SPECIFICATION

Calypso Terminal API

Calypso Card

Version 3.0.0-SNAPSHOT, 2026-07-20

⚠ ***Warning. The present document cancels and replaces the previous release.***

Link and Contact

Website
https://calypsonet.org

Support
support@calypsonet.org

Table of Contents

1. Abstract

This document is the official technical specification of the **Terminal Calypso Card API** standardised by the **Calypso Networks Association** (CNA). It defines, in a language-agnostic way, the interfaces, classes, enumerations, exceptions, structural relationships and behavioural requirements that any conforming implementation of the Terminal Calypso Card API MUST satisfy.

The Terminal Calypso Card API is the **application-facing** surface used by terminal applications to operate Calypso transit and ticketing cards. It builds on top of the **Terminal Reader API** (CNA-TR-API) and delegates every cryptographic operation to a symmetric (CNA-TCCS-API) or asymmetric (CNA-TCCA-API) crypto module.

Document Status

Reference	YYMMDD-SP-CNATerminalAPI-CalypsoCard
Short name	CNA-TCC-API
Version	3.0.0-SNAPSHOT
Revision date	2026-07-20
Editor	Calypso Networks Association
Source repository	https://github.com/calypsonet/calypsonet-terminal-calypso-card-uml-api
Reference license	Creative Commons Attribution-NoDerivatives 4.0 International (CC BY-ND 4.0)

The present document specifies the 3.0.0-SNAPSHOT of the Terminal Calypso Card API. It defines a single, coherent baseline against which conforming implementations are evaluated; the **Since** column indicates the version in which each member was introduced.

Revision List

This section lists the high-level changes per version. The complete, fine-grained changelog is maintained in the `CHANGELOG.md` file at the root of the source repository.

Version / Date	Modifications
v3.0.0-SNAPSHOT 2026-07-20	Baseline.

2. Introduction

2.1. Purpose

The Terminal Calypso Card API defines the public surface used by terminal applications to:

- select a Calypso card through a CNA-TR-API selection scenario enriched with Calypso-specific commands;
- operate read-only transactions without involving a crypto module (**free** transaction);
- operate fully authenticated transactions using a **symmetric** crypto module (CNA-TCCS-API)—either in the **regular** mode (compatible with every Calypso product) or in the **extended** mode (additional features only available on Calypso Prime Extended);
- operate fully authenticated transactions using an **asymmetric** crypto module (CNA-TCCA-API) in **PKI** mode;
- expose a dynamic view of the card (`CalypsoCard`) updated throughout the transaction.

2.2. Scope

This specification covers:

- the factory used to instantiate the public types of the API (`CalypsoCardApiFactory`);
- the data model of a Calypso card (`CalypsoCard`, `ElementaryFile`, `FileData`, `FileHeader`, `DirectoryHeader`, Stored Value log records);
- the selection extension (`CalypsoCardSelectionExtension`);
- the family of transaction managers (Free, Secure Regular, Secure Extended, Secure PKI);
- the **symmetric** and **asymmetric** security setting carriers;
- the **Stored Value** operations and **Search Record Multiple** support;
- the SPIs implemented by **crypto modules** to plug into the secure transaction managers;
- the family of exceptions raised during a Calypso transaction.

The following topics are **out of scope**:

- the on-the-wire APDU dialog with the card (handled internally by the implementation);
- the cryptographic algorithms themselves—only the SPI contract is normative;
- the on-the-wire transport between the terminal and any remote ticketing system.

2.3. Conformance

A software product conforms to this specification if and only if:

1. it implements every interface and class defined in Chapter 5 with the operations and semantics prescribed in this document,
2. it raises exceptions exclusively of the types defined in Section 5.5 for the situations described,
3. it preserves the structural relationships described in Chapter 4.

Throughout this specification, the key words **MUST**, **MUST NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **MAY** and **OPTIONAL** are to be interpreted as described in RFC 2119 and RFC 8174 when, and only when, they appear in all capitals.

In conformance with RFC 2119, **SHALL** is equivalent to **MUST**; this specification uses **MUST** exclusively in order to avoid ambiguity.

2.4. Document Conventions

2.4.1. Naming

Type names follow `UpperCamelCase`; operation, parameter and enumeration-member names follow `lowerCamelCase` except for enumeration values which use `UPPER_SNAKE_CASE`. Names defined by this specification are reproduced as-is in the UML class diagram (Section 4.3).

2.4.2. Language-agnostic types

Symbolic type	Description	Typical bindings
`string`	Sequence of characters, UTF-8 encoded by default.	`String`, `string`, `str`.
`boolean`	Two-state truth value.	`boolean`, `bool`.
`byte`	Signed 8-bit value.	`byte`, `i8`.
`short`	Signed 16-bit integer.	`short`, `Int16`.
`integer`	Signed 32-bit integer.	`int`, `Int32`.
`byte array`	Ordered sequence of octets.	`byte[]`, `[u8]`, `bytes`.
`list<T>`	Ordered collection of `T` values, may contain duplicates.	`List<T>`, `Vec<T>`.
`set<T>`	Unordered collection of unique `T` values.	`Set<T>`, `HashSet<T>`.
`map<K,V>`	Associative array binding keys of type `K` to values of type `V`.	`Map<K,V>`, `Dictionary<K,V>`.
`sortedMap<K,V>`	Associative array sorted by key.	`SortedMap<K,V>`, `BTreeMap<K,V>`.
`T?`	Nullable value: either a `T` value or the absence of value (`null`).	`Integer`/`Short`/`Byte` (boxed), `Optional<T>`, `T?`.
`void`	Indicates that an operation returns no value.	`void`, `Unit`, `None`.

2.4.3. Type tables

Each interface, class, enumeration and exception defined in this document is described by a table of the form:

Name	The type's normalized name.
Kind	The category of the type (interface, class, enumeration, exception, etc.).
Namespace	The namespace in which the type is defined.
Generics	The generic type parameter of this type, when applicable.
Extends	The type extended by this type, when applicable.
Implements	The interface implemented by this type, when applicable.
Since	The version of the API in which the type was introduced.
Purpose	A normative description of the type's responsibility.
See also	Cross-references to related operations or types, each a hyperlink to the corresponding table. Present only when applicable.

The **Purpose** row states, in one or two sentences, the responsibility of the type and the way an instance is

obtained. It is meant to be scanned, not read: any content that requires structure or depth — rules, lists, notes, value tables or examples — is placed as prose below the table.

2.4.4. Operation tables

Each operation defined in this document is described by a table of the form:

Signature	The operation's language-agnostic signature.
Since	The version of the API in which the operation was introduced.
Description	A normative description of the operation's behaviour.
Parameters	A description of each input parameter.
Returns	A description of the returned value, if any.
Throws	A list of exceptional conditions, each mapped to a specific exception type.
See also	Cross-references to related operations or types, each a hyperlink to the corresponding table. Present only when applicable.

2.4.5. Operation contracts

To avoid repetition, the following rules apply to every operation table in Chapter 5 and are therefore not restated for each operation:

- **Non-`null` arguments.** Unless explicitly stated otherwise, every input parameter MUST be non-`null`. An implementation MUST raise `IllegalArgumentException` if a `null` value is supplied. This implicit `null` check is not repeated in the **Throws** row, which states `None.` when no other exception can occur.
- **Non-`null` results.** Unless the return type is marked nullable (`T?`) or the **Returns** row states otherwise, an operation returns a non-`null` value.
- **Collections.** Unless the return type is marked nullable (`T?`), an operation whose return type is a collection (e.g. `list`, `set`, `map`) returns an empty collection rather than `null`.
- **Fluent composition.** Builder-style operations return the current instance so that calls can be chained.

2.4.6. Concurrency

Unless explicitly stated otherwise, the stateful types defined by this specification are **not** thread-safe. A given instance MUST be accessed from a single thread at a time; concurrent use of the same instance without external synchronisation results in undefined behaviour. Distinct instances MAY be used concurrently.

2.5. References and Resources

2.5.1. Calypso References

References to CNA Terminal APIs designate the indicated **major** version; any later release within the same major is backward-compatible per that API's evolution policy.

Reference	Document
[CNA-TR-API]	*CNA Terminal API — Reader* (SP-CNATerminalAPI-Reader), version 3.0, Calypso Networks Association.
[CNA-TCCS-API]	*CNA Terminal API — Calypso Crypto Symmetric* (SP-CNATerminalAPI-CalypsoCryptoSymmetric), version 0.1, Calypso Networks Association.

Reference	Document
[CNA-TCCA-API]	*CNA Terminal API — Calypso Crypto Asymmetric* (SP-CNATerminalAPI-CalypsoCryptoAsymmetric), version 0.2, Calypso Networks Association.

2.5.2. Normative References

Reference	Document
[RFC 2119]	*Key words for use in RFCs to Indicate Requirement Levels*, S. Bradner, March 1997.
[RFC 8174]	*Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words*, B. Leiba, May 2017.
[ISO/IEC 7816]	*Identification cards — Integrated circuit cards*.

2.5.3. External Resources

Resource	Link
Calypso Networks Association	https://calypsonet.org
Terminal APIs website	https://terminal-api.calypsonet.org
Terminal APIs documentation	https://docs.terminal-api.calypsonet.org

3. Glossary and Acronyms

3.1. Glossary

Term	Definition
Calypso Card	Smart card compliant with the Calypso specifications.
Secure Session	Authenticated and integrity-protected exchange between the terminal and the card.
Regular Mode	Secure session mode compatible with every Calypso product.
Extended Mode	Secure session mode supporting additional features (early mutual authentication, encryption of session APDUs, larger sessions). Available on Calypso Prime Extended.
PKI Mode	Secure session mode using asymmetric cryptography (Calypso 3.3+).
Stored Value (SV)	Calypso feature exposing a card-resident monetary counter.
Crypto Module	Component implementing the SPI exposed by CNA-TCCS-API or CNA-TCCA-API.
KIF / KVC	Calypso Key Identifier / Key Version Code.
HCE	Host Card Emulation.

3.2. Acronyms

Abbreviation	Expansion
AID	Application IDentifier
APDU	Application Protocol Data Unit
CNA	Calypso Networks Association
DF	Dedicated File
EF	Elementary File
FCI	File Control Information
FCP	File Control Parameters
HCE	Host Card Emulation
KIF	Key Identifier
KVC	Key Version Code
LID	Long IDentifier (file id)
MAC	Message Authentication Code
PIN	Personal Identification Number
PKI	Public Key Infrastructure
SAM	Secure Application Module
SFI	Short File Identifier
SPI	Service Provider Interface
SV	Stored Value
UML	Unified Modelling Language

4. Architectural Overview

4.1. Functional positioning

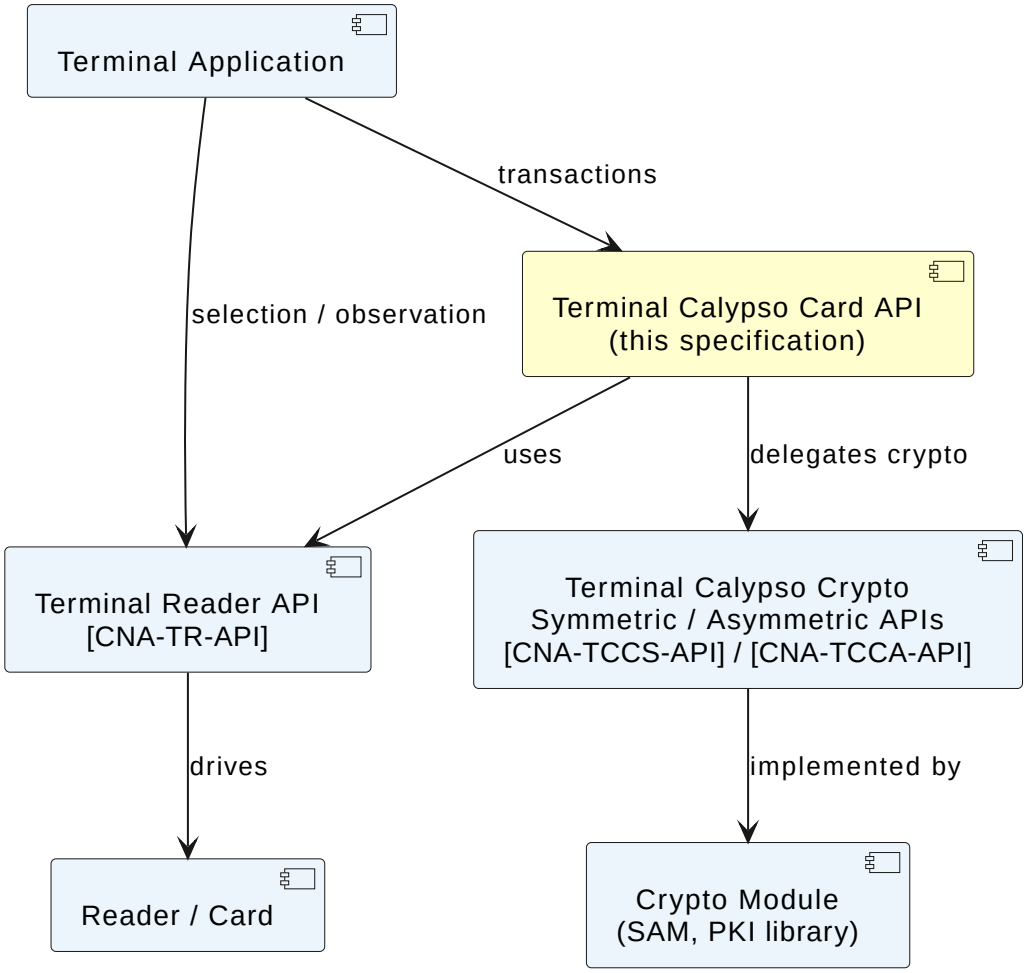

Figure 1. Terminal Calypso Card API — Architecture Overview

4.2. Logical namespaces

Namespace	Role	Summary
`calypso.card`	Public API	Factory, properties, enumerations (`GetDataTag`, `PutDataTag`, `SelectFileControl`, `WriteAccessLevel`).
`calypso.card.card`	Public API	Data model of a Calypso card: `CalypsoCard`, files, headers, SV logs and selection extension.
`calypso.card.transaction`	Public API	Transaction managers (Free / Secure Regular / Secure Extended / Secure PKI), security setting carriers, `SearchCommandData`, `SvAction`, `SvOperation` and the family of exceptions.
`calypso.card.transaction.spi`	SPI	Marker interfaces implemented by crypto extensions (`SymmetricCryptoCardTransactionManagerFactory`, `AsymmetricCryptoCardTransactionManagerFactory`, certificate markers, `CardTransactionCryptoExtension`).

