86	Unsupported value specified for P1, P2, or both. For example, the P1 value does not equal 00h and the P2 value is less than 03h or greater than 0Fh.
6D00	Unsupported value entered for the INS byte.
6E00	Unsupported value entered for the CLA byte.
9000	The command succeeded.

PutData

Use the `PutData` command to add or update a tagged data object. You can add or replace these types of data objects:

- Card issuer BIN, a data object you can use to identify a card or its key sets on the platform level (independent of applet instances)
- Card issuer data, another platform-level identification object
- Card Manager AID
- Card production life cycle (CPLC) personalization data
- CPLC pre-personalization data
- Security domain AID (if the currently selected application is a security domain)



Security requirement – A secure channel must be open, and you must satisfy the security level defined for it. (For information about establishing a secure channel, see page 5.)

Card state – You can issue a `PutData` command to a card that is in either of these states: initialized or secured.

Previous command – To update any data other than a security domain AID, make sure the Card Manager is selected. To update a security domain AID, select the security domain before you call the `PutData` command.

NOTE *If you update a data object on the card other than an AID, you must use data of the same format and length as the current one. If you update an AID, the new AID can be any valid length.*

**Command
Format**

Call a PutData command with the following APDU format.

CLA	INS	P1	P2	Lc	Input Data	Le	Mode
80/84	DA	<i>data type</i>	<i>data descr</i>	<i>lgth +</i>	<i>var</i>	—	S

P1 / P2 *data type / data descr* – P1 and P2 together specify the data object tag. Use one of the following combinations listed below, depending on whether the Card Manager or a security domain is currently selected:

Card Manager selected

- 00 42h = **Card issuer BIN**. Add or update the card issuer BIN, a data object up to 8 bytes long. If updating, the new data must match the current data in length.
- 00 45h = **Card issuer data**. Add or update the 8-byte card issuer data. Both new and updated card issuer data must be 8 bytes long.
- 00 4Fh = **Card Manager AID**. Update the Card Manager AID. The new AID can have a different length than the current one, provided it has a valid length between 5 and 16 bytes long. All subsequent references to the AID must indicate the new AID, including the AID in the file control information (FCI) the card returns when you select the Card Manager.
- 9F 66h = **CPLC personalization data**. Add or update card production life cycle (CPLC) personalization data.
- 9F 67h = **CPLC pre-personalization data**. Add or update CPLC pre-personalization data.

Security domain selected

- 00 4Fh = **Security domain AID**. Update the AID of the currently selected security domain. The new AID can be any valid length between 5 and 16 bytes. All subsequent references to the AID must indicate the new AID, including the AID in the FCI the card returns when you select the security domain.

Lc	<p><i>Input data length</i> – A single byte, whose value varies according to the data object and security requirement:</p> <ul style="list-style-type: none"> • Card issuer data — 08h (10h if MAC included, or increase length as appropriate for MAC+Enc) • Card Manager AID — 05–10h (0D–18h if MAC included, or increase length as appropriate for MAC+Enc) • CPLC personalization data — 08h (10h if MAC included, or increase length as appropriate for MAC+Enc) • CPLC pre-personalization data — 08h (10h if MAC included, or increase length as appropriate for MAC+Enc) • Card issuer BIN — 01–08h (09–10h if MAC included, or increase length as appropriate for MAC+Enc) • Security domain AID — 05–10h (0D–18h if MAC included, or increase length as appropriate for MAC+Enc)
Input Data	<p>Include input data according to the data object you are adding or updating on the card as follows:</p> <ul style="list-style-type: none"> • Card issuer data — 8 bytes of card issuer data + 8-byte cryptogram, if MAC verification required • Card Manager AID — 5–16 byte application AID + 8-byte cryptogram, if MAC verification required • CPLC personalization data — 8 bytes formatted as follows: B 1–2 = Integrated circuit personalizer (2B) B 3–4 = Integrated circuit personalization date (2B) B 5–8 = Integrated circuit personalization equipment (4B) + 8-byte cryptogram, if MAC verification required • CPLC pre-personalization data — 8 bytes formatted as follows: B 1–2 = Integrated circuit pre-personalizer (2B) B 3–4 = Integrated circuit pre-personalization date (2B) B 5–8 = Integrated circuit pre-personalization equipment (4B) + 8-byte cryptogram, if MAC verification required • Card issuer BIN — 1–8 bytes of card issuer BIN data + 8-byte cryptogram, if MAC verification required

**Status Words
Returned**

The card returns status words to indicate the success or failure of the PutData command, as described in the following table.

Status	Description
6400	Technical problem that has no specified diagnosis.
6581	Memory failure.
6700	The specified length of the input data (Lc) is incorrect.
6982	Security status not satisfied. For example, MAC verification failed.
6985	A requirement for using the command is not satisfied. For example, you issued the command outside of a secure channel or the card is locked.
6A80	Incorrect parameter value in the input data. For example, either the length (Lc) does not match the value of the input data, or the input data contains an incorrect number of padding bytes.
6A82	The specified value is incorrect for P1, P2, or both.
6A86	The value specified for P1, P2, or both is unsupported.
6D00	Unsupported value entered for the INS byte.
6E00	Unsupported value entered for the CLA byte.
9000	The command succeeded.

Also See:

- GetData command on page 75
- “Privileges and Attributes Specified in the Application Privileges Byte,” on page 11

PutKey

Use the PutKey command to:

- Replace a key or multiple consecutive keys in an existing key set
- Replace an existing key set with a new one (with at least one updated key), or
- Add a new key set that contains three DES keys.
- *Cyberflex Access e-gate 32K card only:* Add a key set containing an RSA public key (the public key of an RSA public key pair).



Security requirement – A secure channel must be open, and you must satisfy the security level defined for it. (For information about establishing a secure channel, see page 5.)

Key Identification — A key is identified by its key set version and its position (index) in the key set. Each key set contains three static keys — key 1 ($K_{\text{ENC, AUTH}}$), key 2 (K_{MAC}), and key 3 (K_{KEK}). Each security domain on the card (including the Card Manager) can have multiple key sets.

Key Encryption During Loading — Use the static key K_{KEK} to load keys in encrypted form with the security level the active secure channel requires. If the current secure channel requires MAC+Enc, you perform double key encryption by using the diversified key $K_{\text{enc, auth}}$ on top of the K_{KEK} key encryption. If you load a key set for a security domain other than the Card Manager, use the Card Manager's default key set for loading.

Default Key Sets — The original key set that comes loaded on a new card is the Card Manager's default key set. If you create another security domain, select it, and add a key set, that first key set is the security domain's default key set for the life of the security domain.

NOTE *The default Card Manager key set on a new Open Platform Cyberflex Access card has the key set version number 01 (depending on how the card was personalized). The default key values are:*

- **AUTH key (key 1)** – 404142434445464748494A4B4C4D4E4F
- **MAC key (key 2)** – 404142434445464748494A4B4C4D4E4F
- **KEK key (key 3)** – 404142434445464748494A4B4C4D4E4F



If you add or change a key set associated with the Card Manager or another security domain, keep records about the changes you make. The GlobalPlatform specification does not currently support tracking the key sets that are on a card, or for key set contents or version numbers. You cannot retrieve data for an existing key or key set, so you must maintain accurate records.

**Command
Format**

Call a PutKey command with the following APDU format.

CLA	INS	P1	P2	Lc	Input Data	Le	Mode
80/84	D8	keyset version	key index	in lgth	key data +	out lgth	S/R

P1 *keyset version:* The key set version to be added or updated, specified by:

- 00h = A new key set version is being added, whose value is defined in the input data.
Use this value when adding an RSA public key.
- 01h–7Fh = An existing key set version is being updated with one or more new keys

P2 *key index:* Index of the first or only key whose value is to be updated in the key set, specified by a value from 01h–03h.

A single key (key index 1) will always be specified when adding an RSA public key.

- 01h = *Update key 1 ($K_{enc, auth}$) in an existing key set.*
- 02h = *Update key 2 (K_{MAC}) in an existing key set.*
- 03h = *Update key 3 (K_{KEK}) in an existing key set.*
- 81h = *Add a new key set, update all 3 keys in an existing key set, or update key 1 and key 2 in an existing key set.* If the P1 value corresponds to an existing key set version, the amount of input data determines whether you update all 3 keys or just the first 2 keys.
- 82h = *Update key 2 and key 3 in an existing key set.*

- Lc** *in length* – The input data length is a single byte, whose value varies according to the number of keys updated, as noted below.
- *Updating 1 key* — 17h (23 bytes) (+ If the secure channel requires a MAC or MAC+Enc, add 8 bytes to the length for the MAC, and extra bytes if needed for encryption.)
 - *Updating 2 keys* — 2Dh (45 bytes) (+ If the secure channel requires a MAC or MAC+Enc, add 8 bytes to the length for the MAC, and extra bytes if needed for encryption.)
 - *Adding a key set or updating 3 keys* — 43h (67 bytes) (+ If the secure channel requires a MAC or MAC+Enc, add 8 bytes to the length for the MAC, and extra bytes if needed for encryption.)
- Input Data** Include input data as described later in this topic, plus MAC or MAC+Enc if required.
- Le** *out lgth*: The output (or response) data length the card returns if the command is completed successfully, a value from 04–0Ah (4–10 bytes).

NOTES

- *If you replace a key, the new key must be the same length as the original key, 16 bytes.*
- *Key loading and updating operations are atomic, so if a failure occurs during processing, the card discards all the new or updated keys.*
- *To add a key set for a security domain other than the Card Manager, select the security domain, then call the PutKey command.*



Open Platform Implementation — *The current release of the Cyberflex Access Developer 32K card implements the GlobalPlatform specification v2.0.1 for the PutKey command with these options or limitations:*

- *The card does not support P1 values for PutKey commands issued as a series.*
- *The card does support the option for key check values.*



Open Platform Implementation — *The current release of the Cyberflex Access e-gate 32K card implements the GlobalPlatform specification v.2.0.1 and the Visa Card Implementation Requirements Configuration 3 for the PutKey command with this option:*

- *The card does support the option for key check values.*

Format of Input Data

Byte(s)	Description	Length
1	Key set version number indicator: <i>New key set:</i> <ul style="list-style-type: none"> 01–7Fh = Explicit key set version (P1=00h). Cannot match an existing key set version number. must be always 00h for RSA keys. <i>or</i> <i>Updated key set:</i> <ul style="list-style-type: none"> 00h = If P1 specifies an existing key set version number. In this case, the key set version number remains the same. 01–7Fh = Replacement value for an existing key set version number that is specified in P1. 	1 B
2	First (or only) key: must be always 01h for RSA keys. Only one RSA key can be loaded within one APDU. 81h = Algo-ID for DES algorithm in ECB mode, used to generate the session keys for ($K_{enc, auth}$) and (K_{MAC}). Since (K_{KEK}) is also used in ECB mode, the Algo-ID value must be 81h for DES keys.	1 B
3	A1h = Algo-ID for RSA keys.	1 B
4–19	10h = Key length. Must be double-length DES key, 16 bytes long. Key value, encrypted with K_{KEK} as described on page 125, or with K_{KEK} and $K_{enc, auth}$ encryption if the current secure channel requires MAC+Enc.	16 B
20	03h = Length of the check value.	1 B
21–23	Check value (bytes 8, 7, and 6), which the card uses to verify the integrity of the key, as described on page 125.	3 B
24	Second key (if any): 81h = Algo-ID for DES algorithm in ECB mode, used to generate the session keys for ($K_{enc, auth}$) and (K_{MAC}). Since (K_{KEK}) is also used in ECB mode, the Algo-ID value must be 81h.	1 B
25	10h = Key length. Must be double-length DES key, 16 bytes long.	1 B
26–41	Key value, encrypted with K_{KEK} or with K_{KEK} and $K_{enc, auth}$ encryption if the current secure channel requires MAC+Enc. 03h = Length of the check value.	16 B
42	Check value (bytes 8, 7, and 6), which the card uses to verify the integrity of the key.	1 B
43–45		3 B

Byte(s)	Description	Length
	Third key (if any):	
46	81h = Algo-ID. DES algorithm in ECB mode, used to generate the session keys for ($K_{enc, auth}$) and (K_{MAC}). Since (K_{KEK}) is also used in ECB mode, the Algo-ID value must be 81h.	1 B
47	10h = Key length. Must be double-length DES key, 16 bytes long.	1 B
48–63	Key value, encrypted with K_{KEK} or with K_{KEK} and $K_{enc, auth}$ encryption if the current secure channel requires MAC+Enc.	16 B
	03h = Length of the check value.	
64	Check value (bytes 8, 7, and 6), which the card uses to verify the integrity of the key.	1 B
65–67		3 B

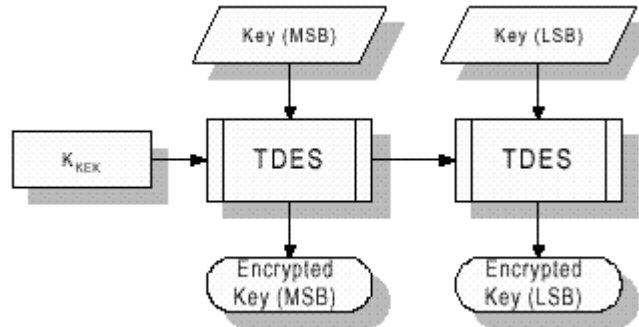
Response Data Returned by the Card

Byte(s)	Description	Length
1	01–7Fh = The number of the key set version added or updated, in clear text.	1 B
2–4	Either: <ul style="list-style-type: none"> • <i>New key set:</i> Returned check value for first key added, or • <i>Updated key set:</i> Returned check value for first (or only) key updated. <i>Note: The host can use a returned check value to verify that the correct key value that was loaded is correct after decryption.</i>	3 B
5–7	Either: <ul style="list-style-type: none"> • <i>New key set:</i> Returned check value for second key added, or • <i>Updated key set: (Optional)</i> Returned check value for second key updated (if any). 	3 B
8–10	Either: <ul style="list-style-type: none"> • <i>New key set:</i> Returned check value for the third key added, or • <i>Updated key set: (Optional)</i> Returned check value for third key updated (if any). 	3 B

Key Encryption Mechanism

Encrypt the keys you add or update with a 3DES key in ECB mode, according to the following formula:

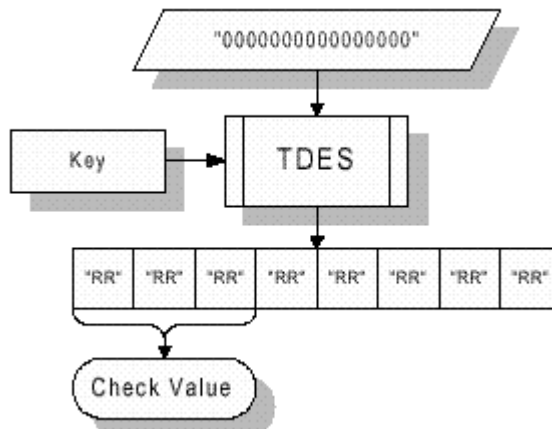
Encrypted key = $\text{TDES_ECB}(K_{\text{KEK}})[\text{Key}]$



Key Check Value Processing

If you load a key, the card performs key check processing as described below. The card returns key check values the host can use to verify that the correct key value was loaded.

Key check value = $\text{TDES}(\text{Key})[00\ 00\ 00\ 00\ 00\ 00\ 00\ 00]$



**Status Words
Returned**

Status	Description
6400	Technical problem that has no specified diagnosis.
6581	Memory failure.
6700	The specified length of the input data (Lc) is incorrect.
6982	Security status not satisfied. For example, MAC verification failed.
6985	A requirement for using the command is not satisfied. For example, you issued the command outside of a secure channel.
6A80	Incorrect value(s) in input data. For example, the specified key set value or key index is invalid.
6A84	Not enough EEPROM available on the card for the new key set.
6A86	The specified value is incorrect for P1, P2, or both. For example, value for P1 > 7Fh, or P2 value is not 01–03h.
6A88	Referenced data not found. For example, key set version specified in P1 not found in the security domain.
6D00	Unsupported value entered for the INS byte.
6E00	Unsupported value entered for the CLA byte.
9000	The command succeeded.
9484	Unsupported algorithm ID included in input data. (Enter 81h for algo-ID for DES keys, and A1h for algo-ID for RSA keys.)
9485	Invalid key check value included in input data.

Also See: PutData command on page 116

SelectApplication

Use the `SelectApplication` command to select an application (an applet instance, the Card Manager, or another security domain) or a file in an application's file system.