4.3. UML class diagram

The following UML class diagram provides a visual representation of the types and relationships defined by this specification:

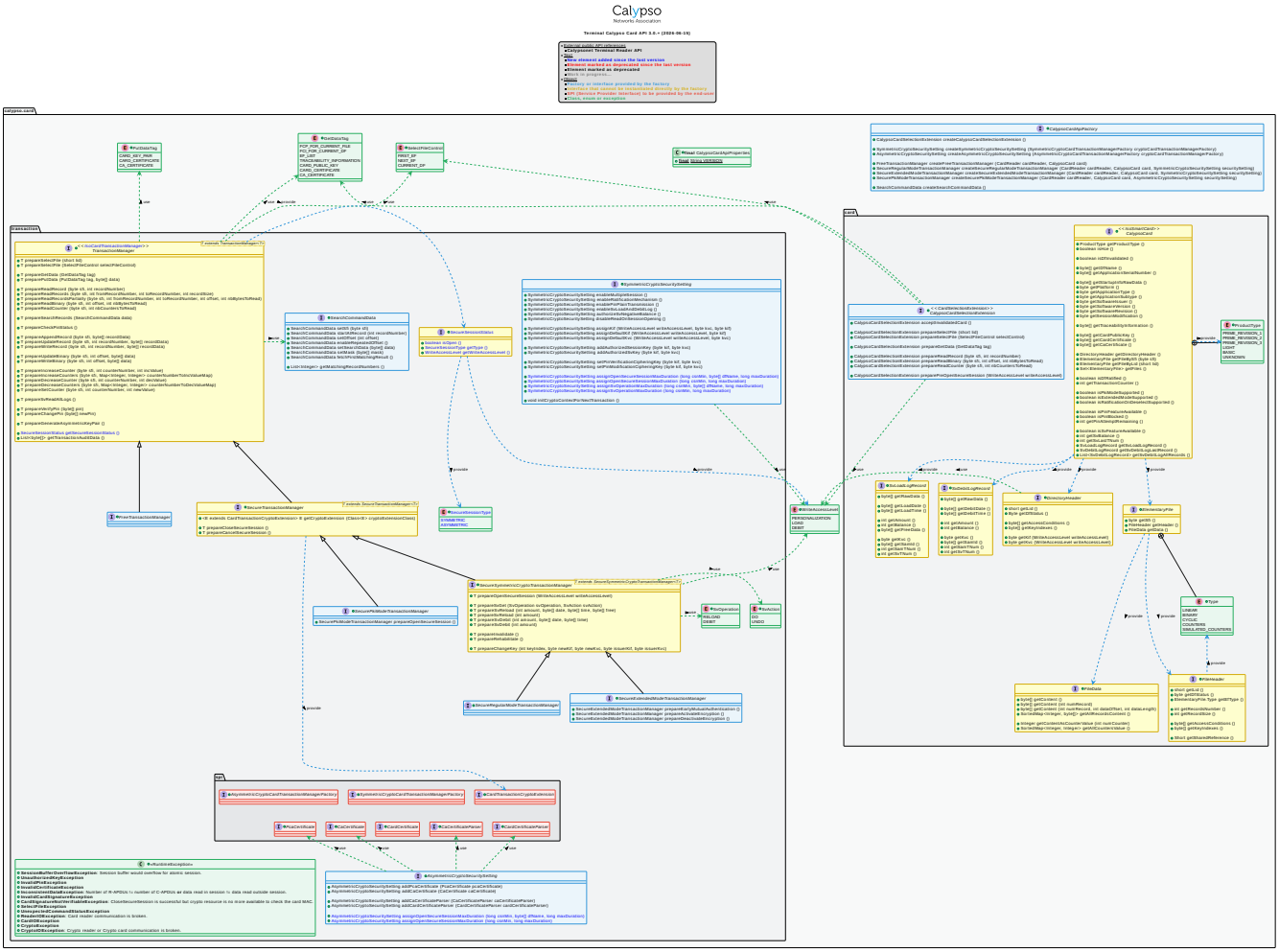

Figure 2. Terminal Calypso Card API — UML Class Diagram

5. API Specification

5.1. Classes

5.1.1. Calypso Card API Properties

Name	`CalypsoCardApiProperties`
Kind	Final class
Namespace	`calypso.card`
Since	1.0
Purpose	Exposes the immutable properties of the Terminal Calypso Card API.

Constants

Name	Type	Since	Description
`VERSION`	`string`	1.0	String representation of the API version (e.g. `"3.0"`).

5.2. API Interfaces

5.2.1. Asymmetric Crypto Security Setting

Name	`AsymmetricCryptoSecuritySetting`
Kind	Interface
Namespace	`calypso.card.transaction`
Since	2.1
Purpose	Security setting carrier for PKI Calypso card transactions. An instance is obtained via `CalypsoCardApiFactory.createAsymmetricCryptoSecuritySetting`.

Add CA Certificate

Signature	`addCaCertificate(`
	`  caCertificate: CaCertificate`
	`) → AsymmetricCryptoSecuritySetting`

Since	2.1
Description	Registers a CA certificate. The issuer's certificate MUST be loaded first. Preloading a CA certificate avoids having to read it from the card. Various checks are performed to ensure the integrity and validity of the supplied CA certificate: verification of the certificate's signature using the issuer's public key, check of the validity period to ensure the certificate is neither expired nor prematurely valid, confirmation of the authenticity of the issuer and subject details, and compliance with any constraints or extensions required for CA certificates.
Parameters	`caCertificate` (`CaCertificate`) — the CA certificate to register.
Returns	The current instance (`AsymmetricCryptoSecuritySetting`).
Throws	`IllegalStateException` — if the contained public key has already been registered. `InvalidCertificateException` — if the check of the supplied certificate failed.
See also	`addPcaCertificate`

Add CA Certificate Parser

Signature	`addCaCertificateParser(`
	`  caCertificateParser: CaCertificateParser`
	`) → AsymmetricCryptoSecuritySetting`

Since	2.1
Description	Registers a CA certificate parser, used when the CA certificate is not already available. The parser provides the means to build a CA certificate from the raw data read from the card.
Parameters	`caCertificateParser` (`CaCertificateParser`) — the CA certificate parser to register.
Returns	The current instance (`AsymmetricCryptoSecuritySetting`).
Throws	`IllegalStateException` — if a parser is already registered for the same certificate type.

Add Card Certificate Parser

Signature	`addCardCertificateParser(`
	`  cardCertificateParser: CardCertificateParser`
	`) → AsymmetricCryptoSecuritySetting`

Since	2.1
Description	Registers a card certificate parser. The parser provides the means to build a card certificate from the raw data read from the card. Only one parser MAY be registered per certificate type.
Parameters	`cardCertificateParser` (`CardCertificateParser`) — the card certificate parser to register.
Returns	The current instance (`AsymmetricCryptoSecuritySetting`).
Throws	`IllegalStateException` — if a parser associated with the same certificate type is already registered.

Add PCA Certificate

Signature	`addPcaCertificate(`
	`  pcaCertificate: PcaCertificate`
	`) → AsymmetricCryptoSecuritySetting`

Since	2.1
Description	Registers a self-signed PCA certificate. Various checks are performed to ensure the integrity and validity of the supplied PCA certificate: verification of the certificate's signature to ensure it is self-signed, check of the validity period to ensure the certificate is neither expired nor prematurely valid, confirmation of the authenticity of the issuer and subject details, and compliance with any constraints or extensions required for PCA certificates.
Parameters	`pcaCertificate` (`PcaCertificate`) — the self-signed PCA certificate to register.
Returns	The current instance (`AsymmetricCryptoSecuritySetting`).
Throws	`IllegalArgumentException` — if the certificate is invalid. `IllegalStateException` — if the contained public key has already been registered. `InvalidCertificateException` — if the check of the supplied certificate failed.

Assign Open Secure Session Max Duration (All DF)

Signature	`assignOpenSecureSessionMaxDuration(`
	`  csnMin: long,`
	`  maxDuration: long`
	`) → AsymmetricCryptoSecuritySetting`

Since	3.0

Description	Sets the maximum duration (in milliseconds) of an open PKI secure session for cards whose CSN is >= `csnMin`, applied to every application DF (no `dfName` filter). See the `csnMin` combination rule.
Parameters	`csnMin` — the lowest card serial number (CSN) the setting applies to. `maxDuration` — the maximum duration of an open secure session, in milliseconds.
Returns	The current instance (`AsymmetricCryptoSecuritySetting`).
Throws	None.

Assign Open Secure Session Max Duration (Per DF)

Signature	`assignOpenSecureSessionMaxDuration(`
	`  csnMin: long,`
	`  dfName: byte array,`
	`  maxDuration: long`
	`) → AsymmetricCryptoSecuritySetting`

Since	3.0
Description	Sets the maximum duration (in milliseconds) of an open PKI secure session for cards whose CSN is >= `csnMin` and whose application DF matches `dfName`. See the `csnMin` combination rule.
Parameters	`csnMin` — the lowest card serial number (CSN) the setting applies to. `dfName` — the application DF name the setting applies to. `maxDuration` — the maximum duration of an open secure session, in milliseconds.
Returns	The current instance (`AsymmetricCryptoSecuritySetting`).
Throws	None.

5.2.2. Calypso Card

Name	`CalypsoCard`
Kind	Interface
Namespace	`calypso.card.card`
Extends	`IsoSmartCard` (CNA-TR-API)
Since	1.0
Purpose	Dynamic view of the card's content, updated from selection to the end of the transaction.

An instance of `CalypsoCard` is obtained by casting the `IsoSmartCard` object produced by the selection process defined in CNA-TR-API.

A `CalypsoCard` carries: application identification (revision, class, DF name, serial number, ATR, issuer), the indication of optional features (Stored Value, PIN, Rev 3.2 mode, ratification management), the management information of the modification buffer, the invalidation status and the files / counters / SV data read or modified during the transaction.

The `CalypsoCard` instance bound to a transaction is a **dynamic view** of the card image, updated after every successful command. Failed write commands do not update the in-memory image.

Get Application Serial Number

Signature	`getApplicationSerialNumber() → byte array`
Since	1.0
Description	Returns the Application Serial Number, an 8-byte value uniquely identifying the card application.
Returns	A non-empty `byte array`.

Throws	None.

Get Application Subtype

Signature	`getApplicationSubtype() → byte`
Since	1.0
Description	Returns the application subtype byte, which references the card's file structure.
Returns	A `byte` value.
Throws	None.

Get Application Type

Signature	`getApplicationType() → byte`
Since	1.0
Description	Returns the application type byte, which encodes the product type together with its options.
Returns	A `byte` value.
Throws	None.

Get CA Certificate

Signature	`getCaCertificate() → byte array`
Since	2.1
Description	CA certificate; empty if not available.
Returns	A `byte array`; empty if not available.
Throws	None.

See also	`CalypsoCardSelectionExtension.prepareGetData`
	`TransactionManager.prepareGetData`
	`SecurePkiModeTransactionManager.prepareOpenSecureSession`

Get Card Certificate

Signature	`getCardCertificate() → byte array`
Since	2.1
Description	Card certificate; empty if not available.
Returns	A `byte array`; empty if not available.
Throws	None.

See also	`CalypsoCardSelectionExtension.prepareGetData`
	`TransactionManager.prepareGetData`
	`SecurePkiModeTransactionManager.prepareOpenSecureSession`

Get Card Public Key

Signature	`getCardPublicKey() → byte array`
Since	2.1
Description	Card public key; empty if not available.
Returns	A `byte array`; empty if not available.

Throws	None.

See also	`CalypsoCardSelectionExtension.prepareGetData`
	`TransactionManager.prepareGetData`
	`SecurePkiModeTransactionManager.prepareOpenSecureSession`

Get DF Name

Signature	`getDfName() → byte array`
Since	1.0
Description	DF name (5 to 16 bytes), as per ISO/IEC 7816-4. It also corresponds to the complete representation of the target covered by the AID value supplied in the selection command: the AID selects the application by specifying all or part of the targeted DF name (5 bytes minimum).
Returns	A non-empty `byte array`.
Throws	None.

Get Directory Header

Signature	`getDirectoryHeader() → DirectoryHeader?`
Since	1.0
Description	Metadata of the current DF; `null` if not set.
Returns	A `DirectoryHeader`, or `null` if not set.
Throws	None.

See also	`CalypsoCardSelectionExtension.prepareSelectFile(short)`
	`CalypsoCardSelectionExtension.prepareSelectFile(SelectFileControl)`
	`TransactionManager.prepareSelectFile(short)`
	`TransactionManager.prepareSelectFile(SelectFileControl)`

Get File By LID

Signature	`getFileByLid(lid: short) → ElementaryFile?`
Since	1.0
Description	EF whose LID matches; `null` if not found. Note that when a secure session is running, the returned object carries all in-session modifications, which are rolled back if the secure session fails.
Parameters	`lid`—the LID to search.
Returns	An `ElementaryFile`, or `null` if not found.
Throws	None.

Get File By SFI

Signature	`getFileBySfi(sfi: byte) → ElementaryFile?`
Since	1.0
Description	EF whose SFI matches; `null` if not found, or if `sfi == 0`. Note that when a secure session is running, the returned object carries all in-session modifications, which are rolled back if the secure session fails.
Parameters	`sfi`—the SFI to search.

Returns	An `ElementaryFile`, or `null` if not found, or if `sfi == 0`.
Throws	None.

Get Files

Signature	`getFiles() → set<ElementaryFile>`
Since	1.1
Description	All known EFs of the current DF. During a secure session, this set reflects every in-session modification (which is rolled back on session failure).
Returns	A `set<ElementaryFile>`; empty if no EF is known.
Throws	None.

Get PIN Attempt Remaining

Signature	`getPinAttemptRemaining() → integer`
Since	1.0
Description	Returns the number of PIN presentation attempts still remaining before the PIN becomes blocked.
Returns	An `integer` value.
Throws	`IllegalStateException` — if the PIN has not been checked.
See also	`TransactionManager.prepareCheckPinStatus` `TransactionManager.prepareVerifyPin`

Get Platform

Signature	`getPlatform() → byte`
Since	1.0
Description	Returns the platform identification byte, which references the card's chip.
Returns	A `byte` value.
Throws	None.

Get Product Type

Signature	`getProductType() → ProductType`
Since	1.0
Description	Returns the card product type (`PRIME_REVISION_1` / `PRIME_REVISION_2` / `PRIME_REVISION_3` / `LIGHT` / `BASIC` / `UNKNOWN`).
Returns	A `ProductType`.
Throws	None.

Get Session Modification

Signature	`getSessionModification() → byte`
Since	1.0
Description	Returns the session modification byte, which — depending on the card — caps either the number of bytes modifiable or the number of write commands allowed within a secure session.
Returns	A `byte` value.

Throws	None.

Get Software Issuer

Signature	`getSoftwareIssuer() → byte`
Since	1.0
Description	Returns the software issuer byte, identifying the issuer of the card's embedded software.
Returns	A `byte` value.
Throws	None.

Get Software Revision

Signature	`getSoftwareRevision() → byte`
Since	1.0
Description	Returns the software revision byte of the card's embedded software.
Returns	A `byte` value.
Throws	None.

Get Software Version

Signature	`getSoftwareVersion() → byte`
Since	1.0
Description	Returns the software version byte of the card's embedded software.
Returns	A `byte` value.
Throws	None.

Get Startup Info Raw Data

Signature	`getStartupInfoRawData() → byte array`
Since	1.0
Description	Returns the raw Calypso startup information block—the sequence of bytes from which the individual startup fields (platform, application type and subtype, software issuer, version and revision, etc.) are derived.
Returns	A non-empty `byte array`.
Throws	None.

Get SV Balance

Signature	`getSvBalance() → integer`
Since	1.0
Description	Returns the current Stored Value (SV) balance held by the card, expressed in its monetary unit.
Returns	An `integer` value.
Throws	`IllegalStateException`—if no SV Get has been executed.

See also	`SecureSymmetricCryptoTransactionManager.prepareSvGet`
	`SecureSymmetricCryptoTransactionManager.prepareSvDebit(int)`
	`SecureSymmetricCryptoTransactionManager.prepareSvDebit(int, byte[], byte[])`
	`SecureSymmetricCryptoTransactionManager.prepareSvReload(int)`
	`SecureSymmetricCryptoTransactionManager.prepareSvReload(int, byte[], byte[], byte[])`

Get SV Debit Log All Records

Signature	`getSvDebitLogAllRecords() → list<SvDebitLogRecord>`
Since	1.0
Description	All SV debit log records read from the card.
Returns	A `list<SvDebitLogRecord>`; empty if no debit log record has been read.
Throws	None.

See also	`SecureSymmetricCryptoTransactionManager.prepareSvGet`
	`SecureSymmetricCryptoTransactionManager.prepareSvDebit(int)`
	`SecureSymmetricCryptoTransactionManager.prepareSvDebit(int, byte[], byte[])`
	`SecureSymmetricCryptoTransactionManager.prepareSvReload(int)`
	`SecureSymmetricCryptoTransactionManager.prepareSvReload(int, byte[], byte[], byte[])`

Get SV Debit Log Last Record

Signature	`getSvDebitLogLastRecord() → SvDebitLogRecord?`
Since	1.0
Description	Last SV debit log record; `null` if not available.
Returns	A `SvDebitLogRecord`, or `null` if not available.
Throws	None.

See also	`SecureSymmetricCryptoTransactionManager.prepareSvGet`
	`SecureSymmetricCryptoTransactionManager.prepareSvDebit(int)`
	`SecureSymmetricCryptoTransactionManager.prepareSvDebit(int, byte[], byte[])`
	`SecureSymmetricCryptoTransactionManager.prepareSvReload(int)`
	`SecureSymmetricCryptoTransactionManager.prepareSvReload(int, byte[], byte[], byte[])`

Get SV Last T Num

Signature	`getSvLastTNum() → integer`
Since	1.0
Description	Returns the number of the last Stored Value (SV) transaction recorded on the card.
Returns	An `integer` value.
Throws	`IllegalStateException` — if no SV Get command has been executed.

See also	`SecureSymmetricCryptoTransactionManager.prepareSvGet`
	`SecureSymmetricCryptoTransactionManager.prepareSvDebit(int)`
	`SecureSymmetricCryptoTransactionManager.prepareSvDebit(int, byte[], byte[])`
	`SecureSymmetricCryptoTransactionManager.prepareSvReload(int)`
	`SecureSymmetricCryptoTransactionManager.prepareSvReload(int, byte[], byte[], byte[])`

Get SV Load Log Record

Signature	`getSvLoadLogRecord() → SvLoadLogRecord?`
Since	1.0

Description	Last SV load log record; `null` if not available.
Returns	A `SvLoadLogRecord`, or `null` if not available.
Throws	None.

See also	`SecureSymmetricCryptoTransactionManager.prepareSvGet`
	`SecureSymmetricCryptoTransactionManager.prepareSvDebit(int)`
	`SecureSymmetricCryptoTransactionManager.prepareSvDebit(int, byte[], byte[])`
	`SecureSymmetricCryptoTransactionManager.prepareSvReload(int)`
	`SecureSymmetricCryptoTransactionManager.prepareSvReload(int, byte[], byte[], byte[])`

Get Traceability Information

Signature	`getTraceabilityInformation() → byte array`
Since	1.1
Description	Traceability information (software issuer ID and discretionary data); empty if not available.
Returns	A `byte array`; empty if not available.
Throws	None.
See also	`CalypsoCardSelectionExtension.prepareGetData` `TransactionManager.prepareGetData`

Get Transaction Counter

Signature	`getTransactionCounter() → integer`
Since	1.2
Description	Transaction counter from the output of the last successful **Open Secure Session**. **Note:** commands such as **Change Key**, **Change/Verify PIN**, **SV Debit/Undebit/Reload** decrement the card counter but do not update this returned value.
Returns	An `integer` value.
Throws	`IllegalStateException` — if no session has been opened.
See also	`CalypsoCardSelectionExtension.preparePreOpenSecureSession` `SecureSymmetricCryptoTransactionManager.prepareOpenSecureSession`

Is DF Invalidated

Signature	`isDfInvalidated() → boolean`
Since	1.0
Description	Tells whether the current DF is invalidated. The invalidation status is determined either from the response to the **Select Application** command or from the response to a **Select File** (DF) command. For a `PRIME_REVISION_3` card, a `6283h` status word is returned in response to the **Select Application** command when the corresponding DF is invalidated. For older Calypso cards, it may be necessary to execute a **Select File** command in order to determine the invalidation status.
Returns	`true` if the current DF has been invalidated, `false` otherwise.
Throws	None.

Is DF Ratified

Signature	`isDfRatified() → boolean`
Since	1.0

Description	Tells whether the last session has been ratified.
Returns	A `boolean` value.
Throws	`IllegalStateException` — if no session has been opened.
See also	`CalypsoCardSelectionExtension.preparePreOpenSecureSession` `SecureSymmetricCryptoTransactionManager.prepareOpenSecureSession`

Is Extended Mode Supported

Signature	`isExtendedModeSupported() → boolean`
Since	1.0
Description	Indicates whether the Extended Mode is supported. This indication is initially interpreted from the **Application Type** byte, but MAY be updated once the secure session is opened: depending on the type of key used, the extended mode functionalities may not be available (for example with a non-AES key).
Returns	A `boolean` value.
Throws	None.

Is HCE

Signature	`isHce() → boolean`
Since	1.0
Description	Indicates whether the card is a Calypso HCE.
Returns	A `boolean` value.
Throws	None.

Is PIN Blocked

Signature	`isPinBlocked() → boolean`
Since	1.0
Description	Indicates whether the PIN is blocked.
Returns	A `boolean` value.
Throws	`IllegalStateException` — if the PIN has not been checked.
See also	`TransactionManager.prepareCheckPinStatus` `TransactionManager.prepareVerifyPin`

Is PIN Feature Available

Signature	`isPinFeatureAvailable() → boolean`
Since	1.0
Description	Indicates whether the Calypso PIN feature is available. This indication is interpreted from the **Application Type** byte.
Returns	`true` if the card has the PIN feature, `false` otherwise.
Throws	None.

Is PKI Mode Supported

Signature	`isPkiModeSupported() → boolean`

Since	1.0
Description	Indicates whether Public Key Authentication is supported. This indication is interpreted from the **Application Type** byte.
Returns	A `boolean` value.
Throws	None.

Is Ratification On Deselect Supported

Signature	`isRatificationOnDeselectSupported() → boolean`
Since	1.0
Description	Indicates whether ratification happens on deselect (ratification command not required). This indication is interpreted from the **Application Type** byte.
Returns	A `boolean` value.
Throws	None.

Is SV Feature Available

Signature	`isSvFeatureAvailable() → boolean`
Since	1.0
Description	Indicates whether the SV feature is available. This indication is interpreted from the **Application Type** byte.
Returns	`true` if the card has the Stored Value feature, `false` otherwise.
Throws	None.

5.2.3. Calypso Card API Factory

Name	`CalypsoCardApiFactory`
Kind	Interface
Namespace	`calypso.card`
Since	2.0
Purpose	Factory used to obtain instances of every public type provided by the API.

Create Asymmetric Crypto Security Setting

Signature	`createAsymmetricCryptoSecuritySetting(`
	`  cryptoCardTransactionManagerFactory:`
	`AsymmetricCryptoCardTransactionManagerFactory`
	`) → AsymmetricCryptoSecuritySetting`

Since	2.1
Description	Creates the `AsymmetricCryptoSecuritySetting` that carries the asymmetric-key (PKI) cryptographic configuration required by `createSecurePkiModeTransactionManager`. The supplied factory provides the cryptographic module that performs those computations.
Parameters	`cryptoCardTransactionManagerFactory` (`AsymmetricCryptoCardTransactionManagerFactory`) — the factory of the crypto card transaction manager to be used.
Returns	An `AsymmetricCryptoSecuritySetting`.
Throws	`IllegalArgumentException` — if the factory is invalid.

Create Calypso Card Selection Extension

Signature	`createCalypsoCardSelectionExtension() → CalypsoCardSelectionExtension`
Since	2.0
Description	Creates a `CalypsoCardSelectionExtension` that enriches a CNA-TR-API selection scenario with Calypso-specific commands and the optional pre-opening of a secure session.
Returns	A `CalypsoCardSelectionExtension`.
Throws	None.

Create Free Transaction Manager

Signature	`createFreeTransactionManager(`
	`  cardReader: CardReader,`
	`  card: CalypsoCard`
	`) → FreeTransactionManager`

Since	2.0
Description	Creates a `FreeTransactionManager` for exchanges with `card` (reached through `cardReader`) that require **no** cryptographic computation—that is, no secure session and no security setting. For secured transactions, use one of the `createSecure*ModeTransactionManager` factories instead.
Parameters	`cardReader` (`CardReader` (CNA-TR-API))—the reader through which the card is reached. `card` (`CalypsoCard`)—the selected card on which to operate the transaction.
Returns	A `FreeTransactionManager`.
Throws	None.

Create Search Command Data

Signature	`createSearchCommandData() → SearchCommandData`
Since	2.0
Description	Creates an empty `SearchCommandData` used to configure and collect the results of `TransactionManager.prepareSearchRecords`.
Returns	A `SearchCommandData`.
Throws	None.

Create Secure Extended Mode Transaction Manager

Signature	`createSecureExtendedModeTransactionManager(`
	`  cardReader: CardReader,`
	`  card: CalypsoCard,`
	`  securitySetting: SymmetricCryptoSecuritySetting`
	`) → SecureExtendedModeTransactionManager`

Since	2.0
Description	Creates a `SecureExtendedModeTransactionManager` that runs a symmetric-key secure session in **extended mode**, adding the operations available only on **Calypso Prime Extended** products (e.g. session encryption and early mutual authentication). For other products, use `createSecureRegularModeTransactionManager`. Cryptographic operations rely on the supplied `securitySetting`.
Parameters	`cardReader` (`CardReader` (CNA-TR-API))—the reader through which the card is reached. `card` (`CalypsoCard`)—the selected card on which to operate the transaction. `securitySetting` (`SymmetricCryptoSecuritySetting`)—the security setting to be used.
Returns	A `SecureExtendedModeTransactionManager`.

Throws	None.

Create Secure PKI Mode Transaction Manager

Signature	`createSecurePkiModeTransactionManager(`
	`  cardReader: CardReader,`
	`  card: CalypsoCard,`
	`  securitySetting: AsymmetricCryptoSecuritySetting`
	`) → SecurePkiModeTransactionManager`

Since	2.1
Description	Creates a `SecurePkiModeTransactionManager` that runs a secure session in **PKI mode**, using asymmetric-key (PKI) cryptography. PKI mode is available only if the card declares support for it (`CalypsoCard.isPkiModeSupported() == true`). For symmetric-key sessions, use `createSecureRegularModeTransactionManager` or `createSecureExtendedModeTransactionManager` instead. Cryptographic operations rely on the supplied `securitySetting`.
Parameters	`cardReader` (`CardReader` (CNA-TR-API)) — the reader through which the card is reached. `card` (`CalypsoCard`) — the selected card on which to operate the transaction. `securitySetting` (`AsymmetricCryptoSecuritySetting`) — the security setting to be used.
Returns	A `SecurePkiModeTransactionManager`.
Throws	`IllegalStateException` — if the card does not support PKI mode (`isPkiModeSupported() == false`).