Security requirement – None.

Card state – If the card is locked, you can select the Card Manager, but no other application. You can select an application other than the Card Manager only if the card is either initialized or secured. You cannot select any application on a terminated card.

Object state – To select an application, its life cycle state must be selectable, personalized, or blocked. You cannot select an application whose state is installed, locked, or deleted.

NOTES

- *If you establish a secure channel, then select the Card Manager, another security domain or an applet instance, the secure channel is terminated.*
- *The Java Card Runtime Environment (JCRC) handles the `SelectApplication` command and make it available at any time, regardless of which application is currently selected.*

Selecting the Card Manager

The Card Manager is the card's default selected application, so you do not need to select the Card Manager unless:

- You selected another application earlier in the card session, or
- You specified another application as selected by default, as described on page 93.

In either case, you can use the `SelectApplication` command to select the Card Manager—for example, in preparation for installing a security domain, load file, or applet instance.

Using a Partial AID to Select an Application

You can use a partial AID to browse through a series of applications (applet instances or security domains), provided the AIDs begin with identical values for the first 5 or more bytes.

To select an application by using a partial AID, perform the following steps:

- 1 Send a `Select` command with the P2 value specified as 00h, and enter the truncated AID as input data.
The card selects the first application in the registry whose initial AID values match the truncated AID.
- 2 Send a `Select` command with the P2 value specified as 02h. As input data, enter the same truncated AID you used in step 1.
The card selects the next matching application in the registry. If the card does not find a match, it returns the status words 6A82h (application not found).
- 3 Repeat step 2 as needed to continue to select applications in the series.

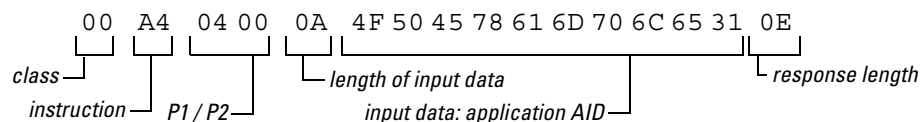
NOTE *You can use the `SelectApplication` command to select any application on the card, overriding any ongoing process method. No command overrides the `SelectApplication` command.*

Command Format Call a `SelectApplication` command with the following APDU format.

CLA	INS	P1	P2	Lc	Input Data	Le	Mode
00/F0	A4	04	00/02	AID length	AID	var	S/R
P1	04h: Specifies that the application to select is the one whose AID is sent to the card as input data.						
P2	<i>Use one of these values to specify which occurrence of the application you want to select:</i> <ul style="list-style-type: none">• 00h – First or only occurrence.• 02h – Next occurrence. (Applicable only if the previous <code>SelectApplication</code> command specified another occurrence of the application, and both occurrences are registered with the truncated AID you specified for both commands.)						
Lc	AID length: 05–10h (5–16 bytes).						
Input Data	Include the application AID as input data.						
Le	Length of the file control information (FCI) the card returns.						
Response Data	File control information the card returns. The FCI is specific to the application selected, as described in the tables found later in this topic.						

Example

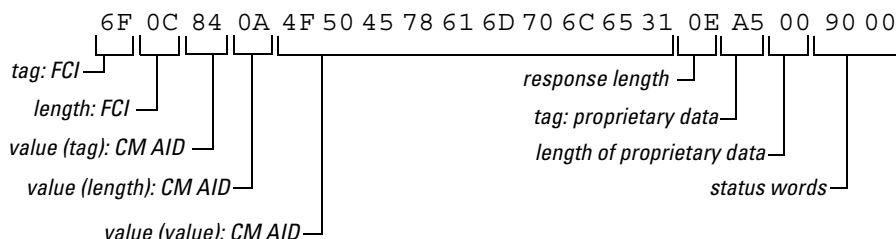
Input Data:



where:

- 00 A4 = **CLA + INS** – Specifies the SelectApplication command.
- 04 = **P1** – Select an application (applet instance), identified by its AID.
- 00 = **P2** – Select the first or only matching occurrence of the application.
- 0A = **Lc** – Ten bytes of input data follow.
- 4F ... 31 = **Input data** – The applet instance AID.
- 0E = **Le** – Fourteen bytes of response data are expected.

Response Data:



where:

- 6F = **Tag** indicating file control information follows.
- 0C = **Length** of file control information (12 bytes).
- 84 = **Value (Tag)** indicating AID of selected application follows.
- 0A = **Value (Length)** of application AID (10 bytes).
- 4F ... 31 = **Value (Value)** of the application AID.
- A5 = **Tag** indicating proprietary data follows.
- 00 = **Length** of proprietary data (0 bytes).
- 90 00 = **Status words** indicating the operation succeeded.

Response Data (Card Manager or Other Security Domain Selected)

If you send a `SelectApplication` command when the Card Manager or another security domain is selected, the card returns the data described in the following table. (Values in the table are shown in hexadecimal format.)

Length	Value	Description
1 B	6F	<i>Tag</i> indicating that FCI data follows.
1 B	<i>varies</i>	<i>Length</i> of the FCI data.
1 B	84	<i>Value (Tag)</i> indicating that the Card Manager AID follows.
5–16 B	05–10	<i>Value (Length)</i> of the Card Manager AID.
1 B	<i>varies</i>	<i>Value (Value)</i> of the Card Manager AID.
1 B	A5	<i>Tag</i> indicating that proprietary data follows.
1 B	0D	<i>Length</i> of proprietary data.
2 B	9F 6E	<i>Tag</i> indicating that the Card Manager production life cycle (CMPLC) data follows (as described in the <code>SetStatus</code> command on page 133).
1 B	06	<i>Length</i> of the CMPLC data.
6 B	<i>varies</i>	<i>Value</i> of the CMPLC data = <ul style="list-style-type: none"> • Card OS ID (2 B), • Card OS release level (2 B), and • Card OS release date (2 B).
1 B	9F 65	<i>Tag</i> indicating that the maximum length for load file blocks follows (for the <code>Load</code> command).
1 B	01	<i>Length</i> of the value that specifies the maximum length of load file blocks.
1 B	<i>varies</i>	<i>Value</i> of the maximum load file block length.

Response Data (Applet Instance Selected)

If you send a SelectApplication command when an applet instance is selected, the card returns the data described in the following table. (Values in the table are shown in hexadecimal format.)

Length	Value	Description
1 B	6F	<i>Tag</i> indicating that FCI data follows.
1 B	<i>varies</i>	<i>Length</i> of the FCI data.
1 B	84	<i>Value (Tag)</i> indicating that the Card Manager AID follows.
5–16 B	05–10	<i>Value (Length)</i> of the Card Manager AID.
1 B	<i>varies</i>	<i>Value (Value)</i> of the Card Manager AID.
1 B	A5	<i>Tag</i> indicating that proprietary data follows.
1 B	00	<i>Length</i> of proprietary data.

Status Words Returned

Status	Description
6283	<i>Card Manager selected</i> – Card Manager is locked. (Card Manager commands are still available.) No change in application selection occurs.
6400	Technical problem that has no specified diagnosis. No change in application selection occurs.
6700	The specified length of the input data (Lc, the AID length) is either an unsupported value or does not match the input data. No change in application selection occurs.
6985	A requirement for using the command is not satisfied. For example, the card is locked and you are trying to select an application other than the Card Manager. No change in application selection occurs.
6999	Application selection failed. If an application was selected when you called the command, the application is deselected and the JCRE is in control.
6A80	Incorrect parameter value in the input data. For example, the P2 value and input data are inconsistent. No change in application selection occurs if the P2 value = 02h.
6A81	<i>Applet instance or security domain (not the Card Manager) selected</i> – The Card Manager or target application is locked. In either case, the application is not selected and its commands are not available. No change in application selection occurs.

Status	Description
6A82	<ul style="list-style-type: none"> • If $P2 = 00h$ – Registry contains no valid application with the specified AID. (The application may be locked or deleted.) • If $P2 = 02h$ – Registry contains no more applications with the specified AID. (The application may be locked or deleted.) <p>No change in application selection occurs.</p>
6A86	Incorrect value specified for P1, P2, or both. No change in application selection occurs.
6D00	Unsupported value entered for the INS byte. No change in application selection occurs.
6E00	Unsupported value entered for the CLA byte. No change in application selection occurs.
6F00	Error thrown by the JVM that has no specified diagnosis.
9000	The command succeeded and the specified application is selected.
<i>Java</i>	If an applet instance is selected, the applet instance itself can trigger other status words.

NOTE *Note that if the Card Manager state is terminated, it does not return any status words—it remains mute.*

SetStatus

Use the `SetStatus` command to change the life cycle state of the card (Card Manager life cycle state, CMLC), another security domain, or an applet instance (application life cycle state, ALC).

With the Card Manager selected, you can use the `SetStatus` command to change the life cycle state of the Card Manager or an applet instance in these ways:

- Secure an initialized card.
- Lock a secured card, or unlock a card that was secured and locked.
- Lock an applet instance, or return a locked applet instance to its previous state.
- Make an installed or blocked applet instance selectable.
- Personalize a selectable or blocked applet instance.
- Block a selectable or personalized applet instance.

With a security domain selected, you can use the `SetStatus` command to change the security domain ALC in these ways when a valid key set has been loaded:

- Make a security domain selectable.
- Personalize or block a selectable security domain.
- Lock or terminate the card, if the currently selected security domain was assigned the appropriate privilege when it was installed.



Security requirement – A secure channel must be open, and you must satisfy the security level defined for it. (For information about establishing a secure channel, see page 5.)

Card state – You can issue a `SetStatus` command to a card that is in any of these states: initialized, secured, or locked.

Command Format Call a `SetStatus` command with the following APDU format.

CLA	INS	P1	P2	Lc	Input Data	Le	Mode
80/84	F0	80/40	<i>status</i>	<i>lgth</i>	— / <i>AID</i>	—	S

Element	Length	Value	Description
P1	1 B	80/40h	Specifies the application that is currently selected and that will execute the <code>SetStatus</code> command, specified by one of these values: <ul style="list-style-type: none">• 40h = Security domain, or• 80h = Card Manager.
P2	1 B	<i>var</i>	Status to set for the target, as described in the P2 Status Values tables that follow.
Lc	1 B	<i>lgth</i>	Length of input data—target AID + MAC or MAC+Enc, if included.
Input data	5–16 B +	<i>var</i>	Target AID: Card Manager, security domain, or applet instance + MAC or MAC+Enc, if included. (For information about these types of verification, see page 14 (MAC), or page 15 (MAC+Enc).)

P2 Status Values (with Card Manager Selected)

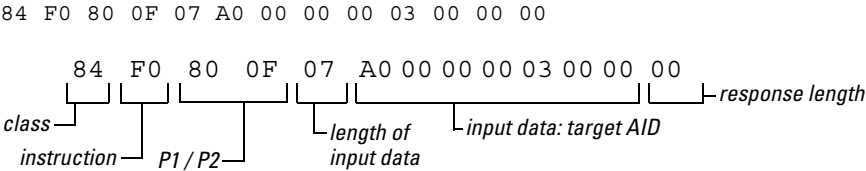
Value	Target and State
03	• <i>Application installed</i> (Applies to applet instance locked from a previous state of installed.)
07	• <i>Application selectable</i> (Applies to installed or blocked applet instance, or one locked while in the selectable state.)
0F	• <i>Application personalized</i> (Applies to selectable or blocked applet instance, or one locked while in the personalized state.) • <i>Card Manager secured</i> (Applies to initialized or locked Card Manager.)
7F	• <i>Application blocked</i> (Applies to selectable or personalized applet instance, or one locked while in the blocked state.) • <i>Card Manager locked</i> (Applies to secured Card Manager.)
FF	• <i>Application locked</i> (Applies to applet instance that is installed, selectable, personalized, or blocked.)

P2 Status Values (with Security Domain Selected)

Value	Target and State
07	• <i>Security domain selectable</i>
0F	• <i>Security domain personalized</i> (Applies to selectable security domain with a valid key set.)
7F	• <i>Card Manager locked</i> (Applies only if card locking privilege was enabled during security domain installation.) • <i>Security domain blocked</i> (Applies to selectable security domain with a valid key set.)
FF	• <i>Card Manager terminated</i> (Applies only if card termination privilege was enabled during security domain installation.)

Example

The following example SetStatus command changes the life cycle state of the Card Manager from initialized to secured. (The example does not include MAC or MAC+Enc data.)



where:

- 84 = CLA – Open Platform-compliant command that requires a secure channel.
- F0 = INS – Instruction in the command set is SetStatus.
- 80 = P1 – (Reference control parameter) The target object is the Card Manager.
- 0F = P2 – Apply the life cycle state “secured” to the target.
- 07 = Lc – Length of input data that follows is 7 bytes.

A0 00 00 00 = **Input data** – Defines the 7-byte AID of the Card Manager.
03 00 00
00 = **Lc** – No response data is expected from the card.

**Status Words
Returned**

Status	Description
6400	Technical problem that has no specified diagnosis.
6581	Memory failure.
6700	The specified length of the input data (Lc) is incorrect.
6982	Security status not satisfied. For example, MAC verification failed.
6985	A requirement for using the command is not satisfied. For example, you issued the command outside of a secure channel or you specified a state transition that is not supported for the target.
6A80	Incorrect value in input data. For example the AID input is incorrect or is inconsistent with the P1 value.
6A86	Unsupported value specified for P1, P2, or both.
6D00	Unsupported value entered for the INS byte.
6E00	Unsupported value entered for the CLA byte.
9000	The command succeeded.

Also See: GetStatus command on page 80



4

Working with Card Applets

This section contains background information for developing Java smart card programs for Open Platform Cyberflex Access cards, and includes a tutorial for creating a sample card applet. These topics are covered:

- Guidelines for Developing Applets (page 137)
- Roadmap for Developing a Card Applet (page 143)
- Writing a Sample Card Applet (page 146)
- Converting Source Code to a Card Application (page 160)
- Sending APDU Commands to the Sample Application (page 171)

Guidelines for Developing Applets

The next few topics describe some programming guidelines that will help you avoid problems and optimize the use of the card.

Card Resource Limitations

The card has a limited amount of EEPROM memory. Here are the memory constraints:

- **Total applet instance size** is limited to 28 KB on Open Platform Cyberflex Access cards.
- **Java heap size** is specified at load time in the Java data container. The heap (memory allocation pool) is a runtime data area used for allocating memory for the class instances, variables, and arrays.

Objects are persistent by default on Java Cards, and the card stores all objects as data associated with the applet instance container. If an allocation requires more than the available Java heap size, the Java Virtual Machine (JVM) issues an exception: `SystemException` with reason `(NO_RESOURCE)`.

- **Java operand stack size** is limited. The Java operand stack contains instructions, arguments, and local variables. If you do not catch a stack size problem before you download the program onto the card, the JVM issues an uncatchable exception at runtime resulting in an ISO 7816.SW_UNKNOWN status (6F00).

Java Card Development Requirements

Observe these requirements as you develop applet instances:

- If you create an applet, create a `process` method and derive from the `Applet` class.
- Use only boolean, byte, and short data types. The card does not support other data types.
- Byte and short data types are *signed*, and must be within the ranges shown below. (For more information about supported data types, see page 141.)

Data Type	Length	Maximum Value	Minimum Value
byte	8 bits	+127	-128
short	16 bits	+32767	-32768

- If you develop a program for an Open Platform Cyberflex Access card, use the debugging option (`-g`) when you compile the source code in the Program File Generator dialog box. (Do not use the optimization option (`-o`).)
- If you develop a program for a Cyberflex Access 16K card, use the debugging option (`-g`) when you compile the source code in the MakeSolo utility.

- The latest version of the Sun Java Converter (either 1.2 or 1.3) requires the package and applet AID RID values (the first five bytes) to be *identical*. This was not a requirement in prior versions, so you might have to modify your package, applet, or applet instance AID values to conform to this restriction.

NOTE *Communication buffers in smart card programs use variable bit patterns, rather than variable values.*

Instantiating the Applet at the Beginning of the Install Method

The first statement in an applet's `install` method must instantiate the applet instance.

Methods That Applets Must Include

An applet must include the `install`, `select`, and `process` methods. Declare these methods in the applet's primary class.

The `install` method creates the class objects the applet will need when you call the `select` method. The `select` method prepares the applet to receive and process APDU commands. You then call the `process` method to handle APDU commands.

Setting the Library Path

Set the host system's environment variable path to the class library directory, and set the path to the Cyberflex Access class files in your development environment.