Create Secure Regular Mode Transaction Manager

Signature	`createSecureRegularModeTransactionManager(`
	`  cardReader: CardReader,`
	`  card: CalypsoCard,`
	`  securitySetting: SymmetricCryptoSecuritySetting`
	`) → SecureRegularModeTransactionManager`

Since	2.0
Description	Creates a `SecureRegularModeTransactionManager` that runs a symmetric-key secure session in **regular mode**, the baseline mode compatible with every Calypso product. To access **Calypso Prime Extended** features use `createSecureExtendedModeTransactionManager`, or `createSecurePkiModeTransactionManager` for PKI cards. Cryptographic operations rely on the supplied `securitySetting`.
Parameters	`cardReader` (`CardReader` (CNA-TR-API)) — the reader through which the card is reached. `card` (`CalypsoCard`) — the selected card on which to operate the transaction. `securitySetting` (`SymmetricCryptoSecuritySetting`) — the security setting to be used.
Returns	A `SecureRegularModeTransactionManager`.
Throws	None.

Create Symmetric Crypto Security Setting

Signature	`createSymmetricCryptoSecuritySetting(`
	`  cryptoCardTransactionManagerFactory: SymmetricCryptoCardTransactionManagerFactory`
	`) → SymmetricCryptoSecuritySetting`

Since	2.0
Description	Creates the `SymmetricCryptoSecuritySetting` that carries the symmetric-key (e.g. SAM-backed) cryptographic configuration required by the regular- and extended-mode secure transaction managers. The supplied factory provides the cryptographic module that performs those computations.

Parameters	`cryptoCardTransactionManagerFactory` (`SymmetricCryptoCardTransactionManagerFactory`) — the factory of the crypto card transaction manager to be used.
Returns	A `SymmetricCryptoSecuritySetting`.
Throws	`IllegalArgumentException` — if the factory is invalid.

5.2.4. Calypso Card Selection Extension

Name	`CalypsoCardSelectionExtension`
Kind	Interface
Namespace	`calypso.card.card`
Extends	`CardSelectionExtension` (CNA-TR-API)
Since	2.0
Purpose	Enriches the CNA-TR-API selection scenario with Calypso-specific commands and pre-opening of a secure session. An instance is obtained via `CalypsoCardApiFactory.createCalypsoCardSelectionExtension`.

By default the selection rejects PRIME revision 3 cards that have been **invalidated**. The application MUST call `acceptInvalidatedCard` to stop ignoring them. For earlier revisions, the application MAY have to handle invalidation itself (typically via a **Select File** on the DF).

For all `prepare*` operations, unless otherwise specified, the following parameter ranges apply:

Parameter	Range
SFI	`[0..30]` (`0` indicates the current EF)
Record number	`[1..250]`
Counter number	`[1..83]`
Counter value	`[0..16777215]`
Offset	`[0..249]`, or `[0..32767]` for binary files
Input data length	`[1..250]`, or `[1..32767]` for binary files

Accept Invalidated Card

Signature	`acceptInvalidatedCard() → CalypsoCardSelectionExtension`
Since	1.0
Description	Requests to accept invalidated cards during the selection stage. **Works only with cards that indicate their invalidation status at the time of selection** (e.g., all PRIME Revision 3+ cards and certain PRIME Revision 2 cards).
Returns	The current instance (`CalypsoCardSelectionExtension`).
Throws	None.

Prepare Get Data

Signature	`prepareGetData(tag: GetDataTag) → CalypsoCardSelectionExtension`
Since	1.0

Description	Adds a **Get Data** command for the supplied tag. This is the way to obtain FCI information when it is not supplied directly by **Select Application** (e.g. the OMAPI case). **Caution:** the resulting APDU command is compliant with PRIME revision 3 cards. It may therefore be rejected by some earlier revision cards.
Parameters	`tag` (`GetDataTag`) — the data object tag to read.
Returns	The current instance (`CalypsoCardSelectionExtension`).
Throws	None.

Prepare Pre Open Secure Session

Signature	`preparePreOpenSecureSession(`
	`  writeAccessLevel: WriteAccessLevel`
	`) → CalypsoCardSelectionExtension`

Since	1.7
Description	Adds a specific **Open Secure Session** command attempting a **pre-opening**. Enables a future single-exchange session execution by anticipating the APDU responses. The objective of the pre-opening is to allow the grouping of all the commands of a secure session. It is only relevant for a distributed system where the ticketing processing is done remotely, so that a complete secure session can be carried out in a single exchange between the server and the terminal. To achieve that single exchange, all the data that will have to be read in session MUST first be read locally, outside the session — otherwise additional exchanges take place. The remote ticketing processing then prepares all the commands of the session, from opening to closing, before executing it: `prepareOpenSecureSession(…)`, then the other `prepare*` calls, then `prepareCloseSecureSession()`, and finally a single `processCommands()`. The mechanism is **ineffective** if: • the card or crypto module does not support extended mode; • the session needs intermediate exchanges (PIN, SV, encryption, early mutual authentication, unread session data); • the session opens with a different access level; • an intermediate `processCommands()` is called; • asymmetric cryptography is used.
Parameters	`writeAccessLevel` (`WriteAccessLevel`) — the write access level of the session to pre-open.
Returns	The current instance (`CalypsoCardSelectionExtension`).
Throws	`IllegalStateException` — if pre-open is already prepared.

Prepare Read Binary

Signature	`prepareReadBinary(`
	`  sfi: byte,`
	`  offset: integer,`
	`  nbBytesToRead: integer`
	`) → CalypsoCardSelectionExtension`

Since	1.7

Description	Adds one or more **Read Binary** commands to read all or part of the supplied binary EF. Once processed, the result is available in the `CalypsoCard` if the requested file exists in the file structure of the card and if the offset and the number of bytes to read are valid (**best-effort** mode). **Caution:** the resulting APDU command is compliant with PRIME revision 3 cards. It may therefore be rejected by some earlier revision cards.
Parameters	`sfi` — the SFI of the EF. `offset` — the offset (0 indicates the first byte). `nbBytesToRead` — the number of bytes to read.
Returns	The current instance (`CalypsoCardSelectionExtension`).
Throws	`IllegalArgumentException` — if one of the supplied arguments is out of range.

Prepare Read Counter

Signature	`prepareReadCounter(`
	`  sfi: byte,`
	`  nbCountersToRead: integer`
	`) → CalypsoCardSelectionExtension`

Since	1.7
Description	Adds a **Read Records** command to read a part of a record of the supplied EF, which SHOULD be a counter file. The record is read up to the supplied counter number, so all preceding counters are also read. Once processed, the result is available in the `CalypsoCard` if the requested file and counter number exist in the file structure of the card (**best-effort** mode). **Caution:** the resulting APDU command is compliant with PRIME revision 3 cards. It may therefore be rejected by some earlier revision cards.
Parameters	`sfi` — the SFI of the EF. `nbCountersToRead` — the number of counters to read.
Returns	The current instance (`CalypsoCardSelectionExtension`).
Throws	`IllegalArgumentException` — if one of the supplied arguments is out of range.

Prepare Read Record

Signature	`prepareReadRecord(`
	`  sfi: byte,`
	`  recordNumber: integer`
	`) → CalypsoCardSelectionExtension`

Since	1.1
Description	Adds a **Read Records** command for a single record of a linear or cyclic EF. Once processed, the result is available in the `CalypsoCard` if the requested file and record exist in the file structure of the card (**best-effort** mode: the command does not fail if the file or the record is absent). **Caution:** the resulting APDU command is compliant with PRIME revision 3 cards. It may therefore be rejected by some earlier revision cards.
Parameters	`sfi` — the SFI of the EF to read. `recordNumber` — the record number to read.
Returns	The current instance (`CalypsoCardSelectionExtension`).
Throws	`IllegalArgumentException` — if one of the supplied arguments is out of range.

Prepare Select File (LID)

Signature	`prepareSelectFile(lid: short) → CalypsoCardSelectionExtension`

Since	1.0
Description	Adds a **Select File** command targeting an EF by its LID. Caution: PRIME 3 compatible only. The command fails if the selected file is not an EF.
Parameters	`lid` — the LID of the EF to select.
Returns	The current instance (`CalypsoCardSelectionExtension`).
Throws	None.

Prepare Select File (Select Control)

Signature	`prepareSelectFile(`
	`  selectControl: SelectFileControl`
	`) → CalypsoCardSelectionExtension`

Since	1.0
Description	Adds a **Select File** command using a navigation control (`FIRST`, `NEXT` or `CURRENT`). **Caution:** the resulting APDU command is compliant with PRIME revision 3 cards. It may therefore be rejected by some earlier revision cards.
Parameters	`selectControl` (`SelectFileControl`) — the navigation control to apply.
Returns	The current instance (`CalypsoCardSelectionExtension`).
Throws	None.

5.2.5. Directory Header

Name	`DirectoryHeader`
Kind	Interface
Namespace	`calypso.card.card`
Since	1.0
Purpose	Carries the metadata of a Calypso DF.

Get Access Conditions

Signature	`getAccessConditions() → byte array`
Since	1.0
Description	Returns a reference to the access conditions.
Returns	A non-empty `byte array`.
Throws	None.

Get DF Status

Signature	`getDfStatus() → byte`
Since	1.0
Description	Returns the status byte of the DF (Dedicated File) described by this header.
Returns	A `byte` value.
Throws	None.

Get Key Indexes

Signature	`getKeyIndexes() → byte array`
Since	1.0
Description	Returns a reference to the keys indexes.
Returns	A non-empty `byte array`.
Throws	None.

Get KIF

Signature	`getKif(writeAccessLevel: WriteAccessLevel) → byte`
Since	1.0
Description	Returns the KIF associated with the supplied write access level.
Parameters	`writeAccessLevel` (`WriteAccessLevel`) — the write access level whose KIF is requested.
Returns	A `byte` value.
Throws	None.

Get KVC

Signature	`getKvc(writeAccessLevel: WriteAccessLevel) → byte`
Since	1.0
Description	Returns the KVC associated with the supplied write access level.
Parameters	`writeAccessLevel` (`WriteAccessLevel`) — the write access level whose KVC is requested.
Returns	A `byte` value.
Throws	None.

Get LID

Signature	`getLid() → short`
Since	1.0
Description	Returns the LID, the 2-byte identifier of the associated file structure (DF or EF).
Returns	A `short` value.
Throws	None.

5.2.6. Elementary File

Name	`ElementaryFile`
Kind	Interface
Namespace	`calypso.card.card`
Since	1.0
Purpose	Calypso Elementary File. Carries an `SFI`, a `FileHeader` and a `FileData`.

Get Data

Signature	`getData() → FileData`

Since	1.0
Description	Returns the `FileData` object carrying this Elementary File's content.
Returns	A `FileData`.
Throws	None.

Get Header

Signature	`getHeader() → FileHeader?`
Since	1.0
Description	Returns the file header, or `null` if not yet set.
Returns	A `FileHeader`, or `null` if not yet set.
Throws	None.

Get SFI

Signature	`getSfi() → byte`
Since	1.0
Description	Returns the Short File Identifier (SFI) of this Elementary File.
Returns	A `byte` value.
Throws	None.

5.2.7. File Data

Name	`FileData`
Kind	Interface
Namespace	`calypso.card.card`
Since	1.0
Purpose	Carries the content of a Calypso EF and exposes counter-oriented accessors.

Get All Counters Value

Signature	`getAllCountersValue() → sortedMap<integer, integer>`
Since	1.0
Description	Returns the values of all counters extracted from record #1.
Returns	A `sortedMap<integer, integer>`; empty if no counter is known.
Throws	None.

Get All Records Content

Signature	`getAllRecordsContent() → sortedMap<integer, byte array>`
Since	1.0
Description	Returns the known content of every record, keyed by record number.
Returns	A `sortedMap<integer, byte array>`; empty if no record content is known.
Throws	None.

Get Content (No Args)

Signature	`getContent() → byte array`
Since	1.0
Description	Returns the known content of record #1 (or the entire content for a binary file).
Returns	A `byte array`; empty if the content is not known.
Throws	None.

Get Content (Num Record)

Signature	`getContent(numRecord: integer) → byte array`
Since	1.0
Description	Returns the known content of the supplied record number.
Parameters	`numRecord`—the record number.
Returns	A `byte array`; empty if the record content is not known.
Throws	None.

Get Content (Num Record, Data Offset, Data Length)

Signature	`getContent(`
	`  numRecord: integer,`
	`  dataOffset: integer,`
	`  dataLength: integer`
	`) → byte array`

Since	1.0
Description	Returns a copy of the known content subset of a record from `dataOffset` to `dataOffset + dataLength`.
Parameters	`numRecord`—the record number. `dataOffset`—the offset index (should be >= 0). `dataLength`—the data length (should be >= 1).
Returns	A `byte array`; empty if the content is not known.
Throws	`IllegalArgumentException`—if `dataOffset < 0` or `dataLength < 1`. `IndexOutOfBoundsException`—if the range exceeds the record content.

Get Content As Counter Value

Signature	`getContentAsCounterValue(numCounter: integer) → integer?`
Since	1.0
Description	Returns the value of counter `numCounter`. The counter value is extracted from the 3-byte slice at offset `(numCounter - 1) * 3` of record #1. Returns `null` if record #1 or the requested counter is not set.
Parameters	`numCounter`—the counter number (should be >= 1).
Returns	An `integer` value, or `null` if record #1 or the requested counter is not set.
Throws	`IllegalArgumentException`—if `numCounter < 1`. `IndexOutOfBoundsException`—if the counter is truncated.

5.2.8. File Header

Name	`FileHeader`
Kind	Interface
Namespace	`calypso.card.card`
Since	1.0
Purpose	Carries the metadata of a Calypso EF.