For example, to set the path to the class library in Visual J++ with Microsoft Developer Studio 5.0, follow these steps:

1 Choose **Tools** → **Options** → **Directories**

2 Point the system to the following path:

*Program Files\Schlumberger\Smart Cards and Terminals\
Cyberflex Access Kits\v4\ClassLibrary\jc_api_21.jar*

General Programming Guidelines

Make the program as efficient as possible to minimize the requirements for the Java operand stack and heap, and to reduce the overall EEPROM the applet instance needs. Sometimes you might have to make your smart card programs less readable to make them more compact.

Here are some tips for using the card to its greatest advantage:

- **Local Scope** — Do not create objects or arrays with a local scope. Each instantiation of an object or array declared with a local scope allocates memory. The card does not perform garbage collection, so this memory is never freed. Each call to the `local` method allocates memory until all of the Java heap resources are exhausted.
- **Final Static Variables** — Use the `static` and `final` modifiers in the declaration statements for variables whenever possible, to make the variables act as constants. This practice minimizes the applet instance's size and improves performance.
- **Reusing Variables** — Reuse variables whenever you can. Each new variable you introduce consumes additional card resources.
- **Local Variables** — Local variables occupy space in the Java operand stack, so limit their use whenever possible.
- **Primitive Types** — Instead of creating objects from primitive types, use the primitive types themselves. If, however, you develop a class that encapsulates primitive types, the cost in card resources might outweigh any functionality you gain.
- **Class Hierarchy** — Calling classes within classes takes up program space, so keep the class hierarchy simple. You can do this by using class hierarchies in the prototype program, then compress the hierarchy to the bare essentials before you convert the code to an applet.
- **Arguments** — Use as few arguments as possible.
- **Calling Tree** — Keep the calling tree shallow. Although calling methods within methods allows you to use code in a modular fashion, it also consumes extra card resources.
- **Cleanup** — Edit your completed program to remove unused variable definitions and operations. Before you convert programs into applets, review the classes and remove any methods that are not called. Make sure reused code does not include unnecessary variables or operations.

- **Data Types** — Use shorts rather than bytes wherever possible. Shorts consume the same amount of card memory as bytes do and offer a wider range of values.

Here is an example from the SimplePurse program that is used in the tutorial (starting on page 148):

```
final static byte Deposit = ( byte ) 0x10
```

This statement declares the `Deposit` variable as a primitive data type `byte`, which can be declared only once and is fixed. The variable has the value `10h`.

Another approach is to declare all the constants in a separate file, then refer to them explicitly in the program. This technique is useful for reusing constant sets.

Smart Card Development in a Multi-Application Environment

When a smart card is a shared resource, possibilities occur for unexpected conflicts between programs. Here are some guidelines for developing smart card applications in a multi-application environment:

- Disable other smart card applications. For example, do not install components such as the CSP, or PKCS #11, and temporarily disable components such as Schlumberger Smart Card Logon, Entrust, and Smart Logon.
- Use common programming techniques for managing shared resources (for example, obtaining an exclusive connection or using transaction-based communications). The Toolkit provides a Begin/End Transaction lock button to assist you in obtaining an exclusive connection to the card, as explained below. Otherwise, keep in mind that you have the responsibility to block transactions to the card through your application when necessary, just as any other application in a multi-application environment like Windows—don't assume the state of the card is the same as when your application left the card.
- Be aware of security requirements and constraints for your application.

Begin/End Transaction Lock Button

If you are using the Toolkit, you can take advantage of the Begin/End Transaction lock button in the APDU Manager to stop other processes from communicating with the card.

A normal mode of operation for an application is as follows:

- 1** Begin a transaction to the card.
- 2** Select the desired applet and perform any other card state initialization required.
- 3** Exchange APDUs (for instance, to retrieve information to display to the user).
- 4** End the transaction.

When the application needs to communicate with the card again:

- 1** Begin a new transaction to the card.
- 2** Setup the card state again by selecting the desired applet and performing any other card state initialization required.
- 3** Exchange APDUs (for instance, to update information in the applet).
- 4** End the transaction.

To summarize, the Begin/End Transaction button will help when developing and communicating with the card from the Toolkit, but any applications will have to perform the same transaction operations as well. In a multi-application environment, when you do not have a transaction lock on the card, you should not assume that the card is in the same state as you left it—other applications might communicate with the card and change the state. Even if other applications are not running, the Smart Card Resource Manager has been known to reset the card.

See the *Cyberflex Access Software Development Kit User's Guide*, Chapter 7, APDU Manager, for details about how to use the Begin/End Transaction lock button to provide exclusive card access during application development.

Roadmap for Developing a Card Applet

To create a Java program to run on an Open Platform Cyberflex Access card, complete the following steps.

Step 1: Explore the Card's Possibilities and Develop Source Code

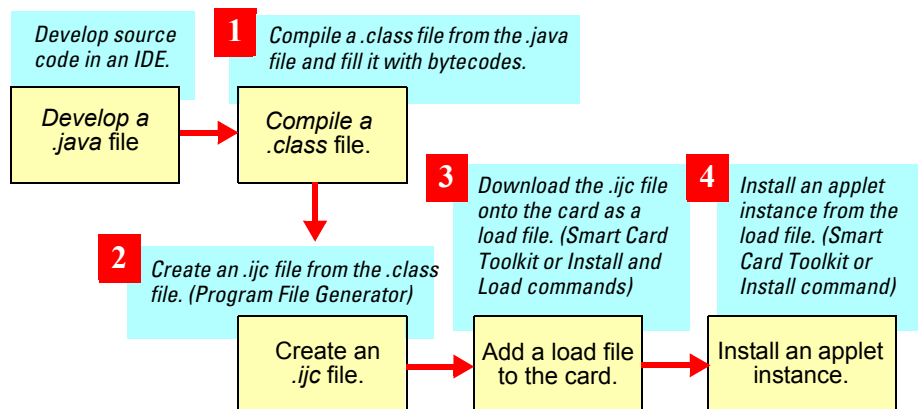
- 1 Install the Cyberflex Access Software Development Kit software and connect the card reader. (For details, see the installation section in the *Cyberflex Access Software Development Kit User's Guide*.)
- 2 Review the other related documents in the Cyberflex Access SDK library:
 - *Cyberflex Access Software Development Kit User's Guide* — Installation instructions, background information, detailed instructions about using the Cyberflex Access SDK applications and utilities, and information about secure email operations.
 - *Cyberflex Access Cards Programmer's Guide* — This document, which contains information about the card's command set, conventions, and procedures.
 - *Guide to SchlumbergerSema Smart Card Middleware* — Information to help you understand the foundation layer APIs that support card program development.
- 3 Work through the tutorial (page 146), and learn to use these software tools (described in the *Cyberflex Access Software Development Kit User's Guide*):
 - Smart Card Toolkit – A suite of tools you can use to work directly with the card and learn about the card's conventions and capabilities.
 - COVE – An application to help you personalize the card, work with keys, and set up secure logins with Microsoft's Graphical Identification and Authentication dynamic-link library (GINA).
- 4 After you have evaluated the card's possibilities and planned a card program, develop the source code for it.

NOTE *This list is oriented to experienced Java programmers who have some familiarity with smart cards, cryptography, and ISO 7816 standards. If you are just starting to program with Java, you also need to set up a Java programming environment on your host system and learn Java*

programming basics. If you are just beginning to work with smart cards, go to the websites listed on page xvi to learn about smart card standards and cryptography basics.

Step 2: Convert the Source Code into an Applet Instance

To prepare Java source code for use on an Open Platform Cyberflex Access card, complete the tasks shown in the following illustration.



- 1** Compile the Java source file into a class file. The card class file contains the program information converted to a stream of Java bytecodes. (For more information, see page 160.)
- 2** Convert the class file to a converted applet (program) file. To make this conversion for an Open Platform Cyberflex Access card, you can use the Program File Generator dialog box. (The Program File Generator dialog box is a graphical user interface to run the Sun Converter that is added to your host system when you install the Cyberflex Access SDK. It replaces the CAP File Generator that was included with the SDK 4.2.¹)

1. Card binary files created with the CAP File Generator have the .cap extension, but these card binary files are not the equivalent of .cap files created using the Sun converter. While CAP File Generator .cap files can be downloaded to some Open Platform Cyberflex Access cards, .cap files generated using the Sun converter cannot be downloaded to Open Platform Cyberflex Access cards. For details, see the section on “Compatibility Issues” in the *Cyberflex Access Software Development Kit User’s Guide*.

NOTE

The Sun Java Card Runtime Environment (JCRE) must be available when you use the Program File Generator. The Program File Generator was tested with JCRE version 1.3; we recommend that you install this version. The Program File Generator does not work with JCRE 1.1 or JCRE 1.4.

- 3 Add the program file data to the card as a load file in either of the following ways:
 - **Smart Card Toolkit** — Start the Program File Generator from the Make Card Applet dialog box (page 160).
or
 - **APDU commands** — Select the program file you want to download by calling an `Install:Load` command (page 93), then call a series of `Load` commands (page 107) to add the bytecodes to the card.
- 4 Install an applet instance on the card in one of these ways:
 - **Smart Card Toolkit** — Use the Create Instance dialog box (page 168).
or
 - **APDU commands** — Call an `Install:Install` or `Install:Install/Make Selectable` command (page 93).

You can create multiple applet instances that are based on a single program, but that have unique data sets. After you have installed an applet instance, you can select it and test its operations. (For information about selecting an applet instance with the `SelectApplication` command, see page 127. For information about selecting an applet instance in the Smart Card Toolkit, see page 172.)

Writing a Sample Card Applet

In this exercise, you develop a sample applet called SimplePurse. SimplePurse is a complete, ISO 7816-compliant smart card application that you can use as an electronic wallet to perform cash transactions with a smart card reader and terminal. The code for SimplePurse is included with the Cyberflex Access SDK, and is added to your system by default in this location:

*C:\Program Files\Schlumberger\Smart Cards and Terminals\
Cyberflex Access Kits\v4\Samples\SimplePurse*

The tutorial walks you through creating the SimplePurse code, then shows you how to convert, load, and test the applet instance on the Cyberflex Access Developer 32K card.

Overview of the Sample Applet

The SimplePurse applet stores and manages electronic cash, using these program-specific commands: `read_balance`, `deposit`, and `debit`. The cardholder gains access to the electronic wallet by presenting a Personal Identification Number (PIN) that the card verifies.

The applet instance can perform any of the following tasks:

- Verify a cardholder or card administrator's PIN code
- Read the card balance
- Add to or subtract from the card balance

Creating the Sample Applet

To create the sample applet, follow these steps:

- 1 Make sure a Java development tool is installed on your host system.
- 2 Enter the code found in the sample that follows (or use the sample copy that the Cyberflex Access SDK installation program added to your system, *SimplePurse.ijc*).

NOTE *You can run the `makeit.bat` file installed in each Sample directory (for example, `\Program Files\Schlumberger\Smart Cards and Terminals\Cyberflex Access Kits\v4\Samples\SimplePurse`) to compile the Java source code into a `.class` file. However, if you run `makeit.bat` from the default directory, you will overwrite the `*.class` file provided as part of the sample code; for that reason, we recommend that you copy the Sample directory to another location before running `makeit.bat`.*

The following sample source code includes comments in addition to those found in the example file itself.

```
//Wallet.java
//
// This file is adapted from the Wallet example provided
// by JavaSoft in the Javacard 2.0 documentation.
// Note: This file is ready for the Cyberflex Smart Card. If
// you wish to use the Cyberflex Simulator you must adjust
// the Java Classlibraries to 'SIMLibrary' rather than
// 'ClassLibrary' in your Compiler. Also switch the lines
// marked by !** below to compile for Simulator operation
// rather than card operation.
```

Import the package classes. Using the `import` keyword gives you access to the methods and definitions of the package's class files.

```
import javacard.framework.*;
import javacard.security.*;
```

Define applets as instances of a class extended from javacard.framework.

```
public class Wallet extends javacard.framework.Applet {
```

```
    /* Constants declaration */
```

```
    // code of CLA byte in the command APDU header
```

Define the first constant: the code of the class (CLA) byte in the command APDU header. The class byte identifies the applet. Next, set the codes for the instruction (INS) byte in the command APDU header.

```
    final static byte Wallet_CLA =( byte ) 0x03;
```

```
    final static byte Deposit = ( byte ) 0x10;
```

```
    final static byte Debit = ( byte ) 0x20;
```

```
    final static byte Balance = ( byte ) 0x30;
```

```
    final static byte Validate = ( byte ) 0x40;
```

```
    final static byte Encrypt = ( byte ) 0x50;
```

```
    final static byte PinChange = ( byte ) 0x60;
```

```
    // maximum number of incorrect tries before the PIN is
```

```
    // blocked
```

Set the first of the PIN object parameters.

```
    final static byte PinTryLimit = ( byte ) 0x03;
```

```
    // maximum size PIN
```

```
    final static byte MaxPinSize=( byte ) 0x04;
```

```
// status word (SW1-SW2) to signal that the balance becomes
// negative;
```

Create applet-specific status words.

```
final static short SW_NEGATIVE_BALANCE = ( short ) 0x6910;
final static short SW_WRONG_PIN = (short) 0x6300;
```

```
/* instance variables declaration */
```

Set the initial balance.

```
OwnerPIN pin;
static byte base_balance = 0x20;
Byte balance;
DESKey      key;
Signature   signature;
```

Define the buffers. The buffer handles the five-byte ISO-compliant APDU command format. The 4-byte header consists of CLA (command class), INS (command instruction), and P1 and P2 (command-specific parameters). The fifth byte (P3 or Lc) is the length of input or output data, which has been received or is expected in the APDU.

Define the constructor. Private constructor — an instance of the Wallet class is instantiated by its install method.

```
private Wallet( byte buffer[], short offset, byte length ) {
```

```
// It is good practice to allocate all the memory that an
// applet needs during its lifetime inside the constructor.

    pin = new OwnerPIN( PinTryLimit, MaxPinSize );
    balance = 0;
    key = ( DESKey )KeyBuilder.buildKey(
        KeyBuilder.TYPE_DES_TRANSIENT_RESET,
        KeyBuilder.LENGTH_DES,false );
    signature = Signature.getInstance(
        Signature.ALG_DES_MAC8_ISO9797_M1, false );
    signature.init( key, Signature.MODE_SIGN );
    if (buffer[offset] == 0) {
        register();
    }
    else {
        register( buffer,
            (short) (offset+1),
            (byte) (buffer[offset]) );
    }
}

// end of the constructor
```


The JCRE invokes the `install` method as the last step in the applet installation process.

```

public static void install( byte buffer[],
                           short offset,
                           byte length){

    // create a Wallet applet instance

        new Wallet( buffer,
                    offset,
                    length );

    }

    // end of install method


    // The JCRE calls the select method to inform the card that
    // this applet is selected and to performs the initialization
    // needed to process the following APDU messages.

    public boolean select() {

        // reset validation flag in the PIN object to false

        pin.reset();

        // returns true to JCRE to indicate that the applet is ready
        // to accept incoming APDUs

        return true;

    }

    // end of select method

```

After the JCRE successfully selects the applet, it dispatches incoming APDUs to the `process` method. The JCRE owns and maintains the APDU object, which encapsulates details of the underlying transmission protocol (T=0 or T=1, as specified in ISO 7816-3). The APDU object provides a common interface.

```
public void process( APDU apdu ) {
```

```
// APDU buffer - APDU object carries a byte array (buffer) to
// transfer incoming and outgoing APDU header and data bytes
// between card and CAD
```

```
byte buffer[] = apdu.getBuffer();
```

```
// Implement a select handler
```

```
if (selectingApplet()) {
    ISOException.throwIt(ISO7816.SW_NO_ERROR)
}
```

```
// Verify whether the applet can accept this APDU message.
```

When an error occurs, the applet can terminate the process and throw an exception that contains status words (SW1 SW2) to reflect the card's processing state.

If an applet does not catch an exception, the JCRE catches it. The main purpose of the process method is to perform the action the APDU specifies and return an appropriate response to the terminal. The INS byte specifies the type of action to be performed.

```
    if (buffer[ISO7816.OFFSET_CLA] != Wallet_CLA
        ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED)
    byte ins = buffer[ISO7816.OFFSET_INS];
    if (ins == Balance) getBalance(apdu);
    else if (ins == Debit) debit(apdu);
    else if (ins == Deposit) deposit(apdu);
    else if (ins == Validate) validate(apdu);
    else if (ins == Encrypt) encrypt(apdu);
    else if (ins == PinChange) PinChange(apdu);
    else ISOException.throwIt (ISO7816.SW_INS_NOT_SUPPORTED);
}
```

```
// end of process method
```

The APDU parameter object contains a field that specifies the amount to add to the balance. On receiving the APDU object from JCRE, the first 5 bytes are available in the APDU buffer. Their offsets in the APDU buffer are specified in the ISO class.