Get Access Conditions

Signature	`getAccessConditions() → byte array`
Since	1.0
Description	Returns the access conditions; empty array if not available.
Returns	A `byte array`; empty if not available.
Throws	None.

Get DF Status

Signature	`getDfStatus() → byte?`
Since	1.0
Description	Returns the DF status, or `null` if not available (e.g. when the header is built from the response to a **Get Data** with `EF_LIST` tag).
Returns	A `byte` value, or `null` if not available.
Throws	None.

Get EF Type

Signature	`getEfType() → ElementaryFile.Type`
Since	1.0
Description	Returns the type of this Elementary File (EF), one of the `ElementaryFile.Type` values.
Returns	An `ElementaryFile.Type`.
Throws	None.

Get Key Indexes

Signature	`getKeyIndexes() → byte array`
Since	1.0
Description	Returns the key indexes; empty array if not available.
Returns	A `byte array`; empty if not available.
Throws	None.

Get LID

Signature	`getLid() → short`
Since	1.0

Description	Returns the LID, the 2-byte identifier of the associated file structure (DF or EF).
Returns	A `short` value.
Throws	None.

Get Record Size

Signature	`getRecordSize() → integer`
Since	1.0
Description	Returns the size of a record. For a **counter** file, this is the original size of record #1; the extra bytes—the remainder of the division of the file size by 3—are not accessible. For a **binary** file, this is the size of the file.
Returns	An `integer` value.
Throws	None.

Get Records Number

Signature	`getRecordsNumber() → integer`
Since	1.0
Description	Returns the number of records. For a **counter** file the number of records is always `1`, and for a **binary** file it is always `1` as well. For a counter file, the extra bytes—the remainder of the division of the file size by 3—are not accessible.
Returns	An `integer` value.
Throws	None.

Get Shared Reference

Signature	`getSharedReference() → short?`
Since	1.0
Description	Returns the non-zero unique identifier of the shared data when the file data is shared. Returns `0` if not shared, `null` if the information is not available.
Returns	A `short` value (`0` if not shared), or `null` if the information is not available.
Throws	None.

5.2.9. Free Transaction Manager

Name	`FreeTransactionManager`
Kind	Interface
Namespace	`calypso.card.transaction`
Extends	`TransactionManager<FreeTransactionManager>`
Since	2.0
Purpose	Transaction manager that does **not** require any cryptographic computation. An instance is obtained via `CalypsoCardApiFactory.createFreeTransactionManager`.

`FreeTransactionManager` does not add any operation.

5.2.10. Search Command Data

Name	`SearchCommandData`
Kind	Interface
Namespace	`calypso.card.transaction`
Since	1.1
Purpose	Carries the input/output data of `TransactionManager.prepareSearchRecords`. An instance is obtained via `CalypsoCardApiFactory.createSearchCommandData`.

Enable Repeated Offset

Signature	`enableRepeatedOffset() → SearchCommandData`
Since	1.1
Description	Analyse the data at the supplied offset, then repeatedly at each following offset until the end of the record.
Returns	The current instance (`SearchCommandData`).
Throws	None.

Fetch First Matching Result

Signature	`fetchFirstMatchingResult() → SearchCommandData`
Since	1.1
Description	Requests to fetch the content of the first matching record into the `CalypsoCard`.
Returns	The current instance (`SearchCommandData`).
Throws	None.

Get Matching Record Numbers

Signature	`getMatchingRecordNumbers() → list<integer>`
Since	1.1
Description	Returns the record numbers that matched.
Returns	A `list<integer>`; empty if no record matched or if the command has not yet been processed.
Throws	None.

Set Mask

Signature	`setMask(mask: byte array) → SearchCommandData`
Since	1.1
Description	Sets the comparison mask. The mask length MUST be ⇐ the search data length; missing bytes are right-padded with `FFh`.
Parameters	`mask` — the mask.
Returns	The current instance (`SearchCommandData`).
Throws	None.

Set Offset

Signature	`setOffset(offset: integer) → SearchCommandData`
Since	1.1
Description	Sets the offset (in bytes) at which the analysis starts within each record (default: 0).
Parameters	`offset` — the offset.
Returns	The current instance (`SearchCommandData`).
Throws	None.

Set Search Data

Signature	`setSearchData(data: byte array) → SearchCommandData`
Since	1.1
Description	Sets the byte sequence to search for within the targeted record.
Parameters	`data` — the data to search.
Returns	The current instance (`SearchCommandData`).
Throws	None.

Set SFI

Signature	`setSfi(sfi: byte) → SearchCommandData`
Since	1.1
Description	Sets the SFI of the EF in which the search is performed.
Parameters	`sfi` — the SFI of the EF.
Returns	The current instance (`SearchCommandData`).
Throws	None.

Start At Record

Signature	`startAtRecord(recordNumber: integer) → SearchCommandData`
Since	1.1
Description	Sets the record number at which the search begins (default: 1).
Parameters	`recordNumber` — the number of the record where the search should begin.
Returns	The current instance (`SearchCommandData`).
Throws	None.

5.2.11. Secure Extended Mode Transaction Manager

Name	`SecureExtendedModeTransactionManager`
Kind	Interface
Namespace	`calypso.card.transaction`
Extends	`SecureSymmetricCryptoTransactionManager<SecureExtendedModeTransactionManager>`
Since	2.0

Purpose	Symmetric-key transaction manager exposing additional operations only available on **Calypso Prime Extended** products. An instance is obtained via `CalypsoCardApiFactory.createSecureExtendedModeTransactionManager`.

Prepare Activate Encryption

Signature	`prepareActivateEncryption() → SecureExtendedModeTransactionManager`
Since	2.0
Description	Requests encryption for all following session commands. This ensures data confidentiality and prevents man-in-the-middle attacks. This command only makes sense in the context of a secure session. Encryption is resource intensive and increases transaction times: it is therefore RECOMMENDED to limit it to the commands that require it. Furthermore, when early mutual authentication is also required, it is RECOMMENDED for performance reasons to place the `prepareEarlyMutualAuthentication` and `prepareActivateEncryption` calls consecutively (in any order).
Returns	The current instance (`SecureExtendedModeTransactionManager`).
Throws	`UnsupportedOperationException` — if the **Manage Secure Session** command is not available for this context (the card and/or the cryptographic module does not support the extended mode).
See also	`SecureExtendedModeTransactionManager.prepareDeactivateEncryption` `SecureExtendedModeTransactionManager.prepareEarlyMutualAuthentication`

Prepare Deactivate Encryption

Signature	`prepareDeactivateEncryption() → SecureExtendedModeTransactionManager`
Since	2.0
Description	Requests to stop the encryption, which restores the exchanges with the card to their normal mode. This command only makes sense in the context of a secure session in which encryption of the commands has previously been requested. NOTE: `prepareCloseSecureSession()` automatically stops the encryption.
Returns	The current instance (`SecureExtendedModeTransactionManager`).
Throws	`UnsupportedOperationException` — if the **Manage Secure Session** command is not available for this context (the card and/or the cryptographic module does not support the extended mode).

See also	`SecureExtendedModeTransactionManager.prepareActivateEncryption`
	`SecureExtendedModeTransactionManager.prepareEarlyMutualAuthentication`
	`SecureTransactionManager.prepareCloseSecureSession`

Prepare Early Mutual Authentication

Signature	`prepareEarlyMutualAuthentication() → SecureExtendedModeTransactionManager`
Since	2.0
Description	Requests to mutually authenticate the card and the terminal **before** the secure session is closed. The use of this feature penalises the execution time of the secure session and SHOULD be used only when needed (e.g. before sending sensitive commands).
Returns	The current instance (`SecureExtendedModeTransactionManager`).
Throws	`UnsupportedOperationException` — if the **Manage Secure Session** command is not available for this context (the card and/or the cryptographic module does not support the extended mode).
See also	`SecureExtendedModeTransactionManager.prepareActivateEncryption` `SecureExtendedModeTransactionManager.prepareDeactivateEncryption`

5.2.12. Secure PKI Mode Transaction Manager

Name	`SecurePkiModeTransactionManager`
Kind	Interface
Namespace	`calypso.card.transaction`
Extends	`SecureTransactionManager<SecurePkiModeTransactionManager>`
Since	2.1
Purpose	Asymmetric-key transaction manager, compatible with Calypso cards in **PKI mode**. An instance is obtained via `CalypsoCardApiFactory.createSecurePkiModeTransactionManager`.

Prepare Open Secure Session

Signature	`prepareOpenSecureSession() → SecurePkiModeTransactionManager`
Since	2.1
Description	Schedules an **Open Secure Session** command in PKI mode. If the next prepared command is a **Read One Record** or **Read One Or More Counters**, it will be merged with the session opening for optimisation.
Returns	The current instance (`SecurePkiModeTransactionManager`).
Throws	None.

5.2.13. Secure Regular Mode Transaction Manager

Name	`SecureRegularModeTransactionManager`
Kind	Interface
Namespace	`calypso.card.transaction`
Extends	`SecureSymmetricCryptoTransactionManager<SecureRegularModeTransactionManager>`
Since	2.0
Purpose	Symmetric-key transaction manager compatible with every Calypso product. An instance is obtained via `CalypsoCardApiFactory.createSecureRegularModeTransactionManager`.

No additional operation declared.

5.2.14. Secure Session Status

Name	`SecureSessionStatus`
Kind	Interface
Namespace	`calypso.card.transaction`
Since	3.0
Purpose	Immutable snapshot of the current secure session state, returned by `TransactionManager.getSecureSessionStatus`.

A `SecureSessionStatus` is **captured** at the moment of the call and does **not** reflect subsequent changes to the session state. To obtain a refreshed status, the caller MUST call `getSecureSessionStatus()` again.

Get Type

Signature	`getType() → SecureSessionType`

Since	3.0
Description	Returns the cryptographic nature of the session (symmetric vs asymmetric).
Returns	A `SecureSessionType`.
Throws	None.

Get Write Access Level

Signature	`getWriteAccessLevel() → WriteAccessLevel?`
Since	3.0
Description	Returns the write access level requested at session opening. Returns `null` in PKI mode (`SecureSessionType.ASYMMETRIC`) where session opening does not take a `WriteAccessLevel`.
Returns	A `WriteAccessLevel`, or `null` in PKI mode.
Throws	None.

Is Open

Signature	`isOpen() → boolean`
Since	3.0
Description	Indicates whether a secure session is open at the snapshot moment.
Returns	A `boolean` value.
Throws	None.

5.2.15. Secure Symmetric Crypto Transaction Manager

Name	`SecureSymmetricCryptoTransactionManager`
Kind	Interface
Namespace	`calypso.card.transaction`
Generics	`<T extends SecureSymmetricCryptoTransactionManager<T>>`
Extends	`SecureTransactionManager<T>`
Since	2.0
Purpose	Common operations for every transaction manager backed by **symmetric** cryptography.

Prepare Change Key

Signature	`prepareChangeKey(`
	`  keyIndex: integer,`
	`  newKif: byte,`
	`  newKvc: byte,`
	`  issuerKif: byte,`
	`  issuerKvc: byte`
	`) → T`

Since	2.0
Description	Schedules a **Change Key** command. **MUST be performed outside a secure session.** All KIFs/KVCs MUST be available in the crypto module.

Parameters	`keyIndex` — the index of the key to be replaced (`1` for the issuer key, `2` for the load key, `3` for the debit key). `newKif` — the KIF of the new key. `newKvc` — the KVC of the new key. `issuerKif` — the KIF of the current card's issuer key. `issuerKvc` — the KVC of the current card's issuer key.
Returns	The current instance (`T`).
Throws	`UnsupportedOperationException` — if the **Change Key** command is not available for this card. `IllegalArgumentException` — if the provided key index is out of range. `IllegalStateException` — if the command is executed while a secure session is open.

Prepare Invalidate

Signature	`prepareInvalidate() → T`
Since	2.0
Description	Schedules an **Invalidate** command, which marks the card application as invalidated, blocking further use until it is rehabilitated. This command is usually executed within a secure session with the `DEBIT` key, depending on the access rights granted to it in the file structure of the card. The resulting DF status is available in the `CalypsoCard` via `isDfInvalidated`.
Returns	The current instance (`T`).
Throws	`IllegalStateException` — if the card is already invalidated. `SessionBufferOverflowException` — if the command would overflow the modifications buffer and the multiple-session mode is not allowed.

Prepare Open Secure Session

Signature	`prepareOpenSecureSession(writeAccessLevel: WriteAccessLevel) → T`
Since	2.0
Description	Schedules an **Open Secure Session** command. The secure session is opened with the supplied `WriteAccessLevel`, depending on whether the transaction profile is personalisation, reload or debit. If the next prepared command is a **Read One Record** or **Read One Or More Counters**, it is merged with the session opening for optimisation, unless the pre-open mode is active or the read-merge optimisation has been disabled. This mechanism may in some cases be incompatible with the security constraints; it can then be disabled via `SymmetricCryptoSecuritySetting.disableReadOnSessionOpening`.
Parameters	`writeAccessLevel` (`WriteAccessLevel`) — the write access level to be used.
Returns	The current instance (`T`).
Throws	`IllegalStateException` — in one of the following cases: no `SymmetricCryptoSecuritySetting` is available, a secure session opening is already prepared, or a secure session is already opened.
See also	`CalypsoCardSelectionExtension.preparePreOpenSecureSession`

Prepare Rehabilitate

Signature	`prepareRehabilitate() → T`
Since	2.0
Description	Schedules a **Rehabilitate** command, which clears the card application's invalidation status, making it usable again. This command is usually executed within a secure session with the `PERSONALIZATION` key, depending on the access rights granted to it in the file structure of the card. The resulting DF status is available in the `CalypsoCard` via `isDfInvalidated`.
Returns	The current instance (`T`).

Throws	`IllegalStateException` — if the card is not invalidated. `SessionBufferOverflowException` — if the command would overflow the modifications buffer and the multiple-session mode is not allowed.