Communication between the card and CAD is accomplished by sending command and response APDU pairs. The response APDU for a deposit contains no data field. The JCRE sends the status words 9000h to indicate successful completion of the command.

Applet developers do not need to construct the response for successful command execution. When the JCRE catches an exception that signals an error during command processing, the JCRE uses status words to construct the response APDU.

```
private void deposit( APDU apdu ) {
    byte buffer[] = apdu.getBuffer()
```

```
// access authentication
```

```
    if ( ! pin.isValidated() )
        ISOException.throwIt (
            ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED );
```

```
// indicate that this APDU has incoming data and receive data
// starting from the offset ISO7816.OFFSET_CDATA.
```

```
    byte byteRead = ( byte )( apdu.setIncomingAndReceive() );
```

```
// it is an error if the number of data bytes read does not
// match the number in the Lc byte
```

```
    if ( byteRead != 1 )
        ISOException.throwIt( ISO7816.SW_WRONG_LENGTH )
```

```
// increase the balance by the amount specified in the
// data field of the command APDU and return successfully
```

```
    balance = ( byte )
        ( balance + buffer[ISO7816.OFFSET_CDATA] );
    return;
}
```

```
// end of deposit method
```

In the debit method, the APDU object contains a data field that specifies the amount to subtract from the balance.

```
private void debit( APDU apdu ) {
    byte buffer[] = apdu.getBuffer()
```

```
// access authentication
```

```
    if ( ! pin.isValidated()
        ISOException.throwIt(
            ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED );

    byte byteRead = ( byte )( apdu.setIncomingAndReceive() );
    if ( byteRead != 1 )
        ISOException.throwIt( ISO7816.SW_WRONG_LENGTH );

    ISOException.throwIt( ISO7816.SW_PIN_REQUIRED );
    byte numBytes = ( byte )( buffer[ISO7816.OFFSET_LC] );
    byte byteRead = ( byte )( apdu.setIncomingAndReceive() );
    if ( byteRead != 1 )

        ISOException.throwIt( ISO7816.SW_WRONG_LENGTH );
```

```
// balance cannot be negative

    if ( (byte)( (byte) balance - (byte)
        buffer[ISO7816.OFFSET_CDATA] ) < (byte) 0 )
        ISOException.throwIt( SW_NEGATIVE_BALANCE );

    balance = (byte) ( balance - buffer[ISO7816.OFFSET_CDATA] )
}

// end of debit method
```

The getBalance method returns the balance in the data field of the response APDU.

Because the data field in the response APDU is optional, the applet must explicitly inform the JCRE of the additional data. The JCRE uses the data array in the APDU object buffer and the proper status words to construct a complete response APDU.

```
private void getBalance( APDU apdu ) {

    byte buffer[] = apdu.getBuffer()

// access authentication

    if ( ! pin.isValidated() )
        ISOException.throwIt(
            ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED );

// inform system that the applet has finished processing
// the command and the system should now prepare to
// construct a response APDU which contains data field

    apdu.setOutgoing();
```

```
// indicate the number of bytes in the data field

    apdu.setOutgoingLength( (byte)1 );

// move the data into the APDU buffer starting at offset 0

    buffer[0] = ( byte ) balance;

// send 1 byte of data at offset 0 in the APDU buffer

    apdu.sendBytes( (short)0, (short)1 );
}

// end of getBalance method
```

Smart cards can use a PIN to protect data from unauthorized access. A PIN records the number of unsuccessful verification attempts since the last successful PIN verification. The card becomes blocked if the number of unsuccessful attempts exceeds the maximum number of allowed tries. After the applet is successfully selected, the PIN must be validated before any other instructions can be performed on the applet.

```
private void validate( APDU apdu ) {
    byte buffer[] = apdu.getBuffer();
```

```
// retrieve the PIN data required to be validated
// the user interface data is stored in the APDU data field

byte byteRead = ( byte )( apdu.setIncomingAndReceive() );

// validate the user interface and set the validation flag in
// the user interface object to be true if the validation
// succeeds.
// if user interface validation fails, PinException would be
// thrown from pin.check() method.

if ( !pin.check(buffer, ISO7816.OFFSET_CDATA, byteRead) )
    ISOException.throwIt(SW_WRONG_PIN);

key.setKey( buffer, (short)0 );
}

// end of validate method

private void encrypt( APDU apdu ) {

    byte buffer[] = apdu.getBuffer();
```



```

// get 4 bytes of data and return the encrypted result.
// access authentication

    if ( ! pin.isValidated() )
        ISOException.throwIt(
            ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED );

    short byteRead = apdu.setIncomingAndReceive();

    signature.sign( buffer,
                    (short)5,
                    byteRead,
                    buffer,
                    (short)0 );
    apdu.setOutgoingAndSend( (short)0, (short)8 );
}
}

// End of class Wallet

```

Converting Source Code to a Card Application

Step 1: Compile the Source Code

Use a standard Java compiler to compile and debug the source code. The Java Virtual Machine (JVM) requires debugging information, so you must use the debugging option (-g) when you compile.

Before you compile the code, verify that the compiler's class path is set to the Cyberflex Access class library directory on the host system. Make sure the path to the Cyberflex Access library files is set correctly in your development environment.

As an example, to set the library path in Visual J++ with the Microsoft Developer Studio V6.0, choose the `Wallet` properties/Classpath to point to the class libraries. Card class files are located in this path:

```
C:\Program Files\Schlumberger\Test and Transactions\  
Cyberflex Access Kits\v4\ClassLibrary
```

Step 2: Convert the Class File to a Program File

After you compile the code as a class file, you can use the Program File Generator to convert the class file to a program file, which puts the load file in a form the JVM can understand. After you prepare the program file, you can add it to the card as a load file and instantiate the load file. You do not need to be connected to a card to convert a class file to a program file.

The Program File Generator dialog box is a graphical user interface for the Sun Converter from Sun Microsystems, Inc. It is added to the host system when you install the Cyberflex Access SDK, and is also available in the Java Card 2.1 Development Kit. You can run the Sun Converter from the command line, as documented in the Java Card 2.1 Development Kit, available from Sun at <http://java.sun.com/products/javacard>.

NOTES

- *The Cyberflex Access SDK also supports command-line use of the Program File Generator, as explained in the Cyberflex Access Software Development Kit User's Guide.*
- *The Program File Generator replaces the CAP File Generator, which was in Cyberflex Access SDK 4.2.*

- *The Sun Java Runtime Environment (JRE) must be available when you use the Program File Generator. This Program File Generator was tested with JRE version 1.3; we recommend that you install this version. The Program File Generator does not work with JRE 1.1 or JRE 1.4.*
- *The latest version of the JAVA Converter (either 1.2 or 1.3) requires the package and applet AID RID (the first five bytes) values to be identical. This was not a requirement in prior versions, so you might have to modify your package, applet, or applet instance AID values to conform to this restriction.*

Using the Program File Generator

To use the Program File Generator dialog box, follow these steps:

- 1 If it is not already running, start the Smart Card Toolkit and insert a card:
Click the **Start** button on the Windows taskbar; then select **Programs** → **Schlumberger Smart Cards and Terminals** → **Cyberflex Access SDK 4.4** → **Cyberflex Access Toolkit 4.4**.

The Schlumberger Smart Card Toolkit window appears. The reader connects to the card. The Card Manager pane of the main window displays the reader. If you expand the reader plus (+) box, the card appears.

- 2 If you have just started a card session, you can establish a secure channel by verifying the AUTH, MAC, and KEK keys. On a new Open Platform Cyberflex Access card, you can verify these keys through the **Open Platform** dialog on the **Key Manager** menu—see the *Cyberflex Access Software Development Kit User's Guide* for detailed steps. Remember, however, that you do not have to be connected to a card to convert a class file to a program file.

After the host system and Card Manager (or other security domain) set up the secure channel, the main window displays the contents of the card in the Card Manager pane.

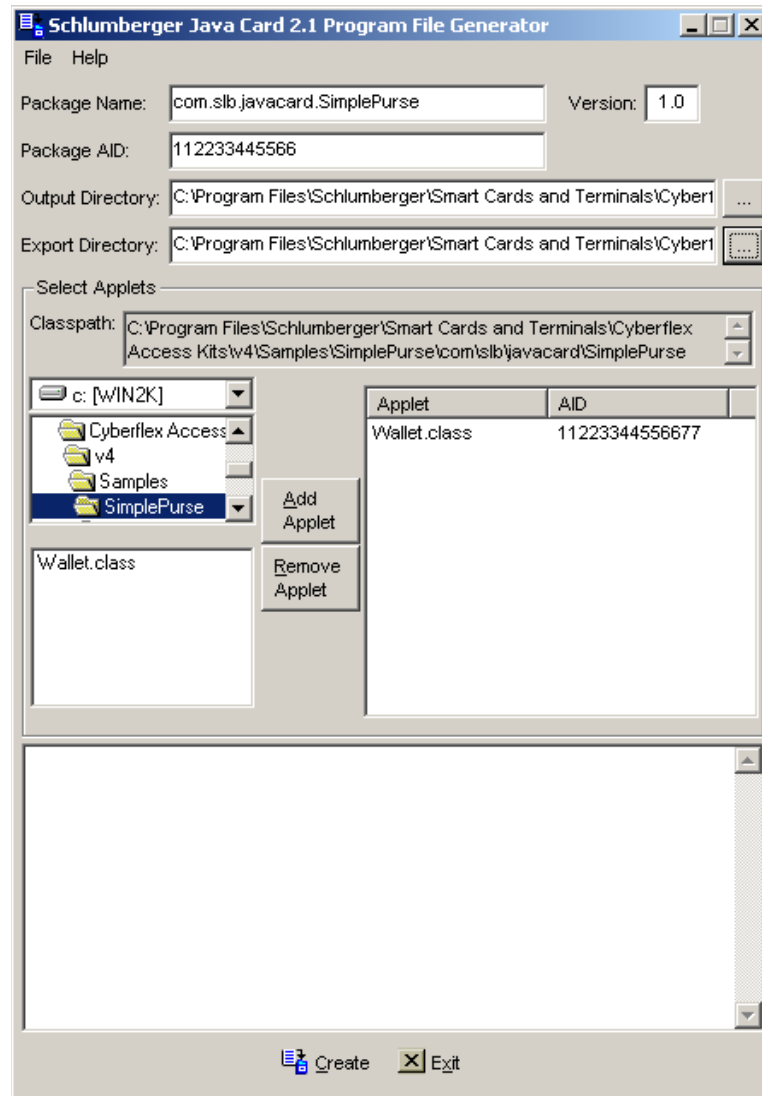
NOTE

On a new Open Platform Cyberflex Access card, the key values in the default key set are:

- **AUTH key** – 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4Fh
- **MAC key** – 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4Fh
- **KEK key** – 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4Fh

- 3 Click the **Make Card Applet** button in the Smart Card Toolkit toolbar, or select **Make Card Applet** from the **Tools** menu.
- 4 Select the **Program File Generator** option.

The Program File Generator dialog box appears. The example shows the dialog box with some information added.



- 5 Enter the following information in the top of the Program File Generator dialog box:
 - **Package Name** — The name for the fully-qualified Java package contents. (A Java package is a group of related classes and interfaces within a single directory.) The name used for the output file name is the last entry of the Package Name field.

NOTE *Package Name must be specified before you select the directory in which the applet .class file is located.*

- **Version** — The package version. Revise this value if necessary.
- **Package AID** — The AID you specify for a load file (or package) that can be downloaded from the program file. The package AID must be a hexadecimal value 5–16 bytes long. (You will use this package AID to download the program file to the card as a load file—in the Create Program dialog box, as described on page 166. COVE will automatically fill in the package AID from an .ijc or .cap file when creating the program file to download the applet to the card.)

NOTE *The first 5 bytes of the package AID and the first 5 bytes of the applet AID must be identical.*

- **Output Directory** — Destination for the output file (the .ijc file). You can use the keyboard to enter a path in the Output Directory box or click the ellipsis (...) button and use the file navigation dialog box to select a directory.
 - **Export Directory** — Directory that contains the Java export files the Cyberflex Access SDK installation program adds to the host system. The location for the installed export files appears automatically in the Export Directory box. To specify other Java export files, click the ellipsis (...) button and locate the directory in the file navigation dialog box that appears.
- 6 Select a class file to add to the program file. Under **Select Applets**, navigate to the .class file for the applet you want to create.
 - 7 Highlight the correct class file name, and click the **Add Applet** button. The Applet AID dialog box appears, which you use to specify an AID 5-16 bytes long for the class in the load file.

NOTE *The first 5 bytes of the package AID and the first 5 bytes of the applet AID must be identical.*

- 8** Enter the AID for the class file, then click **OK**. (Note that the AID you specify here will be the one you use to instantiate the applet in the Create Instance dialog box, as shown on page 169.)

The Applet AID dialog box closes. The AID and class file name appear in the list next to the class navigation pane.

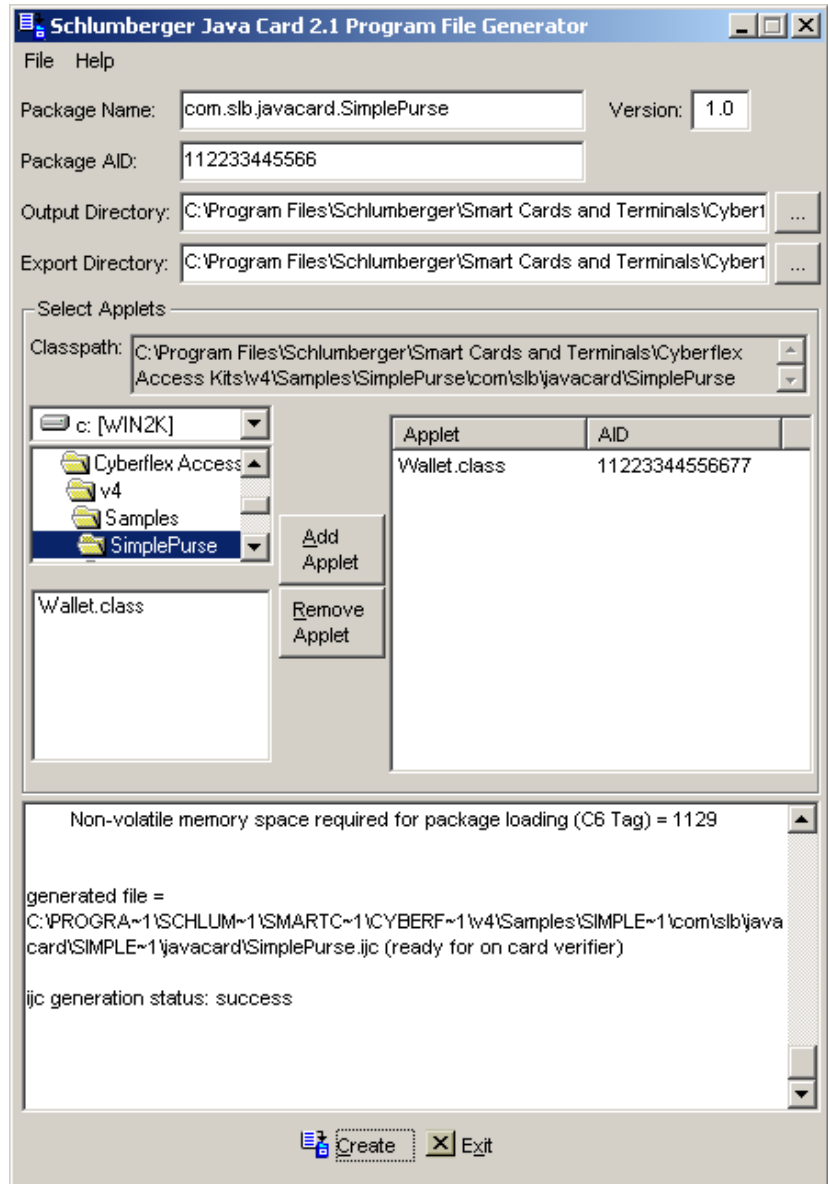
Modifying the List of Class Files to Add to the Program File

- If you want more class files in the program file, repeat the last two steps until all the needed class files appear in the list area. (You will be able to create instances from each class file in the program file.)
 - If you decide to remove a class file before you create the program file, click the **Remove Applet** button.
- 9** When all the information in the dialog box is correct, click the **Create** button to create the new *.ijc* file.