Prepare SV Debit (Date)

Signature	`prepareSvDebit(`
	`  amount: integer,`
	`  date: byte array,`
	`  time: byte array`
	`) → T`

Since	2.0
Description	Schedules an **SV Debit** or **SV Undebit** command using the supplied additional data. It decreases the current SV balance by the supplied amount, or cancels a previous debit, according to the operation type chosen when the preceding **SV Get** command was called. Amount: `0..32767` for debit, `0..32768` for undebit. The key used is the **debit** key. Once processed, the data is available in the `CalypsoCard` through its dedicated SV data management operations.
Parameters	`amount` — the amount to be subtracted (`0..32767`) or added (`0..32768`). `date` — a 2-byte free value. `time` — a 2-byte free value.
Returns	The current instance (`T`).
Throws	`IllegalArgumentException` — if one of the provided arguments is out of range. `IllegalStateException` — in one of the following cases: the new value is negative and negative balances are not allowed, another SV command was already prepared inside the same secure session, the SV command is not placed in the first position in the list of prepared commands, the SV command does not follow an **SV Get** command, or the command and the SV operation are not consistent. `SessionBufferOverflowException` — if the command would overflow the modifications buffer and the multiple-session mode is not allowed.

Prepare SV Debit (No Date)

Signature	`prepareSvDebit(amount: integer) → T`
Since	2.0
Description	Schedules an **SV Debit** or **SV Undebit** command with the optional data fields set to zero. It decreases the current SV balance by the supplied amount, or cancels a previous debit. The key used is the **debit** key. Once processed, the data is available in the `CalypsoCard` through its dedicated SV data management operations.
Parameters	`amount` — the amount to be subtracted (`0..32767`) or added (`0..32768`).
Returns	The current instance (`T`).
Throws	`IllegalArgumentException` — if the provided argument is out of range. `IllegalStateException` — in one of the following cases: the new value is negative and negative balances are not allowed, another SV command was already prepared inside the same secure session, the SV command is not placed in the first position in the list of prepared commands, the SV command does not follow an **SV Get** command, or the command and the SV operation are not consistent. `SessionBufferOverflowException` — if the command would overflow the modifications buffer and the multiple-session mode is not allowed.

Prepare SV Get

Signature	`prepareSvGet(`
	`  svOperation: SvOperation,`
	`  svAction: SvAction`
	`) → T`

Since	2.0
Description	Schedules an **SV Get** command, which reads the Stored Value data (balance and logs) that MUST precede any SV debit, undebit or reload. Once processed, the data is available in the `CalypsoCard` through its dedicated SV data management operations.
Parameters	`svOperation` (`SvOperation`) — the nature of the intended operation: debit or reload. `svAction` (`SvAction`) — the type of action: `DO` (a debit or a positive reload), `UNDO` (an undebit or a negative reload).
Returns	The current instance (`T`).
Throws	`UnsupportedOperationException` — if the SV feature is not available for this card.

Prepare SV Reload (Date)

Signature	`prepareSvReload(`
	`  amount: integer,`
	`  date: byte array,`
	`  time: byte array,`
	`  free: byte array`
	`) → T`

Since	2.0
Description	Schedules an **SV Reload** command using the supplied additional data. Amount range: `-8388608..8388607`. The key used is the **reload** key. Once processed, the data is available in the `CalypsoCard` through its dedicated SV data management operations.
Parameters	`amount` — the value to be reloaded, positive or negative integer in the range `-8388608..8388607`. `date` — a 2-byte free value. `time` — a 2-byte free value. `free` — a 2-byte free value.
Returns	The current instance (`T`).
Throws	`IllegalArgumentException` — if one of the provided arguments is out of range. `IllegalStateException` — in one of the following cases: another SV command was already prepared inside the same secure session, the SV command is not placed in the first position in the list of prepared commands, the SV command does not follow an **SV Get** command, or the command and the SV operation are not consistent. `SessionBufferOverflowException` — if the command would overflow the modifications buffer and the multiple-session mode is not allowed.

Prepare SV Reload (No Date)

Signature	`prepareSvReload(amount: integer) → T`
Since	2.0
Description	Schedules an **SV Reload** command with the optional data fields set to zero. The key used is the **reload** key. Once processed, the data is available in the `CalypsoCard` through its dedicated SV data management operations.
Parameters	`amount` — the value to be reloaded, positive integer in the range `0..8388607` for a `DO` action, in the range `0..8388608` for an `UNDO` action.
Returns	The current instance (`T`).

Throws	`IllegalArgumentException` — if the provided argument is out of range. `IllegalStateException` — in one of the following cases: another SV command was already prepared inside the same secure session, the SV command is not placed in the first position in the list of prepared commands, the SV command does not follow an **SV Get** command, or the command and the SV operation are not consistent. `SessionBufferOverflowException` — if the command would overflow the modifications buffer and the multiple-session mode is not allowed.

5.2.16. Secure Transaction Manager

Name	`SecureTransactionManager`
Kind	Interface
Namespace	`calypso.card.transaction`
Generics	`<T extends SecureTransactionManager<T>>`
Extends	`TransactionManager<T>`
Since	2.0
Purpose	Common operations for every secure (cryptographically-backed) transaction manager.

Get Crypto Extension

Signature	`getCryptoExtension(`
	`  cryptoExtensionClass: Class<E>`
	`) → E extends CardTransactionCryptoExtension`

Since	2.0
Description	Returns the associated `CardTransactionCryptoExtension`.
Parameters	`cryptoExtensionClass` — the class of the crypto extension.
Returns	An `E extends CardTransactionCryptoExtension`.
Throws	None.

Prepare Cancel Secure Session

Signature	`prepareCancelSecureSession() → T`
Since	2.0
Description	Schedules a special **Close Secure Session** command aborting the current secure session. Executed in safe mode — never raises exceptions.
Returns	The current instance (`T`).
Throws	None.

Prepare Close Secure Session

Signature	`prepareCloseSecureSession() → T`
Since	2.0
Description	Schedules a **Close Secure Session** command. The ratification mechanism is disabled by default but MAY be enabled via `SymmetricCryptoSecuritySetting.enableRatificationMechanism`. In this case, a ratification command MUST be added after the **Close Secure Session** command when in contactless mode.
Returns	The current instance (`T`).

Throws	`IllegalStateException` — in one of the following cases: no secure session is opened and no secure session opening is prepared, a secure session closing is already prepared, or a secure session cancelling is prepared.

5.2.17. SV Debit Log Record

Name	`SvDebitLogRecord`
Kind	Interface
Namespace	`calypso.card.card`
Since	1.0
Purpose	One record of a Stored Value debit log.

Get Amount

Signature	`getAmount() → integer`
Since	1.0
Description	Returns the debit amount.
Returns	An `integer` value.
Throws	None.

Get Balance

Signature	`getBalance() → integer`
Since	1.0
Description	Returns the SV balance after the debit.
Returns	An `integer` value.
Throws	None.

Get Debit Date

Signature	`getDebitDate() → byte array`
Since	1.0
Description	Returns the 2-byte debit date.
Returns	A non-empty `byte array`.
Throws	None.

Get Debit Time

Signature	`getDebitTime() → byte array`
Since	1.0
Description	Returns the 2-byte debit time.
Returns	A non-empty `byte array`.
Throws	None.

Get KVC

Signature	`getKvc() → byte`
Since	1.0
Description	Returns the KVC of the debit key.
Returns	A `byte` value.
Throws	None.

Get Raw Data

Signature	`getRawData() → byte array`
Since	1.0
Description	Returns the raw bytes of the debit log record.
Returns	A non-empty `byte array`.
Throws	None.

Get SAM ID

Signature	`getSamId() → byte array`
Since	1.0
Description	Returns the 4-byte SAM ID.
Returns	A non-empty `byte array`.
Throws	None.

Get SAM T Num

Signature	`getSamTNum() → integer`
Since	1.0
Description	Returns the SAM transaction number.
Returns	An `integer` value.
Throws	None.

Get SV T Num

Signature	`getSvTNum() → integer`
Since	1.0
Description	Returns the SV transaction number.
Returns	An `integer` value.
Throws	None.

5.2.18. SV Load Log Record

Name	`SvLoadLogRecord`
Kind	Interface
Namespace	`calypso.card.card`

Since	1.0
Purpose	One record of a Stored Value load log.

Get Amount

Signature	`getAmount() → integer`
Since	1.0
Description	Returns the load amount.
Returns	An `integer` value.
Throws	None.

Get Balance

Signature	`getBalance() → integer`
Since	1.0
Description	Returns the SV balance after the load.
Returns	An `integer` value.
Throws	None.

Get Free Data

Signature	`getFreeData() → byte array`
Since	1.0
Description	Returns the 2-byte free data.
Returns	A non-empty `byte array`.
Throws	None.

Get KVC

Signature	`getKvc() → byte`
Since	1.0
Description	Returns the KVC of the load key.
Returns	A `byte` value.
Throws	None.

Get Load Date

Signature	`getLoadDate() → byte array`
Since	1.0
Description	Returns the 2-byte load date.
Returns	A non-empty `byte array`.
Throws	None.

Get Load Time

Signature	`getLoadTime() → byte array`
Since	1.0
Description	Returns the 2-byte load time.
Returns	A non-empty `byte array`.
Throws	None.

Get Raw Data

Signature	`getRawData() → byte array`
Since	1.0
Description	Returns the raw bytes of the load log record.
Returns	A non-empty `byte array`.
Throws	None.

Get SAM ID

Signature	`getSamId() → byte array`
Since	1.0
Description	Returns the 4-byte SAM ID.
Returns	A non-empty `byte array`.
Throws	None.

Get SAM T Num

Signature	`getSamTNum() → integer`
Since	1.0
Description	Returns the SAM transaction number.
Returns	An `integer` value.
Throws	None.

Get SV T Num

Signature	`getSvTNum() → integer`
Since	1.0
Description	Returns the SV transaction number.
Returns	An `integer` value.
Throws	None.

5.2.19. Symmetric Crypto Security Setting

Name	`SymmetricCryptoSecuritySetting`
Kind	Interface
Namespace	`calypso.card.transaction`

Since	2.0
Purpose	Security setting carrier for symmetric-key Calypso card transactions (e.g. SAM-backed). An instance is obtained via `CalypsoCardApiFactory.createSymmetricCryptoSecuritySetting`.

Add Authorized Session Key

Signature	`addAuthorizedSessionKey(`
	`  kif: byte,`
	`  kvc: byte`
	`) → SymmetricCryptoSecuritySetting`

Since	2.0
Description	Adds an authorised session key. Default: every key accepted; once at least one key is added, only listed keys are accepted.
Parameters	`kif` — the KIF value. `kvc` — the KVC value.
Returns	The current instance (`SymmetricCryptoSecuritySetting`).
Throws	None.

Add Authorized SV Key

Signature	`addAuthorizedSvKey(`
	`  kif: byte,`
	`  kvc: byte`
	`) → SymmetricCryptoSecuritySetting`

Since	2.0
Description	Adds an authorised Stored Value (SV) key, identified by its KIF/KVC. Default: every SV key accepted; once at least one key is added, only listed keys are accepted.
Parameters	`kif` — the KIF value. `kvc` — the KVC value.
Returns	The current instance (`SymmetricCryptoSecuritySetting`).
Throws	None.

Assign Default KIF

Signature	`assignDefaultKif(`
	`  writeAccessLevel: WriteAccessLevel,`
	`  kif: byte`
	`) → SymmetricCryptoSecuritySetting`

Since	2.0
Description	Defines the default KIF for a write access level.
Parameters	`writeAccessLevel` (`WriteAccessLevel`) — the write access level concerned. `kif` — the default KIF to use.
Returns	The current instance (`SymmetricCryptoSecuritySetting`).
Throws	None.

Assign Default KVC

Signature	`assignDefaultKvc(`
	`  writeAccessLevel: WriteAccessLevel,`
	`  kvc: byte`
	`) → SymmetricCryptoSecuritySetting`

Since	2.0
Description	Defines the default KVC for a write access level.
Parameters	`writeAccessLevel` (`WriteAccessLevel`) — the write access level concerned. `kvc` — the default KVC to use.
Returns	The current instance (`SymmetricCryptoSecuritySetting`).
Throws	None.

Assign KIF

Signature	`assignKif(`
	`  writeAccessLevel: WriteAccessLevel,`
	`  kvc: byte,`
	`  kif: byte`
	`) → SymmetricCryptoSecuritySetting`

Since	2.0
Description	Defines, for a write access level, the KIF to use when the card only provides a KVC.
Parameters	`writeAccessLevel` (`WriteAccessLevel`) — the write access level concerned. `kvc` — the KVC provided by the card. `kif` — the KIF to associate with that KVC.
Returns	The current instance (`SymmetricCryptoSecuritySetting`).
Throws	None.

Assign Open Secure Session Max Duration (All DF)

Signature	`assignOpenSecureSessionMaxDuration(`
	`  csnMin: long,`
	`  maxDuration: long`
	`) → SymmetricCryptoSecuritySetting`

Since	3.0
Description	Sets the maximum duration (in milliseconds) of an open secure session for cards whose CSN is >= `csnMin`, applied to every application DF (no `dfName` filter). See the `csnMin` combination rule.
Parameters	`csnMin` — the lowest card serial number (CSN) the setting applies to. `maxDuration` — the maximum duration of an open secure session, in milliseconds.
Returns	The current instance (`SymmetricCryptoSecuritySetting`).
Throws	None.

Assign Open Secure Session Max Duration (Per DF)

Signature	`assignOpenSecureSessionMaxDuration(`
	`  csnMin: long,`
	`  dfName: byte array,`
	`  maxDuration: long`
	`) → SymmetricCryptoSecuritySetting`

Since	3.0
Description	Sets the maximum duration (in milliseconds) of an open secure session for cards whose CSN is >= `csnMin` and whose application DF matches `dfName`. See the `csnMin` combination rule.