The conversion begins, and, unless the operation is interrupted, the status of the ongoing operation appears in the status pane in the Program File Generator dialog box (at the bottom of the dialog box). When the conversion is complete, the status pane fills with information.

NOTE *If the Sun converter detects a problem in the class file, it logs a system error. Error message logs are generated in the base directory, for example, SimplePurse, where the classes are located in SimplePurse\com\slb\javacard\SimplePurse. The log files are JCANormal.txt, JCAError.txt, IJCNormal.txt, and IJCError.txt.*

The following example shows the status pane for a program file produced from *Wallet.class*. To view all of the status data, use the scroll bars.



The status pane displays several types of information, including:

- **Sun converter version** — Version number of the converter included in the Cyberflex Access SDK. This converter produces a program file that can be downloaded to the smart card.
- **Sun CAP File Builder and SchlumbergerSema tool versions** — Version numbers of the tools that process the JAR and produces the *.ijc* file.
- **Program file components** — Components of the program file (taken from the package), with the number of bytes each component occupies.
- **Number of class files/packages** — Number of class files (applets) included in the program file, and number of packages imported.
- **Destination of the output file** — Complete path to the program file.
- **Argument to the C6 tag** — The argument you furnish to the C6 tag if you download the program file by using APDU commands. (If you use the Smart Card Toolkit to add the load file to the card, you do not need to supply this data.)

The next step is to download the program file as a load file on the card.

Step 3: Download the Program File as a Load File

Next, you add a load file to the card. (The load file is also sometimes called a program file or package.) The load file contains the bytecodes for the class or classes in the program file. Any applet instance you create from the load file refers back to the load file class information at runtime.

You can add a load file to the card in either of these ways:

- **APDU commands** — Select the program file you want to download by calling an `Install:Load` command (page 93), then call a series of `Load` commands (page 107).
- or*
- **Smart Card Toolkit** — Use the Create Program dialog box in the Smart Card Toolkit (as described in this topic).

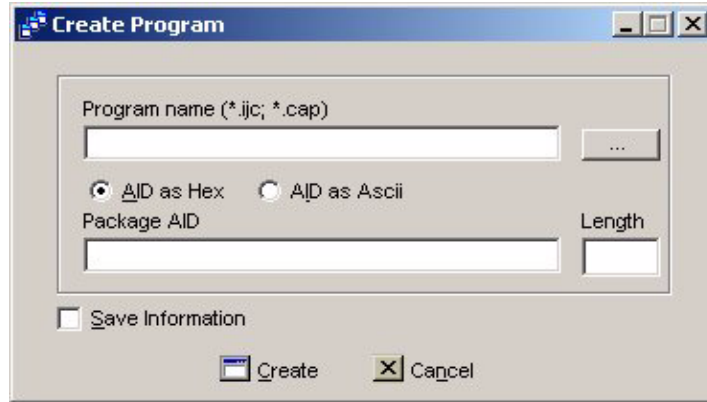
Using the Smart Card Toolkit to Download a Load File

- 1 If it is not already running, start the Smart Card Toolkit and establish a secure channel.

The Schlumberger Smart Card Toolkit window appears, with the card contents displayed.

- 2 Right-click the **Load Files and Libraries** folder icon in the Card Manager window, and select **New** → **Program File** from the pop-up menu that appears.

The Create Program dialog box appears, as shown below.



- 3 Enter the information for the *.ijc* file you want to create:
 - **Program name (*.ijc, *.cap) box** — Specify the *.ijc* file you want to use to create the load file, by using one of these methods:
 - Enter the full path to the *.ijc* file in the Program name box, or
 - Click the **Program name** ellipsis button (...).

The Select Card Applet to load dialog box appears, which you can use to locate and select the *.ijc* file you want to download to the card as a load file. When you have selected the file, click **Open**.

NOTE *Program files with the .ijc extension can be downloaded to all Open Platform Cyberflex Access cards. Program files with the .cap extension can be downloaded only to some Open Platform Cyberflex Access cards. See the Cyberflex Access Software Development Kit User's Guide for details.*

- **Package AID box** — The **Package AID** box automatically displays the AID for the load file (package), a unique hexadecimal value 5-16 bytes long. This AID must match the value you entered in the Package AID box of the Program File Generator dialog, as described on page 162.
- The **Length** box automatically displays the number of AID bytes.

- **AID as Hex** radio button — Make sure the **AID as Hex** radio button is selected. You must enter the AID in hexadecimal format.
 - **Save Information** check box — *Optional:* Select this option to save the information that appears in the Create Program dialog box. The saved information appears automatically whenever you redisplay the dialog box, even in a future card session with a different card inserted. The information remains in the host system database until you overwrite it by again clicking the Create button with the Save information check box selected.
- 4 When all the information is correct, click the **Create** button.
- The Card Manager (or other current security domain) calls the `Install:Load` and `Load` commands and downloads the load file with the specified parameters.

Step 4: Instantiate the Applet

Now you are ready to instantiate the applet. You can instantiate an applet instance in either of these ways:

- **APDU commands** — Call one of these commands (page 93):
 - `Install:Install` command — Install an applet instance without making it selectable.
 - `Install:Install/Make Selectable` command — Install an applet instance and simultaneously make it selectable or selected by default.
- **Smart Card Toolkit** — Use the Create Instance dialog box (as described in this topic). If you use the Smart Card Toolkit to instantiate the applet, it is installed as selectable.

Using the Smart Card Toolkit to Download a Load File

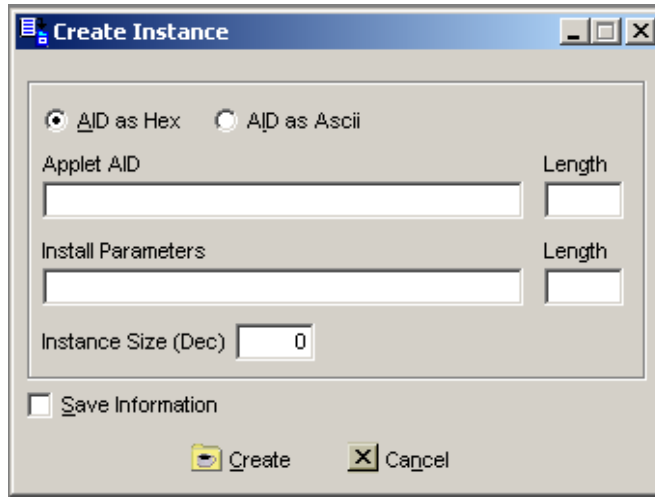
- 1 If it is not already running, start the Smart Card Toolkit and establish a secure channel, as described beginning on page 161.

The Schlumberger Smart Card Toolkit window appears with the card contents displayed.
- 2 In the Card Manager pane of the main window, right-click the load file you want to instantiate.

A pop-up menu appears.

3 Select **New** → **Instance** from the pop-up menu.

The Create Instance dialog box appears, as shown in the following illustration.



4 Specify the information for the instance:

- **Applet AID** box —Enter the AID you want the applet instance to have. The AID must be a unique value 5-16 bytes long. (Note that the instance AID must match the class AID that was specified in the program file—for example, as shown on page 162.)
The **Length** box automatically displays the number of AID bytes as you enter the AID.
- **AID as Hex / AID as Ascii** radio buttons —Select one of these options to specify the format of the AID value (hexadecimal or ASCII). If you enter the AID in ASCII format, make sure it is the equivalent of the hexadecimal AID specified during program file generation.
- **Install Parameters** box —Enter any application-specific parameters that you want the **Install** command to pass to the applet instance.
The **Length** box automatically displays the number of bytes as you enter Install parameter data.

- **Instance Directory Size (Dec)** box —Enter the amount of EEPROM needed for the applet instance and its associated files and headers. Enter the size in bytes, expressed in decimal format. To determine this size, refer to the program element sizes defined in the Java Card 2.1 specifications. You must also add space for the header. The amount of header space required is highly variable, depending on your program. For this reason, you may have to experiment to determine the total amount of EEPROM the program requires.
- **Save Information** check box — *Optional:* Select this option to save the information that appears in the Create Instance dialog box. The saved information appears automatically whenever you redisplay the dialog box, even in a future card session with a different card inserted. The information remains in the host system database until you overwrite it by again clicking the Create button with the Save information check box selected.

5 Click the **Create** button.

The Card Manager (or other security domain that is exchanging data with the host system in the current secure channel) calls the `Install:Install` command and creates the specified instance from the load file. The Schlumberger Smart Card Toolkit window is refreshed automatically and displays the applet instance under the Instance folder in the Card Manager window.

NOTE

The applet must have an `install` method, and the `install` method must call the `register` method. If these methods are absent, the card registry has no record of the applet instance, so the card cannot locate the instance to select it or send it commands.

Sending APDU Commands to the Sample Application

Now that you have an applet instance on the card, you can select it and send it APDU commands. Using the APDU Manager, you can test whether the APDU commands yield the expected results and verify that the applet instance runs as it should on the card.

Sending Commands to the Application in the APDU Manager Window

You have two ways to send APDU commands to the SimplePurse applet instance by using the Smart Card Toolkit's APDU Manager window:

- Use the following steps to enter APDU commands manually in the APDU Manager and save the functions to the function list. This approach will help you prepare to work with your own applet.
- Load the function file added to your system by the installation program. This approach makes it possible to complete the exercise without entering any values or saving functions.

To load the function file, select **File** → **Load List** from the APDU Manager window's menu bar. A file navigation dialog box opens, which you use to find and select the *Wallet.fcn* file. This file is located by default in the following path:

C:\Program Files\Schlumberger\Smart Cards and Terminals\Cyberflex Access Kits\v4\Samples\SimplePurse\com\slbjavacard\SimplePurse

Once you select the *Wallet.fcn* file, click **Open**. This action returns you to the APDU Manager window with the Wallet function list loaded. (The Wallet function list contains commands for testing the SimplePurse applet instance.

Step 1: Setting Up the APDU Session

Begin by setting up the card session:

- 1 If it is not already running, start the Smart Card Toolkit and establish a secure channel, as described beginning on page 161.

The Schlumberger Smart Card Toolkit window appears.

- 2 Next, you must select the application that will receive the commands. (To use the APDU Manager window to send commands to an application other than the Card Manager, you must first select the application.)

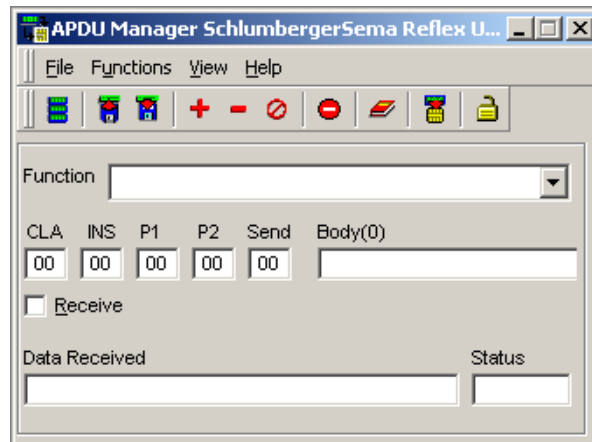
To select the SimplePurse applet instance, right-click its icon (the icon that appears in the Schlumberger Smart Card Toolkit window), and select **Select Application** from the pop-up menu.

NOTE

You can also make an applet instance the default application, so it is automatically selected by default. For an applet instance you have already added to the card, make it selected by default by using the Install command in Make Selectable mode (page 93) or the SetStatus command (page 133).

- 3 Display the APDU Manager window by clicking the **APDU Manager** button on the Card Manager window's toolbar (or by selecting **Tools** → **APDU Manager** from the menu bar).

The APDU Manager window appears.



Step 2: Reading the Card Balance

In this step, you attempt to read the card balance.

- 1** Enter the hexadecimal values 80 30 00 00 in the Class (**CLA**), Instruction (**INS**), Parameter 1 (**P1**), Parameter 2 (**P2**) fields, respectively.
- 2** Select the **Receive** checkbox to indicate that you are expecting to receive output data from the card.

The APDU Manager window changes to Receive mode. (The Length box is added, which you use to specify the number of bytes of response data you expect the card to return.)

- 3** In the **Length** box, enter 01, to specify that you expect the applet instance to return one byte of data.
- 4** In the **Function** box, enter the name View Balance.
- 5** Click the **Save Function** button in the APDU Manager window toolbar.
The function is added to the current function list. Now you can enter the command's APDU values by selecting its name from the drop-down Function list.
- 6** Click the **Send** button.

The APDU Manager sends the command to the card.

The APDU Manager displays 6982 in the Status box, which indicates that the attempt failed due to insufficient permission. The SimplePurse applet requires you to authorize yourself by presenting the PIN before you can view the balance. The PIN value is initialized to 00000000 in the applet instance.

Step 3: *Verifying the PIN*

In this step, you enter the `Verify PIN` command, which verifies the PIN for the SimplePurse applet. Once you have done this, you can read the card balance.

- 1 Clear the **Receive** check box, so the APDU Manager window reverts to Send mode.
- 2 Enter `80 40 00 00` in the Class (**CLA**), Instruction (**INS**), Parameter 1 (**P1**), Parameter 2 (**P2**) fields, respectively.
- 3 In the **Body** section, enter the PIN value, `00000000`. (Use the Body field to enter the command input data.)

The APDU Manager fills in the amount of input data in the **Length** box automatically (`04h`, or 4 bytes).

- 4 In the **Function** box, enter the name `Verify PIN`, and click the **Save Function** button in the APDU Manager window toolbar.

The function is added to the current function list.

- 5 Click the **Send** button.

The APDU Manager sends the command to the card to verify the PIN. The card returns the status words `9000h` to indicate success.

- 6 Select the `View Balance` command from the drop-down **Function** list, and click the **Send** button.

The APDU Manager sends the command to the card.

The Data Received box displays the value `20`, the SimplePurse balance in hexadecimal format, (`32` in decimal format). This is the value you initialized in the SimplePurse applet when you instantiated it. (Refer to the SimplePurse source code listing and look for the initialization of `base_balance`.)

Step 4: Debiting the Balance

In this step, you debit a value from the card balance.

- 1** Make sure the **Receive** check box is cleared, and enter 80 20 00 00 01 in the **CLA**, **INS**, **P1**, **P2**, and **Length** (P3) boxes, respectively. Enter 04 in the **Body** box. (The Body box contains the input data—the amount you are subtracting from the balance.)
- 2** In the **Function** list, enter the name **Debit**, and click the **Save Function** button in the APDU Manager window toolbar.
The function is added to the current function list.
- 3** Click the **Send** button.
The APDI Manager sends the command to the card, and you receive the status words 9000h to indicate success.
- 4** Select the **View Balance** command from the drop-down **Function** list.
- 5** Click the **Send** button.
The Data Received box displays a balance of 1Ch (28 decimal). (20h – 04h = 1Ch.)

Step 5: Crediting the Balance

In this step, you add value to the card balance.

- 1** Make sure the **Receive** check box is cleared, and enter 80 10 00 00 01 in the **CLA**, **INS**, **P1**, **P2**, and **Length** (P3) boxes, respectively. Enter 07 in the **Body** box. (The Body box contains the input data—the amount you are adding to the balance.)
- 2** In the **Function** list, enter the name **Deposit**, and click the **Save Function** button in the APDU Manager window toolbar.
The function is added to the current function list.
- 3** Click the **Send** button.
The APDU Manager sends the command to the card, and you receive the status words 9000h to indicate success.

- 4 Check the balance by recalling the `View Balance` command and sending it to the applet instance.

The balance the card returns is 23h (35 decimal). ($1\text{Ch} + 07\text{h} = 23\text{h}$.)

Step 6: Saving the APDU Function List

Before you close the APDU Manager window, save the function list for future use:

- 1 Click the **Save List** button on the APDU Manager toolbar.
The Select save file dialog box appears.
- 2 Locate the directory you want to use for function lists, enter a name for the function list, and click **Save**.



Java Card 2.1.1 API Support

The Open Platform Cyberflex Access cards fully support the Java Card 2.1.1 API specification. For information about Java Card classes and interfaces, see the Java Card 2.1.1 API specification and the Java Card Runtime Environment (JCRE) 2.1 specification (available at <http://java.sun.com/products/javacard.>)

Some elements in the the Java Card 2.1.1 API specification are not included in the current implementation of the Open Platform Cyberflex Access cards. This topic describes the algorithms that are not currently included, along with any other notable options or exceptions.

NOTE *It is important to use the original class when you compile your program.*

Support for the `javacard.framework` Package

The Open Platform Cyberflex Access cards implement the `javacard.framework` package with no notable exceptions.

Support for the `javacard.security` Package

The following list shows the implementation restrictions for the `javacard.security` package.

- **DSAKey** interface — Not implemented
- **DSAPrivateKey** interface — Not implemented
- **DSAPublicKey** interface — Not implemented

- **KeyBuilder** class — If you attempt to build a key with an algorithm that is not currently included on the card, the `buildKey` method fails and returns an error message.
 - **MessageDigest** class — The `ALG_RIPEMD160` algorithm is not currently included.
 - **Signature** class — The following Signature class algorithms are not currently included:
 - `ALG_DES_MAC4_ISO9797_M1`
 - `ALG_DES_MAC4_ISO9797_M2`
 - `ALG_DES_MAC4_NOPAD`
 - `ALG_DES_MAC4_PKCS5`
 - `ALG_DES_MAC8_PKCS5`
 - `ALG_DES_SHA`
 - `ALG_RSA_MD5_RFC2409`
 - `ALG_RIPEMD160_ISO9796`
 - `ALG_RIPEMD160_PKCS1`
 - `ALG_RSA_SHA_ISO9796`
 - `ALG_RSA_SHA_RFC2409`
- `getInstance` method — The software does not check values for the `externalAccess` parameter in the `javacard.security.Signature.getInstance` method.

Support Status for the `javacardx.crypto` Package

The classes and interfaces in the `javacardx.crypto` package are implemented with the following exceptions:

- **Cipher** class — The following Cipher class algorithms are not currently included:
 - `ALG_DES_CBC_PKCS5`
 - `ALG_RSA_ISO14888`
 - `ALG_RSA_ISO9796`
- **KeyEncryption** interface — The `KeyEncryption` interface is not currently implemented.



Command Conventions and APDU Basics

This section describes the Application Protocol Data Unit (APDU) format specified by ISO 7816 and the command conventions used in this document to describe the Open Platform Cyberflex Access card commands.

The command description tables contain these components:

CLA	INS	P1	P2	Lc	Le	Mode
-----	-----	----	----	----	----	------

Data sent to the card is in command APDU format, which consists of a mandatory header (the CLA, INS, P1, and P2 components) and an optional body (the Lc, input data, and Le components). Data returned to the host is in response APDU format, which consists of an optional body (the response data) and mandatory trailer (the SW1 and SW2).

Command Description Table Components

CLA	<p>Class of the command (1 byte). The class byte value is specific to the card version and command security level. Open Platform Cyberflex Access cards use these class byte values:</p> <ul style="list-style-type: none"> • 00h – Command that conforms to ISO standards • 80h – Command that conforms to Open Platform standards • 84h – Command that conforms to Open Platform standards and that requires a secure channel <p>If you order custom-manufactured cards, you can specify other command class values. Once manufacturing is complete, command class values cannot change.</p>
INS	<p>Instruction identifier of the command (1 byte). The CLA and INS bytes together uniquely identify each command.</p> <p>If the class and instruction bytes identify the command as a <code>SelectApplication</code> command, it is forwarded to the JCRE. All other APDU commands are sent to the currently selected application.</p>
P1, P2	Input parameters (1 byte each), which contain command-specific data.
Lc	Input parameter (1 byte) that specifies the number of input data bytes the host sends the card as part of the command. The Lc value is typically an explicit value between 01–FFh. For example, if you send the card a command with 2 bytes of input data, the Lc value is 02h.
Input Data	Command-specific data the host application sends the card, included with commands in send (S) or send/receive (S/R) mode.
Le	<p>Input parameter (1 byte), that specifies the number of data bytes the host expects the card to return as response data. The Le value specifies the length of the return data field, regardless of the amount of available data. If the available data does not fill the specified Le field, the terminal byte(s) are null. Available data that does not fit in the specified Le field is not returned.</p> <p><i>Note: For more information about command input/output formats, see “ISO Protocol Basics,” on page 182.</i></p>

Mode	<p>Send/receive mode of the command, which corresponds to the command cases specified by ISO 7816-4:</p> <ul style="list-style-type: none"> • — (None): Send no input data and receive no response data — Case 1. Case 1 command components are CLA + INS + P1 + P2. The card returns status words (SW). • S: Send input data to the card — Case 2. Case 2 command components are CLA + INS + P1 + P2 + Lc + input data. The card returns SW. • R: Receive response data from the card — Case 3. Case 3 command components are CLA + INS + P1 + P2 + Le. The card returns SW and response data. • S / R: Send input data to the card, and receive response data — Case 4. Case 4 command components are CLA + INS + P1 + P2 + Lc + input data + Le. The card returns SW and response data. <p><i>Note: The input/output APDU protocols are illustrated on page 182.</i></p>
-------------	---

Other Command Components

Response Data	Data the card returns to the host application, if the command mode is receive (R) or send/receive (S/R). The card returns response data in response to an internally called <code>GetResponse</code> command.
SW1, SW2	<p><i>Status words:</i> Two bytes the card returns to the host application at the end of a command transaction, which indicate whether the command succeeded or failed. If the command failed, the status words typically indicate the type of failure. The status word values presented in this guide conform to ISO, JVM, and GSM standards. Under certain conditions, cards can also return other types of status words (depending on the active applet instance). (For more information about error codes, see “Status Words,” on page 185.)</p>

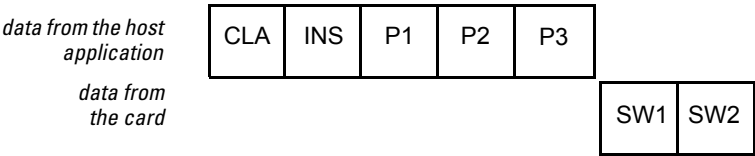
TPDU Protocol

APDU data is transmitted between the host and card by a transport protocol, or transmission protocol data units (TPDUs), which conform to the ISO 7816-3 specification. All Open Platform Cyberflex Access cards can use the T=0 protocol—an asynchronous, byte-oriented, half-duplex transmission protocol in which the smallest data unit that can be transmitted is a byte. Some Open Platform Cyberflex Access cards also support the T=1 protocol, which is an asynchronous, block-oriented, half-duplex transmission protocol. See “Supported Cards,” on page xi, for a list of supported cards.

ISO Protocol Basics

This topic describes T=0 ISO input and output command formats for the exchange of APDU data between the card and host application (or terminal). This information will help you avoid ISO protocol errors. The illustrations that follow show data bytes are exchanged between the host application and card.

Case 1: No Input or Output

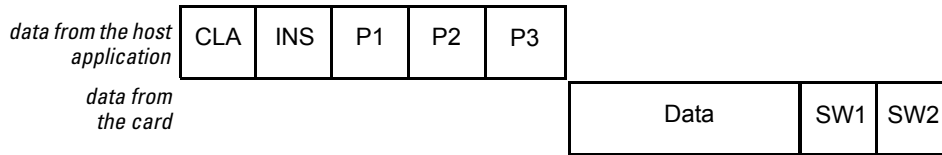


In this type of command, no input data or output data are exchanged between the host application and card. These events occur:

- 1 The host application sends the card a command with no input data, and a P3 value of 00h (0 bytes). (P3 typically specifies the length of input or output data.)
- 2 The card returns two bytes of status word data that indicate the operation’s success or failure. If the operation fails, the status words may identify the reason for the failure.

Commands with no input or output data are in neither Send nor Receive mode.

Case 2: Receive Mode



In a Receive mode command, the host application sends no input data with the command, but receives output data from the card. These events occur:

- 1 The host application sends the card a command that is expected to generate or locate output data. The command's P3 value indicates the expected length (number of bytes) of the output data.

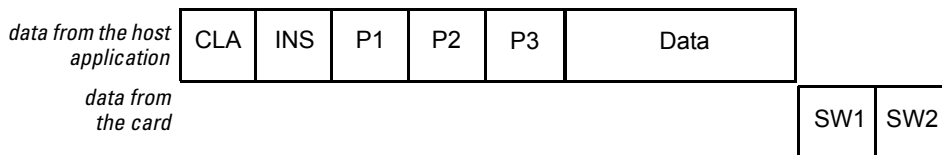
The P3 value is either:

- An explicit value between 01h and FFh, which specifies the actual number of data bytes the host application expects the card to return.
- A value of 00h — The convention for retrieving the maximum amount of response data, 256 bytes.

- 2 The card returns the output data and status words.

Example: PutData command on page 116 (if no DAP is included)

Case 3: Send Mode

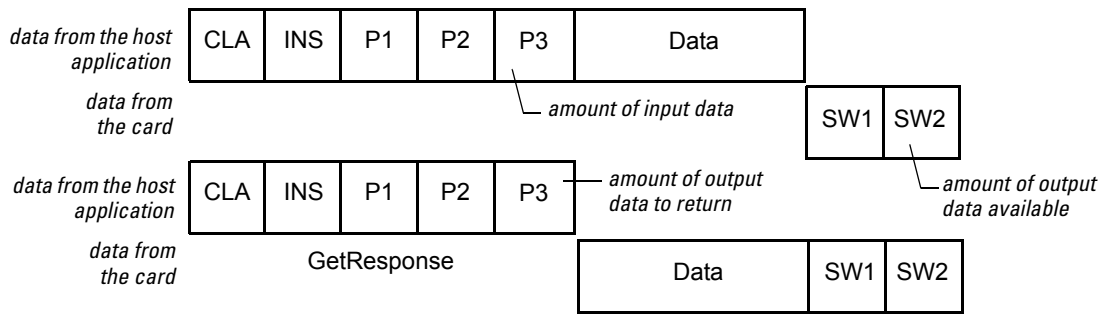


In a Send mode command, the host application sends input data with the command, and receives no output data from the card. These events occur:

- 1 The host application sends the card a command that requires input data, but will not produce any output data. P3 specifies a *length* (number of bytes of input data).
- 2 The card returns status words.

Example: ExternalAuthenticate command on page 70

Case 4: Send/Receive Mode



In a Send/Receive (S/R) mode command, the host application sends input data with the command, and retrieves output data through an automatic or voluntary follow-up `GetResponse` command. These events occur:

- 1 The host application sends the card a command with input data, which is intended to produce output data.
- 2 The card returns status words (SW1 and SW2) that indicate the initial operation’s success or failure. If the operation succeeds, SW2 indicates the amount of data available for return.
- 3 In T=0, the host follows with a `GetResponse` command, in which the P3 value is equal to or less than the amount of data reported in SW2. In T=1, return data is automatic as part of the response block.

NOTE Refer to ISO 7816-4 for more information about the APDU transport protocols T=0 and T=1.

- 4 The card returns the amount of data specified in P3.

Example: `GetStatus` command on page 80



B

Status Words

This appendix is an overview of status words (error codes) that the Open Platform Cyberflex Access cards can return in response to APDU commands. The test cards included with the Cyberflex Access SDK return ISO status codes, whose values may not match those in earlier releases.

Status Words the Card Returns for APDU Commands

The following table shows status words the card commands can return:

ISO Status	Meaning
6283	The Card Manager is locked (<code>SelectApplication</code>).
6300	Authentication of the host cryptogram failed.
6310	More data is available for return than is specified in the <code>Le</code> value.
6400	Technical problem that has no specified diagnosis.
6581	Memory failure.
6700	The specified length of the input data (<code>Lc</code>) is incorrect.
6981	No key is specified (<code>GetResponse</code> , called internally).
6982	Security status not satisfied. For example, MAC verification failed, the authentication key is locked, or the current security domain requires DAP verification and no verification data was included with the command.
6983	The key is blocked (<code>GetResponse</code> , called internally).

ISO Status	Meaning
6985	A requirement for using the command is not satisfied. For example: <ul style="list-style-type: none"> • Command issued outside of a secure channel. • Current application does not have the required application privilege or life cycle state. • The required preceding command was not present. • The object to delete is referenced by another object on the card.
6987	The MAC or other verification data is missing (Install)
6999	Application selection failed (SelectApplication).
6A80	Invalid or inconsistent input data, including input data that is inconsistent with a command header parameter, an LV/TLV-format elements in the input data that are not self-consistent. For example: <ul style="list-style-type: none"> • Incorrect number of padding bytes, incorrect key used for encryption, or the specified key set or key index value is invalid. • Referenced AID is not found in the card registry or package, or the newly specified AID already exists in the registry. • Inappropriate application privilege byte value (<i>installing security domain</i>), or card already has a default selected application (<i>specifying default selected application</i>). • First block of input data for a load file is not preceded by the correct tag and/or valid length, or the load file refers to a nonexistent package.
6A81	Target is locked (SelectApplication).
6A82	Registry contains no valid application (or no additional valid application) with the specified AID (SelectApplication).
6A84	Insufficient EEPROM memory available to add the object to the card.
6A86	Incorrect or unsupported value is specified for P1, P2, or both.
6A88	Data referred to in P1, P2, or both is not found.
6D00	Unsupported value entered for the INS byte.
6E00	Unsupported value entered for the CLA byte.
6F00	JVM error that has no specified diagnosis.
9000	Command succeeded.
9481	Target has an invalid life cycle state.
9484	Unsupported algorithm ID in input data (PutKey).
9485	Invalid key check value in input data (PutKey).



3

3DES — triple data encryption standard. A symmetric key system that uses the DES algorithm to encrypt and decrypt data. The most common form of double-key 3DES processes data three times—forward (encrypted) with the first key, backward (decrypted) with the second key, and forward again (encrypted) with the first key. Decryption reverses those steps. *Also see: DES.*

A

AID — application identifier.

APDU — application protocol data unit. A sequence of hexadecimal values that conform to the low-level format for data exchanged between the host application and the card (through a card reader or terminal). Command and response APDU formats are defined by the ISO 7816-4 specification. *Also see: case, CLA, INS, Lc, Le, mode, P1/P2/P3.*

API — application programming interface. Software that defines the calling conventions an application uses to gain access to lower-level services performed by an operating system or another application.

application — Broadly applied, an application on an Open Platform Cyberflex Access card is the Card Manager, another security domain, or an applet instance. In some contexts, the term refers only to the Card Manager or another security domain. (This use is noted wherever it occurs in the Cyberflex Access Software Development Kit documentation.) *Also see: current application, default selected application, selectable application.*

atomic operation — An operation performed entirely or not at all. For example, if a failure prevents the operation from reaching completion, you must begin the operation again at the start.

ATR — answer to reset. The card-specific data string the card sends to the host (through the reader) when power is first applied to the card. An ATR signals the reader that the power-up was successful, and sends identification and (possibly) protocol data.

AUTH key — authentication key. On an Open Platform Cyberflex Access card, an AUTH key is the first key ($K_{enc, auth}$) in a keyset. The AUTH key is a static key used to generate a session key for encryption and authentication. *Also see: KEK key, MAC key.*

authentication — In cryptography, authentication is the ability of the sender and receiver to confirm each other's identity. *Also see: mutual authentication.*

B

blob — binary large object. A sequence of bytes of arbitrary length.

C

CA — *See certificate authority (CA).*

CAD — card acceptance device. Terminal or card reader used to communicate with a smart card. *Also see: IFD.*

CAP file — converted applet file. A file on an Open Platform Cyberflex Access card that contains an executable binary representation of the classes in a Java package, formatted to be compatible with the Java Card platform. Now replaced by the IJC file type. *Also see: IJC file.*

CAP File Generator — An application added by the toolkit to the host system, which converts the class files in a Java package into a CAP file and an export file for the converted package. Now replaced by the Program File Generator application. *Also see: Program File Generator.*

Card Manager — The application on a new Open Platform Cyberflex Access card that by default handles incoming commands. The Card Manager controls which applications can be loaded onto the card after it is issued. *Also see: current application, selectable application.*

case — APDU command type, as designated by the ISO 7816-4 specification. The type of input and output data included by each case of command is described below:

- Case 1 — No input data and no response data. Also referred to as a no-mode command (—).
- Case 2 — No input data, but elicits response data (which the card either returns automatically or in response to a follow-up GetResponse command). Also referred to as a Receive mode command (R).