Parameters	`csnMin`—the lowest card serial number (CSN) the setting applies to. `dfName`—the application DF name the setting applies to. `maxDuration`—the maximum duration of an open secure session, in milliseconds.
Returns	The current instance (`SymmetricCryptoSecuritySetting`).
Throws	None.

Assign SV Operation Max Duration (All DF)

Signature	`assignSvOperationMaxDuration(`
	`  csnMin: long,`
	`  maxDuration: long`
	`) → SymmetricCryptoSecuritySetting`

Since	3.0
Description	Sets the maximum duration (in milliseconds) of a Stored Value operation for cards whose CSN is >= `csnMin`, applied to every application DF (no `dfName` filter).
Parameters	`csnMin`—the lowest card serial number (CSN) the setting applies to. `maxDuration`—the maximum duration of a Stored Value operation, in milliseconds.
Returns	The current instance (`SymmetricCryptoSecuritySetting`).
Throws	None.

Assign SV Operation Max Duration (Per DF)

Signature	`assignSvOperationMaxDuration(`
	`  csnMin: long,`
	`  dfName: byte array,`
	`  maxDuration: long`
	`) → SymmetricCryptoSecuritySetting`

Since	3.0
Description	Sets the maximum duration (in milliseconds) of a Stored Value operation for cards whose CSN is >= `csnMin` and whose application DF matches `dfName`.
Parameters	`csnMin`—the lowest card serial number (CSN) the setting applies to. `dfName`—the application DF name the setting applies to. `maxDuration`—the maximum duration of a Stored Value operation, in milliseconds.
Returns	The current instance (`SymmetricCryptoSecuritySetting`).
Throws	None.

Authorize SV Negative Balance

Signature	`authorizeSvNegativeBalance() → SymmetricCryptoSecuritySetting`
Since	2.0
Description	Allows the SV balance to become negative. Default: disabled.
Returns	The current instance (`SymmetricCryptoSecuritySetting`).
Throws	None.

Disable Read On Session Opening

Signature	`disableReadOnSessionOpening() → SymmetricCryptoSecuritySetting`
Since	2.0

Description	Disables the automatic merging of **Open Secure Session** with a possible **Read Record** command. By default, the optimisation is performed; it may however be incompatible with the security requirements in some cases.
Returns	The current instance (`SymmetricCryptoSecuritySetting`).
Throws	None.

Enable Multiple Session

Signature	`enableMultipleSession() → SymmetricCryptoSecuritySetting`
Since	2.0
Description	Enables multiple-session mode to handle more changes than the session buffer allows.
Returns	The current instance (`SymmetricCryptoSecuritySetting`).
Throws	None.

Enable PIN Plain Transmission

Signature	`enablePinPlainTransmission() → SymmetricCryptoSecuritySetting`
Since	2.0
Description	Enables the PIN transmission in plain text.
Returns	The current instance (`SymmetricCryptoSecuritySetting`).
Throws	None.

Enable Ratification Mechanism

Signature	`enableRatificationMechanism() → SymmetricCryptoSecuritySetting`
Since	2.0
Description	Enables the ratification mechanism to handle early card removal preventing reception of the closing ACK.
Returns	The current instance (`SymmetricCryptoSecuritySetting`).
Throws	None.

Enable SV Load And Debit Log

Signature	`enableSvLoadAndDebitLog() → SymmetricCryptoSecuritySetting`
Since	2.0
Description	Enables retrieval of both load **and** debit log records. Default: disabled.
Returns	The current instance (`SymmetricCryptoSecuritySetting`).
Throws	None.

Init Crypto Context For Next Transaction

Signature	`initCryptoContextForNextTransaction() → void`
Since	2.0

Description	Prepares the crypto module for the next transaction by anticipating the security context configuration. This feature is only useful when the currently allocated cryptographic module will be used for the next transaction. It is particularly relevant to optimise the transaction time in a ticketing context of user card validation. For the optimisation to be effective, this operation MUST be called at the very end of the current transaction, that is, **after** the user has been notified of the access right (e.g. after opening the gate).
Throws	`CryptoException` — if an error occurred when computing a crypto operation. `CryptoIOException` — if a communication error with the crypto module occurred (e.g. timeout with the reader or the computing unit, network error, etc.).

Set PIN Modification Ciphering Key

Signature	`setPinModificationCipheringKey(`
	`  kif: byte,`
	`  kvc: byte`
	`) → SymmetricCryptoSecuritySetting`

Since	2.0
Description	Sets the KIF/KVC of the PIN modification ciphering key. Default: 0/0.
Parameters	`kif` — the KIF value. `kvc` — the KVC value.
Returns	The current instance (`SymmetricCryptoSecuritySetting`).
Throws	None.

Set PIN Verification Ciphering Key

Signature	`setPinVerificationCipheringKey(`
	`  kif: byte,`
	`  kvc: byte`
	`) → SymmetricCryptoSecuritySetting`

Since	2.0
Description	Sets the KIF/KVC of the PIN verification ciphering key. Default: 0/0.
Parameters	`kif` — the KIF value. `kvc` — the KVC value.
Returns	The current instance (`SymmetricCryptoSecuritySetting`).
Throws	None.

`csnMin` combination rule. When several `assignOpenSecureSessionMaxDuration(…)` or `assignSvOperationMaxDuration(…)` calls are made with different `csnMin` values, each call defines a **range** bounded by its `csnMin` and the immediately higher declared `csnMin` (or `+∞` for the highest threshold). For a given card, the range to which its CSN belongs determines the applied bound. This mechanism allows the integrator to introduce or tighten duration bounds progressively as new generations of cards (with higher CSNs) are issued.

5.2.20. Transaction Manager

Name	`TransactionManager`
Kind	Interface
Namespace	`calypso.card.transaction`

Generics	`<T extends TransactionManager<T>>`
Extends	`IsoCardTransactionManager` (CNA-TR-API)
Since	2.0
Purpose	Common operations for every Calypso card transaction. Prepare/process model. The stereotype on `IsoCardTransactionManager` grants access to the multi-channel cast `asMultichannelCardTransactionManager()` defined by CNA-TR-API when the underlying ISO/IEC 7816-4 card supports multi-channel.

`TransactionManager` defines the full **prepare** command surface (file selection, reads, writes, counters, PIN, asymmetric key-pair generation). The preparation step makes it possible to group commands together in order to minimise network data exchanges, which is especially useful in a distributed architecture. The bound `CalypsoCard` is updated after each successful APDU exchange, and parameter ranges are validated as documented in Section 5.2.4.

Processing of prepared commands is inherited from `CardTransactionManager` (CNA-TR-API) via the `IsoCardTransactionManager` stereotype. The Calypso `TransactionManager` does **not** redefine `processCommands()`; it relies on the inherited contract. Multi-channel operations (`processCommandsAndCloseChannel`, `closeChannel`) are reached on demand through `asMultichannelCardTransactionManager()` of CNA-TR-API when the underlying card supports multi-channel.

Every write operation MAY raise `SessionBufferOverflowException` if the command would overflow the modification buffer and the multiple-session mode is not allowed.

Get Secure Session Status

Signature	`getSecureSessionStatus() → SecureSessionStatus`
Since	3.0
Description	Returns a `SecureSessionStatus` snapshot of the current secure session state. The returned object is immutable and captures the state at the moment of the call.
Returns	A `SecureSessionStatus`.
Throws	None.

Get Transaction Audit Data

Signature	`getTransactionAuditData() → list<byte array>`
Since	1.2
Description	Returns the audit data of the transaction (every APDU exchange with the card and the crypto module).
Returns	A `list<byte array>`; empty if no exchange has taken place.
Throws	None.

Prepare Append Record

Signature	`prepareAppendRecord(`
	`  sfi: byte,`
	`  recordData: byte array`
	`) → T`

Since	1.0

Description	Adds an **Append Record** command: a new record is added, and the oldest record is deleted in a cyclic file. Once processed, the data is available in the `CalypsoCard` through its dedicated file and data management operations.
Parameters	`sfi` — the SFI of the EF. `recordData` — the new record data to write.
Returns	The current instance (`T`).
Throws	`IllegalArgumentException` — if one of the supplied arguments is out of range.

Prepare Change PIN

Signature	`prepareChangePin(newPin: byte array) → T`
Since	1.6
Description	Adds a **Change PIN** command. It MUST be performed outside a secure session. If transmitted plain, the command MUST be preceded by a successful **Verify PIN**. Once processed, the PIN status is available in the `CalypsoCard` through `getPinAttemptRemaining` and `isPinBlocked`.
Parameters	`newPin` — the new PIN code value (4-byte long `byte array`).
Returns	The current instance (`T`).
Throws	`UnsupportedOperationException` — if the PIN feature is not available for this card. `IllegalArgumentException` — if the provided argument is out of range. `IllegalStateException` — if the command is executed while a secure session is open.

Prepare Check PIN Status

Signature	`prepareCheckPinStatus() → T`
Since	1.0
Description	Adds a **Verify PIN** command without PIN presentation, in order to read the attempt counter. Once processed, the PIN status is available in the `CalypsoCard` through `getPinAttemptRemaining` and `isPinBlocked`.
Returns	The current instance (`T`).
Throws	`UnsupportedOperationException` — if the PIN feature is not available for this card.

Prepare Decrease Counter

Signature	`prepareDecreaseCounter(`
	`  sfi: byte,`
	`  counterNumber: integer,`
	`  decValue: integer`
	`) → T`

Since	1.0
Description	Adds a **Decrease** command, which subtracts the supplied value from the designated counter of the selected EF. If several counters of the same file have to be decremented at the same time of the transaction, it is RECOMMENDED to use `prepareDecreaseCounters` for optimisation reasons. Once processed, the data is available in the `CalypsoCard` through its dedicated file and data management operations.
Parameters	`sfi` — the SFI of the EF. `counterNumber` — the number of the counter (MUST be zero in case of a simulated counter). `decValue` — the value to subtract from the counter (positive integer `⇐ 16777215`).

Returns	The current instance (`T`).
Throws	`IllegalArgumentException` — if one of the supplied arguments is out of range.

Prepare Decrease Counters

Signature	`prepareDecreaseCounters(`
	`  sfi: byte,`
	`  counterNumberToDecValueMap: map<integer, integer>`
	`) → T`

Since	1.1
Description	Adds a **Decrease Multiple** command, or multiple **Decrease** commands. The decision to execute one or the other is made according to the type of card. Once processed, the data is available in the `CalypsoCard` through its dedicated file and data management operations.
Parameters	`sfi` — the SFI of the EF. `counterNumberToDecValueMap` — the map containing the counter numbers to be decremented and their associated decrement values.
Returns	The current instance (`T`).
Throws	None.

Prepare Generate Asymmetric Key Pair

Signature	`prepareGenerateAsymmetricKeyPair() → T`
Since	2.1
Description	Adds a **Generate Asymmetric Key Pair** command. The public part can be retrieved via `prepareGetData(GetDataTag.CARD_PUBLIC_KEY)`.
Returns	The current instance (`T`).
Throws	None.

Prepare Get Data

Signature	`prepareGetData(tag: GetDataTag) → T`
Since	1.0
Description	Adds one or more **Get Data** commands. **Security warning:** this command MUST NOT be used within a secure session, in both contact and contactless modes. Once processed, the data is available in the `CalypsoCard` through `ElementaryFile.getHeader` or `getDirectoryHeader`, depending on the supplied tag.
Parameters	`tag` (`GetDataTag`) — the data type to retrieve.
Returns	The current instance (`T`).
Throws	`UnsupportedOperationException` — if the **Get Data** command with the provided tag is not supported. `IllegalStateException` — if a secure session is open.

Prepare Increase Counter

Signature	`prepareIncreaseCounter(`
	`  sfi: byte,`
	`  counterNumber: integer,`
	`  incValue: integer`
	`) → T`

Since	1.0
Description	Adds an **Increase** command. If several counters of the same file have to be incremented at the same time of the transaction, it is RECOMMENDED to use `prepareIncreaseCounters` for optimisation reasons. Once processed, the data is available in the `CalypsoCard` through its dedicated file and data management operations.
Parameters	`sfi` — the SFI of the EF. `counterNumber` — the number of the counter (MUST be zero in case of a simulated counter). `incValue` — the value to add to the counter (positive integer `⇐ 16777215`).
Returns	The current instance (`T`).
Throws	`IllegalArgumentException` — if one of the supplied arguments is out of range.

Prepare Increase Counters

Signature	`prepareIncreaseCounters(`
	`  sfi: byte,`
	`  counterNumberToIncValueMap: map<integer, integer>`
	`) → T`

Since	1.1
Description	Adds an **Increase Multiple** command, or multiple **Increase** commands. The decision to execute one or the other is made according to the type of card. Once processed, the data is available in the `CalypsoCard` through its dedicated file and data management operations.
Parameters	`sfi` — the SFI of the EF. `counterNumberToIncValueMap` — the map containing the counter numbers to be incremented and their associated increment values.
Returns	The current instance (`T`).
Throws	None.

Prepare Put Data

Signature	`preparePutData(`
	`  tag: PutDataTag,`
	`  data: byte array`
	`) → T`

Since	2.1
Description	Adds one or more **Put Data** commands. **MUST be called outside a secure session.**
Parameters	`tag` (`PutDataTag`) — the data type to inject. `data` — the data to inject.
Returns	The current instance (`T`).
Throws	`UnsupportedOperationException` — if the **Put Data** command with the provided tag is not supported. `IllegalArgumentException` — if `data` is empty. `IllegalStateException` — if a secure session is open.

Prepare Read Binary

Signature	`prepareReadBinary(`
	`  sfi: byte,`
	`  offset: integer,`
	`  nbBytesToRead: integer`
	`) → T`

Since	1.1
Description	Adds **Read Binary** commands. Depending on whether the command is placed inside a secure session, two behaviours apply. Outside a secure session (**best-effort** mode), the following `processCommands()` does not fail whatever the existence of the targeted file or the validity of the offset and the number of bytes to read—the `CalypsoCard` may then simply not be filled. Inside a secure session (**strict** mode), it fails in those cases. Once processed, the data is available in the `CalypsoCard` through its dedicated file and data management operations.
Parameters	`sfi` —the SFI of the EF. `offset` —the offset (`0` indicates the first byte). `nbBytesToRead`—the number of bytes to read.
Returns	The current instance (`T`).
Throws	`UnsupportedOperationException`—if this command is not supported by this card. `IllegalArgumentException`—if one of the supplied arguments is out of range.