- Case 3 — Input data, but no response data. Also referred to as a Send mode command (S).
- Case 2 — Send/Receive (S/R). Input data and potential response data (which the card either returns automatically or in response to a follow-up GetResponse command). Also referred to as a Send/Receive mode command (S/R).

Also see: APDU, Lc, Le.

CBC — cipher block chaining. A DES mode of encryption and decryption in which data blocks (plaintext or ciphertext) are bitwise exclusive-ORed with previous data blocks. The resulting data is encrypted or decrypted with a DES or 3DES key. In this way, each data block is affected by the previous blocks. CBC makes it easy to determine if a message has been altered. *Also see: EBC.*

certificate, digital certificate — A message that contains a user's public key, digitally signed by a certificate authority (CA) to assure the recipient that the key belongs to the user. The recipient can use the CA's signature to verify the authenticity of the certificate. Digital certificates may be kept in registries, which authenticated users search to find other users' public keys.

certificate authority (CA) — A trusted entity that maintains information about a client user's identities and issues a digital certificate to each user, which verifies to other parties the authenticity of the user's claimed identity. A CA may be an independent enterprise or an arm of the user's company. The usefulness of the endorsement depends on the certificate issuer's standing as a known and trusted authority. As a result, CAs may be endorsed by more widely known CAs, creating a chain of trust.

challenge — A random number. A challenge may be provided by one party, encrypted by a second party, and returned to prove the first party's identity.

checksum — A count of the number of bits in a transmission unit, typically included with the transmission so the receiver can make sure the correct number of bits arrived.

CLA — class. The first byte of an APDU, which identifies the command class. The class and instruction (INS) bytes uniquely identify the command type. *Also see: APDU, INS, P1/P2/P3.*

Codeshield — On-card bytecode verification feature available on the SchlumbergerSema Cyberflex Access e-gate 32K card. The Codeshield feature can be turned on or off when the card is manufactured.

confidentiality — In cryptography, confidentiality means restricting access to the meaning of information, so that the information can be understood only by the intended recipient. *Also see: authentication, cryptography, integrity, nonrepudiation.*

COVE — Cryptographic Object Viewer and Editor. A Cyberflex Access SDK utility for personalizing cards and setting up keys and key files on the card.

CRT — Chinese remainder theorem.

CryptoAPI — Cryptographic Application Programming Interface. A PC/SC-supported high-level programming environment that routes high-level function calls to a *cryptographic service provider* (CSP) as an interface to card-based services. CryptoAPI is included in all 32-bit Windows software, and provides the cardholder with an easy-to-use interface for identity checks and digital signature creation. *Also see: CSP.*

cryptogram, cryptograph — A block of encrypted data. Smart cards use cryptograms to demonstrate possession of a secret key without revealing the key itself.

cryptography — The science of information security, in which plaintext (ordinary text) is encrypted into ciphertext, then decrypted.

current application — On an Open Platform Cyberflex Access card, the current application is either the Card Manager, another security domain, or an applet instance on the card that is currently selected to receive commands. *Also see: default selected application, selectable application.*

D

DAP — data authentication pattern. Global Platform term for a means of authenticating data origin and/or integrity, such as a SHA-1 hash or DES signature used for transmitting a load file.

default selected application — On an Open Platform Cyberflex Access card, the default selected application is the application that receives commands unless another application is explicitly selected. On a new Open Platform Cyberflex

Access card, the default selected application is the Card Manager. You can specify a different default selected application with the `Install` or `SetStatus` command. *Also see: current application, selectable application.*

DES — Data Encryption Standard. A symmetric algorithm (specified in ANSI X3.92 and X3.106) that uses a single key for both encryption and decryption. DES is suitable for use when keys are distributed and stored dependably and securely, or when keys are exchanged between systems that have already authenticated each other (and the key life is restricted to the session or transaction). DES is commonly used to protect data from eavesdropping during transmission. *Also see: 3DES.*

digital certificate — *See certificate.*

digital signature — A digital string generated from a private key (such as an RSA private key). Since only the key's possessor has the key, the signature can come only from that person. A recipient of a message (encrypted or not encrypted) that has a digital signature attached can use the sender's public key to verify it. The receiver can verify the sender's identity and can determine whether the message was altered after it was signed. In addition to its other characteristics, a digital signature is difficult to repudiate.

diversified keys — To establish a secure channel on an Open Platform Cyberflex Access card, three new AUTH, MAC, and KEK keys are used after being generated with a Master Key set (AUTH, MAC, and KEK), a 3DES cryptographic algorithm, and the CPLC data from the card or other static data. This way, each card will have different keys. The only way to access the card is by knowing not only the Master Keys but also the cryptographic algorithm and the data used. Diversified keys give a card greater security.

E

EBC — Electronic Book Code. A mode of DES encryption and decryption, in which a plaintext or ciphertext block is encrypted independently with a DES or 3DES key (rather than bitwise exclusive-ORed with the previous block, as in CBC mode). EBC mode is as secure as the underlying block cipher, but unlike CBC, does not conceal plaintext patterns. EBC allows easy parallelization, so it takes less time than CBC. *Also see: CBC.*

EEPROM — Electrically erasable programmable read-only memory. Microprocessor memory that does not lose its data when power is removed from the microprocessor, and that can be erased and reprogrammed repeatedly with applied electrical voltage.

e-gate — commercial name of the FMSC smart card, capable of interfacing with a standard ISO 7816 reader or with the USB port of a PC.

EMV — Europay / MasterCard / Visa. Standards designed to provide interoperability between a range of smart card hardware and software programs in the payment industry. EMV standards were developed by an alliance of bankcard associations (sometimes called *EMV96*).

external authentication — The process a host-side application uses to establish its credentials for gaining access to a smart card. For example, the host asks the card for a challenge (a random number), which the host application encrypts. The card performs a parallel operation on the challenge string and compares the resulting cryptograms. If the cryptograms match, the host has proven that it possesses a key stored on the card. *Also see: challenge, internal authentication.*

F

FCI — file control information. Data the card returns in response to selection of the Card Manager or another security domain.

file-based smart card — A smart card whose operating system is based on a file system. *Also see: Global Platform specifications, Open Platform smart card, SCOS.*

G

global PIN — global personal identification number. An alphanumeric string used as a password to establish person-to-card authentication. *Also see: PIN.*

Global Platform specifications — Open, cross-industry standards for card program management and terminal communications, which support a security-oriented approach to card application development and operation. *Also see: file-based smart card, Open Platform smart card.*

H

hash — A digest (typically of a fixed length) of a longer string of characters or numbers. SHA-1 is a widely used one-way hash code. *Also see: MAC, SHA.*

I

ICC — integrated circuit card. ISO term for a smart card.

IDE — integrated development environment.

IFD — interface device. ISO term for a smart card reader. *Also see: CAD.*

IJC— interoperable javacard CAP file. A file on an Open Platform Cyberflex Access card that contains an executable binary representation of the classes in a Java package, formatted to be compatible with the Java Card platform. As compared to the CAP file type, contains extra information needed for on-card bytecode verification (Codeshield feature) on the Cyberflex Access e-gate 32K card. Replaces the CAP file type. *Also see: Codeshield, CAP file.*

INS — instruction. The second byte of an APDU, which identifies the command instruction. The class (CLA) and instruction bytes uniquely identify the command type. *Also see: APDU, CLA, P1/P2/P3.*

integrity — In cryptography, integrity is the ability to detect whether transmitted or stored information has been altered. *Also see: authentication, confidentiality, cryptography, nonrepudiation.*

internal authentication — The process for establishing the card's credentials with the host system. For example, the host sends the card a challenge (a random number), which the card encrypts. The host application performs a parallel operation on the challenge string and compares the resulting cryptograms. If the cryptograms match, the card is proven to be trustworthy.

IOP — interoperability layer. The SchlumbergerSema foundation software that acts as a translator for calls from the mid-level card implementations to the Microsoft Windows Resource Manager.

ISO 7816 — International Standardization Organization specification #7816. A set of standards for smart card physical attributes, data elements, basic commands, and security architecture.

IV — initialization vector. A value used in DES block encryption and decryption.

J

Java card applet — Smart card applet written in the Java language. You can download an applet to a Java-enabled smart card as a load file, instantiate it, and execute it in the JCRE.

JCRE — Java card runtime environment. Consists of the Java virtual machine, Java smart card libraries, and supporting files, which together comprise an environment for executing smart card applet instances.

JNI — Java native interface. *Also see: IOP JNI.*

JVM — Java card virtual machine. Consists of a converter that runs on the host system and an interpreter that runs on the card. These elements together act as a translator that converts an application's instructions into a format the smart card can understand. The JVM loads and executes Java class files and makes applications portable.

K

KEK key — key encryption key. On an Open Platform Cyberflex Access card, a KEK key is the third key (K_{KEK}) in a keyset. The KEK key is a static key used to generate a session key for key encryption. *Also see: AUTH key, MAC key.*

key — A number or character string used for security purposes:

- A *cryptographic key* is a number chosen for mathematical properties that are useful in encryption and decryption. The key length typically determines how strong the key is—how difficult it is to decrypt the ciphertext without having the key.
- An *identification key* is used to prove identification, such as a PIN the cardholder or card administrator presents that must match a stored value.

key pair — A private and public key set: complementary components of a key (such as an RSA key) used for asymmetric encryption and decryption. RSA keys are used for such purposes as secure transmission of data and for digital signatures. You use the private key to decrypt text that has been encrypted with your public key by someone else (who can find out what your public key is from you or from a certificate authority's public directory). In addition to the role in encrypting messages, RSA key pairs enable you to authenticate yourself to others by using your private key to encrypt a digital certificate. When the message arrives, the recipient uses your public key to decrypt it. *Also see: digital signature, private key, public key, RSA.*

keyset — On an Open Platform Cyberflex Access card, a keyset is a group of three keys, each of which has a specific use. A keyset can be associated with the Card Manager (to protect card data) or another security domain (to protect data loaded in the security domain environment). *Also see: AUTH key, KEK key, MAC key, secure channel, security domain.*

L

Lc — length of command data. The fifth byte of an APDU command in Send mode (case 3) or Send/Receive mode (case 4). The Lc specifies the length of the input data included immediately after the Lc byte. In a Receive mode command, the Lc may also be referred to as parameter 3 (P3). *Also see: Le.*

Le — length of response data. The APDU component byte included at the end of a Receive mode command (case 2) or Send/Receive mode command (case 4) command, which specifies the length of the data you expect the card to return. In a Receive mode command, the Le may also be referred to as parameter 3 (P3). *Also see: Lc.*

load file — A file downloaded to an Open Platform Cyberflex Access card from a program file, which you then instantiate to create a Java applet instance that runs on the card. *Also see Program File Generator.*

LSB — least significant byte.

LSN — least significant nibble.

LV — Length/Value. A format for command input data that is used on an Open Platform Cyberflex Access card. The format consists of length and value elements, without the tag element found in TLV-formatted data.

M

MAC — Message Authentication Code. A variation on a one-way hash function, which uses a DES or 3DES key to compute the hash digest. This feature makes a MAC useful for confirming data integrity, as well as authenticating the identities of the message originator and recipient. RSA key pairs can also be used with one-way hash functions to produce a digital signature.

MAC key — Message Authentication Code key. On an Open Platform Cyberflex Access card, a MAC key is the second key (K_{MAC}) in a keyset. The MAC key is a static key used to compute a MAC hash. *Also see: AUTH key, KEK key.*

mode — Shorthand designator for the type of APDU command, which corresponds to one of the cases defined by the 7816-4 specification, as described below:

- — (no mode) — Case 1 command: Includes no input data and does not create or locate response data from the card.
- R — (Receive mode) Case 2 command: Includes no input data, but prompts the card to return data.

- **S** — (Send mode) Case 3 command: Includes input data, but does not create or locate response data from the card.
- **S/R** — (Send/Receive mode) Case 4 command: Includes input data and prompts the card to return data.

modulus — (*abbreviated as mod*) An RSA key element. Mathematically, the modulus is the number by which a logarithm in one system must be multiplied to obtain the corresponding logarithm in another system.

MSB — most significant bit.

MSN — most significant nibble.

mutual authentication — The process of confirming the identity of two communicating parties, such as the host system and a smart card application (the on-card agent that is currently processing incoming commands). On an Open Platform Cyberflex Access card, mutual authentication is essential for establishing a secure channel. *Also see: keyset, security domain.*

N

nonrepudiation — In cryptography, nonrepudiation is assurance that the originator of the information cannot later deny that he or she was the information source. *Also see: authentication, confidentiality, cryptography, integrity.*

O

octet — *As applied to data:* evenly divisible into 8-byte blocks.

Open Platform smart card — A smart card whose operating system is based on the Global Platform specification, rather than on a file system. *Also see: file-based smart card, Global Platform specifications.*

operating system — On a smart card, the operating system is an application that can execute a set of instructions which form the basic operations that enable a card program to run. *Also see: Card Manager, file-based smart card, Global Platform specifications, Open Platform smart card.*

P

P1, P2, P3 — Parameters 1, 2, 3. The third, fourth, and fifth bytes of an APDU, which supply extra information about the command. The type of data P1 and P2 contain is command-specific. P3 is typically the length of input data (Lc) or length of expected response data (Le). *Also see: APDU, CLA, INS.*

padding — Extra characters or bytes inserted into data to standardize the size of the data block. For example, you might apply padding to data to perform an operation that requires octet data (data that is evenly divisible by 8 bytes).

PC/SC — Personal Computer/Smart Card. A PC-based, open architecture for interoperation between hardware and software components from different vendors. The PC/SC architecture was developed by a group of smart card and PC operating system vendors, including SchlumbergerSema, Microsoft, Siemens Nixdorf, HP, and CP8 Transac. For more information about the PC/SC workgroup, see <http://www.pcscworkgroup.com>.

PIN — personal identification number. An alphanumeric string which can be used as a password to establish person-to-card authentication. *Also see: global PIN.*

PIN policy — During card personalization, specifies the rules that apply to PIN management.

PKCS #11 — Public Key Cryptography Standard #11. An RSA Laboratory-sponsored set of intervender standard protocols developed to facilitate secure information exchange. Cyberflex Access series cards support PKCS #11-compliant card programs by supplying a library of functions that provide cryptographic and security services to the PKCS #11 interface, Cryptoki. These functions are part of the SchlumbergerSema Smart Card middleware.

PKI — Public Key Infrastructure. An infrastructure model for exchanging data and money through the use of a public and private key pair, which is obtained and shared through a trusted authority. PKIs provide for digital certificates that can identify individuals and organizations, and for directory services that can store and revoke the digital certificates. An Internet standard for PKI is currently underway.

private key — The secret key component of an asymmetric key pair, such as an RSA key pair. The private key value is known only by its owner—it is never shared with anyone. You can use the key pair for such purposes as email encryption and decryption, and for digital signatures. *Also see: key pair, public key, RSA.*

program — Software on the host system or smart card designed to execute special operations. Sometimes referred to as an application, although the term application can also refer to any other type of command processor on an Open Platform Cyberflex Access card.

Program File Generator — An application added by the toolkit to the host system, which converts the class files in a Java package into an IJC file and an export file for the converted package. Replaces the CAP File Generator application. *Also see: CAP File Generator.*

public key — The publicly available key component of an asymmetric key pair, such as an RSA key pair. The public key is published and available to anyone who wants to send an encrypted communication to the owner of the private key. *Also see: key pair, private key, RSA.*

R

R mode — receive mode. *See: Le, mode.*

RA — *See: registration authority (RA).*

registration authority (RA) — registration authority. An entity that acts as the verifier for a certificate authority before a digital certificate is issued to a requestor. *Also see: certificate authority.*

RFU — reserved for future use.

RNG — random number generator. An agent that generates 0s and 1s in a sequence designed so that, at any point, the next bit cannot be predicted from analyzing the previous bits. In some cases, a pseudo random number generator may be used. In this case, the key may be an apparently random string generated from a relatively small random seed.

RSA — A widely used *asymmetric key system* known by the initials of its originators: Rivest, Shamir, and Adleman. In this guide, RSA stands for the 512-bit, 768-bit, or 1024-bit encryption algorithm used by Cyberflex Access series smart cards. RSA uses a public and private *key pair*, with the *public key* published openly, while the *private key* remains secret. *Also see: key pair, private key, public key.*

S

S mode, S/R mode — send mode, send/receive mode. *See: Lc, Le, mode.*

secure channel — A gateway for transferring commands and data between the host system and an Open Platform Cyberflex Access card. You establish a secure channel by performing card and host mutual authentication, using keys from a keyset in the target security domain. *Also see: keyset, mutual authentication, security domain.*

security domain — An environment that is established on an Open Platform Cyberflex Access card to protect data, such as an applet and its collateral data. Security domains enable applications from multiple providers to reside together on a smart card without compromising each other's security. A security domain typically has a domain-specific keyset for security use. *Also see: keyset, mutual authentication, secure channel.*

selectable application — An application (Card Manager, other security domain, or applet instance on an Open Platform Cyberflex Access card) whose life cycle state is selectable. *Also see: current application, default selected application.*

session key — A key that is generated during a card session, which expires with the card session.

SHA, SHA-1 — Secure Hash Algorithm. An algorithm similar to the MD4 family of hash functions, which is specified in ANSI X9.30. SHA-1 is a technical revision of SHA (FIPS 180).

signing key — *See key pair.*

T

TLV — tag/length/value. A format for command input data. Like command headers, TLV-encoded data fields comply with the ISO 7816-4 communication protocol. The format consists of these three elements:

- Tag — A prefix of 1 to 2 bytes that indicates the type of data in the following byte stream
- Length — A hexadecimal specifier of 1 to 4 bytes that indicates how many data bytes are in the byte stream
- Value — The byte stream

Also see: LV.

TPDU — transmission protocol data unit. Basic element of the lower-level protocol for exchanging APDU data between the host application and the card. TPDU protocols are defined by the ISO 7816-3 specification. The T=0 protocol is an asynchronous, byte-oriented, half-duplex transmission protocol in which a byte is the smallest transmissible data unit. The T=1 protocol is an asynchronous, block-oriented, half-duplex transmission protocol.

triple-DES — *See 3DES.*

W

weak key — A key with regularities that result in a poor level of encryption. For cards that generate DES keys check for four weak and twelve semi-weak DES keys, which the card always discards.

Windows for Smart Cards — A smart card runtime environment developed by Microsoft, based on Windows and Visual Basic tools and principles.



3

- 3DES key
 - used to encrypt keys 125

A

adding

- applet instance (Install) 93
- applet instance (Toolkit) 168
- global PIN (PinChange) 113
- key set (overview) 33
- key set (PutKey) 120
- load file (Load) 107
- load file (Toolkit) 166
- program file to host system 160
- tagged data object (PutData) 116

AID

- changing (PutData) 116
- effect on register method 100
- retrieving (GetData) 75
- retrieving (GetStatus) 80
- specifying 39
- using partial AID for selection 127

APDU commands

- general description of components 180
- input/output formats 182
- S/R modes illustrated 182–184
- sending to the card (tutorial) 171–176

APDU Manager utility (Toolkit)

- saving a function list 176
- saving a function to a function list 173
- sending a function to the card 173
- setting up a session 172
- used in development 171

Applet class

- using for card program 138

applet instances

- blocking/unblocking 46
- creating a program file 160
- creating from load files 168
- default AID value for 39
- deleting (Delete) 66
- input data for making selectable 102
- instantiating (Install) 93
- limit on size 137
- mandating privileges on installation 101
- response data when selected 131
- retrieving status (GetStatus) 80
- roadmap for developing 143
- selecting (SelectApplication) 127
- setting life cycle state (SetStatus) 133
- setting size (Toolkit) 170
- steps for creating 36

applets

- data types/values supported 138, 141
- information for creating 137
- instantiating with Toolkit 166
- methods required 139
- writing (tutorial) 146–176

application privileges data

- also see: default selected application*
- examples of 12
- format of 11
- mandating for installed application 101
- retrieving (GetStatus) 80
- setting (Install) 93
- ways of enabling default selection 43

applications: *see applet instances*

arguments, limiting 140

ATR

- value for Cyberflex Access 32K v4 card 2
- value for Cyberflex Access 64K v1 card 2
- value for Cyberflex Access Developer 32K card 2
- value for Cyberflex Access e-gate 32K card 2

authenticating

- host/card 70

B

blocking

- applet instance (overview) 46
- security domain (overview) 45
- security domain/applet (SetStatus) 133
- selectability maintained 127
- suspension of default selection 103

buffers

- defining in example 149

C

calling methods

- using shallow calling tree 140

card

- also see: Card Manager*
- authentication to host 70
- maximum EEPROM available on 137

Card Applet Manager (Toolkit)

- adding applets 166

card applets

- see: applet instances*

card commands 61–136

- also see: APDU commands*

- Delete 66
- ExternalAuthenticate 70
- GetData 75
- GetStatus 80
- guide to using 27–60
- InitializeUpdate 85
- Install 93
- Load 107
- PinChange 113
- PutData 116
- PutKey 120
- SelectApplication 127
- SetStatus 133
- summary of commands 62

card cryptogram

- calculation method 91

- card issuer BIN
 - adding/changing (PutData) 116
 - format of retrieved data 78
 - retrieving (GetData) 75
- card issuer data
 - adding/changing (PutData) 116
 - format of retrieved data 78
 - retrieving (GetData) 75
- Card Manager
 - application privilege for locking 11
 - automatic default selection of 103
 - changing AID (PutData) 116
 - locking/unlocking/terminating 44
 - response data when selected 130
 - retrieving AID (GetData) 75
 - retrieving status (GetStatus) 80
 - selecting (SelectApplication) 127
 - setting life cycle state (SetStatus) 133
 - when necessary to select 127
- Card Manager pane
 - changing key values 35
 - creating an applet instance 43
 - deleting instance/load file 40
- card production life cycle
 - adding/changing data (PutData) 116
 - changing (SetStatus) 133
 - data returned by InitializeUpdate 90
 - format of retrieved data (GetData) 77
 - retrieving (GetData) 75
 - retrieving (GetStatus) 80
 - retrieving (overview) 50
- changing
 - card status/life cycle state (SetStatus) 133
 - data object values (PutData) 116
 - global PIN (PinChange) 113
 - key set values (overview) 35
 - key/key set values (PutKey) 120
 - privilege for changing global PIN 11
 - selectability/default selection (Install) 93
- class (CLA) byte
 - description of 180
- class files
 - converting to .ijc files in Toolkit 162
 - location of 160
- class hierarchy, limiting 140
- class library path
 - importance of setting 160
 - setting 139
- classes, importing (example) 147
- compiling program code
 - example (tutorial) 160
- constants
 - defining in example 148
 - guidelines for use of 140
- constructors
 - defining in example 150
- Create Instance dialog box 43, 168
- Create Program dialog box 167
- cryptogram
 - calculating MAC/host cryptogram 72
 - encryption calculation illustrated 60
 - MAC calculation illustrated 56
- Cyberflex Access Library reference 177–178

D

- DAP verification
 - application privileges bit for 11
 - implemented for input commands/data 6
 - lack of mandatory verification 100
- data authentication patterns (DAPs)
 - DAPs supported on card 32
- data objects
 - change values of (PutData) 116
 - retrieving (GetData) 75
- data types
 - range of values supported 138
 - types supported 141
 - types/values supported 138
- default selected application
 - application privilege bit 11
 - conditions for use 103
 - previous state required 93
 - setting/clearing privilege (Install) 93
- delegated management
 - current support status 4
- Delete command 66
- deleting
 - applet instance (example) 68
 - load file/applet (Delete) 66
 - load file/applet (Toolkit) 40
 - security domain (InitializeUpdate) 85
 - unavailability of logical deletion 81
- diversification, key 87

E

- EBC DES operation
 - EBC mode briefly described 191
- EEPROM
 - data returned by InitializeUpdate 90
 - defined 191
 - determining amount for applet 170
 - memory limit 137
 - recovery when object deleted 66
 - retrieving identification data 77
- electronic book code (ECB)
 - used in PutKey 125
- environment variables
 - setting class library path 139, 160
- errors
 - also see: status words (SW1/SW2)*
 - GPOS error codes 185
 - program file generation failure 164
 - status words (overall list) 185
- exceptions
 - throwing in example 153
- ExternalAuthenticate command 70
 - example 73
 - preceding command for 85

F

- files
 - selecting (SelectApplication) 127

G

- GetData command 75
 - CPLC response data 77
 - example 76
 - issuer BIN returned 78
 - issuer data returned 78
- GetStatus command 80
 - example 82
 - response data 83
- global PIN
 - adding/changing (PinChange) 113
 - also see: PIN*
 - application privilege for changing 11
 - format of data for changing 115
- GlobalPlatform specification
 - class byte value 180

H

- host cryptogram
 - calculation method 72
 - returned by InitializeUpdate 90
- host, authentication to the card 70

I

- importing classes, example of 147
- initialization chaining vector
 - description of 55
- InitializeUpdate command 85
 - example 89
 - sending before ExternalAuthenticate 70

- input/output APDU command formats 182
- Install command 93
 - application-specific parameters 101
 - example 104
 - input: installing security domain 101
 - input: making an application selectable 102
- install method
 - required in applet 139
- installing
 - format of load file install buffer 97
 - load file install buffer size limit 97
 - load file parameters data block 96
- instruction (INS) byte
 - defining INS values in example 148
 - description of 180
- ISO protocol
 - input/output APDU formats described 182
 - ISO 7816 standard briefly defined 193

J

- jar files
 - location of class library files 139
- Java heap size 137
- Java Virtual Machine (JVM)
 - OutOfMemoryException 138

K

- key diversification 87
- key encryption key (KEK)
 - used in PutKey 125
 - used to change PIN value 113

key sets

- adding (overview) 33
- adding/changing (PutKey) 120
- changing key values (Toolkit) 35
- default key set (described) 9
- default key set values 120
- limitations on retrievable data 34
- modifying (overview) 35
- reasons for adding 34
- version number returned (PutKey) 124

keys

- changing values (PutKey) 120
- check values (PutKey) 124, 125
- encryption (PutKey) 125
- identification of 120
- length of 122

L

Lc byte

- general description of 180

Le byte

- general description of 180

length/value (LV) format

- example of 95

life cycle state

- changing (SetStatus) 133
- effect on selectability 127
- retrieving (GetStatus) 80
- setting (SetStatus) 133

life cycle states

- making an application selectable 42

Load command 107

- input: final block verified 111
- input: first block verified 110

input: no verification 109

- preceding command (Install) 93

load files

- adding (Load) 107
- AID values for 39
- creating 166
- creating (Toolkit) 166
- creating instances of 168
- deleting (Delete) 66
- installation buffer format 97
- installation buffer size limit 97
- load parameters data block 96
- retrieving status of (GetStatus) 80
- SHA-1 hash used for 32

local scope, results of using 140

locking

- applet instance (overview) 46
- application privilege for locking card 11
- card/elements (overview) 44
- card/elements (SetStatus) 133
- effect on selectability 127
- restrictions on selecting applications 127
- suspension of default selection 103

logical deletion

- unavailability of 81

M

MAC

- calculation illustrated 56
- calculation illustration 72
- card cryptogram calculation 91
- command format illustrated 54
- commands that require security 33
- overview of operations 53

- privilege for DAP verification 11
- sending with ExternalAuthenticate 70
- setting level for security channel 71
- source of ICV 71
- uses of 32
- verification not mandatory 100

MAC+Enc

- command checking 59
- command format illustrated 59
- commands that require security 33
- encryption calculation illustrated 60
- overview of operations 57
- privilege for DAP verification 11
- setting level for security channel 71
- uses of 32
- verification not mandatory 100

Make Card Applet button 162

making selectable

- applet/security domain (Install) 93
- applet/security domain (SetStatus) 133
- input data (install) 102
- previous state required 93

microprocessor

- see: EEPROM*

Microsoft Developer Studio 5.0

- setting class path for 139

mode

- description of command modes 181

mutual authentication

- beginning process of 85
- ExternalAuthenticate process 70

O

object reference space

- recovery when instance deleted 66

OutOfMemoryException

- result of oversize Java heap 138

P

P1/P2 bytes

- general description of 180

padding

- input data for card cryptogram 91
- input data for encryption 59
- input data for host cryptogram 72
- input data for MAC generation 55

personalizing

- card/elements (SetStatus) 133
- roadmap for personalizing card 28

PIN

- added in COVE application 25
- also see: global PIN*
- setting object parameters in example 148

PinChange command 113

primitive data types

- Deposit variable (example) 141
- using 140

privileges

- see: application privileges data*

process method

- including in applet 138
- required in applet 139

Program File Generator

- behavior if class file error detected 164
- output displayed 164
- running from Toolkit 160
- using 160
- utilities used by 37

program files

- converting from .class files in Toolkit 162
- converting from class files 160
- creating 37
- creating a load file from 166
- creating load file from (Load) 107
- see: load files*

programs

- also see applet instances*
- programming guidelines 140

protocol (T=0)

- briefly described 182
- input/output APDU formats 182

PutData command 116

- retrieving data added by 75

PutKey command 120

R

Receive mode

- APDU format illustrated 183

register method

- forms included in applets 100

retrieving

- applet data (SelectApplication) 127
- card data (SelectApplication) 127
- card identification objects (GetData) 75
- key set data (limitations of) 34

- load block limit (SelectApplication) 127
- status data (GetStatus) 80

ROM identification data

- retrieving 77

RSA keys

- RSA term briefly defined 198

S

Schlumberger Smart Card Toolkit window

- displaying 161
- displaying Create Program dialog 167

secure channel

- class byte value indicating 180
- determining security level of 30
- establishing (ExternalAuthenticate) 70
- establishing (InitializeUpdate) 85
- optional for GetData 75
- security levels described 7
- setting security level 31
- steps for establishing 29
- terminating 6

securing card

- SetStatus command 133

security domains

- application privileges bit for 11
- blocking/unblocking 45
- changing AID (PutData) 116
- input data for installing 101
- input data for making selectable 102
- installing (Install) 93
- mandating privileges on installation 101
- response data when selected 130
- retrieving AID (GetData) 75
- retrieving status (GetStatus) 80

- selecting (SelectApplication) 127
 - setting life cycle state (SetStatus) 133
 - unavailable commands 61
 - security level
 - class bytes for 7
 - security levels described 7
 - setting 31
 - setting for secure channel 70
 - select method
 - required in applet 139
 - selectable life cycle state
 - ways of setting 42
 - SelectApplication command 127
 - example 129
 - response data for applet instance 131
 - response data for card/security domain 130
 - running from Toolkit 41
 - selected by default privilege
 - ways of enabling 43
 - selecting an application
 - overview 41
 - Send mode
 - APDU format illustrated 183
 - Send/Receive mode
 - APDU format illustrated 184
 - SetStatus command 133
 - example 135
 - SHA-1 hashes, use of 32
 - SHA-1 Last command
 - SHA-1 operations briefly described 199
 - shorts, using short data types 141
 - signatures
 - briefly described 191
 - SimplePurse sample program (tutorial)
 - defining INS values 148
 - starting
 - Program File Generator 160
 - Smart Card Toolkit 161
 - status of card/elements
 - return data format (GetData) 83
 - status words (SW1/SW2)
 - general description 181
 - listed 185
 - not returned by terminated card 132
 - specifying for applet (example) 149
 - table data described 185
 - TE9 SW availability 65
- T

T=0 ISO protocol (used on card)

 - input/output formats 182

tag/length/value (TLV)

 - example of TLV format 67

TE9 status words, availability of 65

terminating card

 - overview 44

terminal

 - see: host*

terminating card

 - card (SetStatus) 133
 - effect on selecting applications 127
 - muting of status words 132
 - privilege for 11

terminating secure channel

 - ways to end secure channel 6

TPDU protocol briefly described 182
tutorial for writing an applet 146–176

U

unblocking

- applet instance (overview) 46
- global PIN (PinChange) 113
- security domain (overview) 45
- security domain/applet (SetStatus) 133

unlocking

- applet instance (overview) 46
- card/applet instance (SetStatus) 133
- card/elements (overview) 44

V

variables, guidelines for using 140

verifying DAP

- application privilege for 11

Visual J++

- setting class library path 139
- setting the class path for 160

W

Wallet sample program (tutorial) 146–176

- calling debit method 155
- calling getBalance method 156
- calling install method 151
- calling process method 152
- calling select method 151

- calling validate method 158
- defining buffers 149
- defining constants 148
- defining constructors 150
- general description 146
- importing classes 147
- sending APDUs to the card 171–176
- setting PIN object parameters 148
- specifying status words 149
- steps for creating 147–159
- throwing exceptions 153

websites

- smart cards resource list xvii

windows

- Schlumberger Smart Card Toolkit 161