Prepare Read Counter

Signature	`prepareReadCounter(`
	`  sfi: byte,`
	`  nbCountersToRead: integer`
	`) → T`

Since	1.1
Description	Adds a **Read Records** command targeting the supplied counter. The record is read up to the counter location supplied as parameter, so all preceding counters are also read. Depending on whether the command is placed inside a secure session, two behaviours apply. Outside a secure session (**best-effort** mode), the following `processCommands()` does not fail whatever the existence of the targeted file or counter—the `CalypsoCard` may then simply not be filled. Inside a secure session (**strict** mode), it fails if the targeted file or counter is not found. Once processed, the data is available in the `CalypsoCard` through its dedicated file and data management operations.
Parameters	`sfi` —the SFI of the EF. `nbCountersToRead`—the number of counters to read.
Returns	The current instance (`T`).
Throws	`IllegalArgumentException`—if one of the supplied arguments is out of range.

Prepare Read Record

Signature	`prepareReadRecord(`
	`  sfi: byte,`
	`  recordNumber: integer`
	`) → T`

Since	1.1

Description	Adds a **Read Records** command for a single record. **Security warning:** this command MUST NOT be used within a secure session, in both contact and contactless modes. For the in-session case, use `prepareReadRecords` instead, with valid parameters. Once processed, the data is available in the `CalypsoCard` through its dedicated file and data management operations. The following `processCommands()` does not fail whatever the existence of the targeted file or record: the `CalypsoCard` may then simply not be filled (**best-effort** mode).
Parameters	`sfi` — the SFI of the EF. `recordNumber` — the record to read.
Returns	The current instance (`T`).
Throws	`IllegalArgumentException` — if one of the supplied arguments is out of range.

Prepare Read Records

Signature	`prepareReadRecords(`
	`  sfi: byte,`
	`  fromRecordNumber: integer,`
	`  toRecordNumber: integer,`
	`  recordSize: integer`
	`) → T`

Since	1.1
Description	Adds a **Read Records** command for one or more records. Depending on whether the command is placed inside a secure session, two behaviours apply. Outside a secure session (**best-effort** mode), the following `processCommands()` does not fail whatever the existence of the targeted file or record — the `CalypsoCard` may then simply not be filled. Inside a secure session (**strict** mode), it fails if the targeted file or record is not found. Once processed, the data is available in the `CalypsoCard` through its dedicated file and data management operations.
Parameters	`sfi` — the SFI of the EF. `fromRecordNumber` — the number of the first record to read. `toRecordNumber` — the number of the last record to read. `recordSize` — the record length.
Returns	The current instance (`T`).
Throws	`IllegalArgumentException` — if one of the supplied arguments is out of range.

Prepare Read Records Partially

Signature	`prepareReadRecordsPartially(`
	`  sfi: byte,`
	`  fromRecordNumber: integer,`
	`  toRecordNumber: integer,`
	`  offset: integer,`
	`  nbBytesToRead: integer`
	`) → T`

Since	1.1

Description	Adds **Read Record Multiple** commands. **Security warning:** this command MUST NOT be used within a secure session, in both contact and contactless modes. The following `processCommands()` does not fail whatever the existence of the targeted file or the validity of the offset and the number of bytes to read: the `CalypsoCard` may then simply not be filled (**best-effort** mode). Once processed, the data is available in the `CalypsoCard` through its dedicated file and data management operations.
Parameters	`sfi` — the SFI of the EF. `fromRecordNumber` — the number of the first record to read. `toRecordNumber` — the number of the last record to read. `offset` — the offset in the records where to start reading (`0` indicates the first byte). `nbBytesToRead` — the number of bytes to read from each record.
Returns	The current instance (`T`).
Throws	`UnsupportedOperationException` — if this command is not supported by this card. `IllegalArgumentException` — if one of the provided arguments is out of range. `IllegalStateException` — if a secure session is open.

Prepare Search Records

Signature	`prepareSearchRecords(data: SearchCommandData) → T`
Since	1.1
Description	Adds a **Search Record Multiple** command. The command searches whether the supplied data are present in the records of the file; an optional mask MAY be applied to specify precisely the bits to take into account in the comparison. It is only possible with a **linear**, **cyclic**, **Counters** or **Simulated Counter** EF. **Security warning:** this command MUST NOT be used within a secure session, in both contact and contactless modes. Once processed, the result is available in the supplied input/output `SearchCommandData`, and the content of the first matching record in the `CalypsoCard` when requested. The following `processCommands()` does not fail whatever the existence of the targeted file or the validity of the record number and offset: the `SearchCommandData` and `CalypsoCard` may then simply not be updated (**best-effort** mode).
Parameters	`data` (`SearchCommandData`) — the input/output data containing the parameters of the command.
Returns	The current instance (`T`).
Throws	`UnsupportedOperationException` — if the **Search Record Multiple** command is not available for this card. `IllegalArgumentException` — if the input data is inconsistent. `IllegalStateException` — if a secure session is open.
See also	`SearchCommandData`

Prepare Select File (LID)

Signature	`prepareSelectFile(lid: short) → T`
Since	1.1
Description	Adds a **Select File** command targeting an EF by LID. **Caution:** the command fails if the selected file is not an EF. Once processed, the data is available in the `CalypsoCard` through `getFileBySfi` / `getFileByLid` and `ElementaryFile.getHeader`.
Parameters	`lid` — the LID of the EF to select.

Returns	The current instance (`T`).
Throws	None.

Prepare Select File (Select Control)

Signature	`prepareSelectFile(selectFileControl: SelectFileControl) → T`
Since	1.0
Description	Adds a **Select File** command using a navigation control. Once processed, the data is available in the `CalypsoCard` through `ElementaryFile.getHeader`.
Parameters	`selectFileControl` (`SelectFileControl`) — a `SelectFileControl` enum entry.
Returns	The current instance (`T`).
Throws	None.

Prepare Set Counter

Signature	`prepareSetCounter(`
	`  sfi: byte,`
	`  counterNumber: integer,`
	`  newValue: integer`
	`) → T`

Since	1.0
Description	Adds either an **Increase** or a **Decrease** command to set the counter to the supplied value. The operation is selected according to whether the difference between the current value and the desired value is negative (**Increase**) or positive (**Decrease**). Two assumptions are made, and neither is checked by this operation: the counter value has been read beforehand, and the type of session — and its associated access rights — is consistent with the requested operation (a reload session if the counter is to be incremented, a debit session if it is to be decremented). An inconsistency is only detected when the session is closed. Once processed, the data is available in the `CalypsoCard` through its dedicated file and data management operations.
Parameters	`sfi` — the SFI of the EF. `counterNumber` — the counter number (`>= 1` for a counters file, `0` for a simulated "counter" file). `newValue` — the desired value for the counter (positive integer `<= 16777215`).
Returns	The current instance (`T`).
Throws	`IllegalStateException` — if the current counter value is unknown. `IllegalArgumentException` — if one of the supplied arguments is out of range.

Prepare SV Read All Logs

Signature	`prepareSvReadAllLogs() → T`
Since	1.0

Description	Schedules the reading of every Stored Value log, that is, both the load and the debit log records. This operation requires the selected application to be of type **Store Value** (file structure `20h`). The SV transaction logs are contained in two files with fixed identifiers: the file whose SFI is `14h` holds one record carrying the unique reload log, and the file whose SFI is `15h` holds three records carrying the last three debit logs. Once processed, the data is available in the `CalypsoCard` in raw format through its dedicated file and data management operations, or as dedicated objects through `getSvLoadLogRecord` and `getSvDebitLogAllRecords`.
Returns	The current instance (`T`).
Throws	`UnsupportedOperationException` — if the SV feature is not available for this card.

Prepare Update Binary

Signature	`prepareUpdateBinary(`
	`  sfi: byte,`
	`  offset: integer,`
	`  data: byte array`
	`) → T`

Since	1.1
Description	Adds one or more **Update Binary** commands. The data of the file before the offset and after the supplied data are left unchanged. Once processed, the data is available in the `CalypsoCard` through its dedicated file and data management operations.
Parameters	`sfi` — the SFI of the EF. `offset` — the offset (`0` indicates the first byte). `data` — the new data.
Returns	The current instance (`T`).
Throws	`UnsupportedOperationException` — if this command is not supported by this card. `IllegalArgumentException` — if one of the supplied arguments is out of range.

Prepare Update Record

Signature	`prepareUpdateRecord(`
	`  sfi: byte,`
	`  recordNumber: integer,`
	`  recordData: byte array`
	`) → T`

Since	1.0
Description	Adds an **Update Record** command. Bytes beyond `recordData.length` are left unchanged. Once processed, the data is available in the `CalypsoCard` through its dedicated file and data management operations.
Parameters	`sfi` — the SFI of the EF. `recordNumber` — the record to update. `recordData` — the new record data. If length is less than the record size, bytes beyond length are left unchanged.
Returns	The current instance (`T`).
Throws	`IllegalArgumentException` — if one of the supplied arguments is out of range.

Prepare Verify PIN

Signature	`prepareVerifyPin(pin: byte array) → T`
Since	1.6
Description	Adds a **Verify PIN** command. This command MAY be performed both inside and outside a secure session. The PIN code is transmitted in plain text or enciphered, according to the parameter set in the `SymmetricCryptoSecuritySetting`. Once processed, the PIN status is available in the `CalypsoCard` through `getPinAttemptRemaining` and `isPinBlocked`.
Parameters	`pin` — the PIN code value (4-byte long `byte array`).
Returns	The current instance (`T`).
Throws	`UnsupportedOperationException` — if the PIN feature is not available for this card. `IllegalArgumentException` — if the provided argument is out of range.

Prepare Write Binary

Signature	`prepareWriteBinary(`
	`  sfi: byte,`
	`  offset: integer,`
	`  data: byte array`
	`) → T`

Since	1.1
Description	Adds one or more **Write Binary** commands, performing a binary OR with the existing data. The data of the file before the offset and after the supplied data are left unchanged. Once processed, the data is available in the `CalypsoCard` through its dedicated file and data management operations.
Parameters	`sfi` — the SFI of the EF. `offset` — the offset (`0` indicates the first byte). `data` — the data to write over the existing data.
Returns	The current instance (`T`).
Throws	`UnsupportedOperationException` — if this command is not supported by this card. `IllegalArgumentException` — if one of the supplied arguments is out of range.

Prepare Write Record

Signature	`prepareWriteRecord(`
	`  sfi: byte,`
	`  recordNumber: integer,`
	`  recordData: byte array`
	`) → T`

Since	1.0
Description	Adds a **Write Record** command, performing a binary OR with the existing data. Once processed, the data is available in the `CalypsoCard` through its dedicated file and data management operations.
Parameters	`sfi` — the SFI of the EF. `recordNumber` — the record to write. `recordData` — the data to overwrite in the record. If length is less than the record size, bytes beyond length are left unchanged.
Returns	The current instance (`T`).

Throws	`IllegalArgumentException` — if one of the supplied arguments is out of range.

5.3. SPI Interfaces

5.3.1. Asymmetric Crypto Card Transaction Manager Factory

Name	`AsymmetricCryptoCardTransactionManagerFactory`
Kind	Marker interface (SPI)
Namespace	`calypso.card.transaction.spi`
Since	2.0
Purpose	Marker provided by crypto extensions to secure Calypso card transactions with **asymmetric** keys (PKI). Declares no operation.

5.3.2. Card Transaction Crypto Extension

Name	`CardTransactionCryptoExtension`
Kind	Marker interface (SPI)
Namespace	`calypso.card.transaction.spi`
Since	2.0
Purpose	Marker enriching the card transaction command set with specific crypto commands (e.g. signature computation/verification). Declares no operation at the API level.

5.3.3. PCA Certificate

Name	`PcaCertificate`
Kind	Marker interfaces (SPI)
Namespace	`calypso.card.transaction.spi`
Since	2.1
Purpose	Markers for PCA, CA and card certificates exposed by CNA-TCCA-API extensions.

`PcaCertificate`, `CaCertificate` and `CardCertificate` declare no operation at the API level. Their concrete content is exposed through the CNA-TCCA-API extension that produced them.

5.3.4. PCA Certificate Parser

Name	`PcaCertificateParser`
Kind	Marker interfaces (SPI)
Namespace	`calypso.card.transaction.spi`
Since	2.1
Purpose	Markers for CA and card certificate parsers exposed by CNA-TCCA-API extensions. Used by `AsymmetricCryptoSecuritySetting.addCaCertificateParser` / `addCardCertificateParser`.

5.3.5. Symmetric Crypto Card Transaction Manager Factory

Name	`SymmetricCryptoCardTransactionManagerFactory`

Kind	Marker interface (SPI)
Namespace	`calypso.card.transaction.spi`
Since	2.0
Purpose	Marker provided by crypto extensions to secure Calypso card transactions with **symmetric** keys (e.g. SAM). Declares no operation.

5.4. Enumerations

5.4.1. Calypso Card Product Type

Name	`ProductType`
Kind	Enumeration
Namespace	`calypso.card.card.CalypsoCard`
Since	1.0
Purpose	Product type of a Calypso card.

Value	Since	Description
`PRIME_REVISION_1`	1.0	Calypso Prime revision 1.x.
`PRIME_REVISION_2`	1.0	Calypso Prime revision 2.x.
`PRIME_REVISION_3`	1.0	Calypso Prime revision 3.x.
`LIGHT`	1.0	Calypso Light.
`BASIC`	1.0	Calypso Basic.
`UNKNOWN`	1.0	Application Type is `0` or `FFh`, or selection data could not be properly parsed.

5.4.2. Elementary File Type

Name	`Type`
Kind	Enumeration
Namespace	`calypso.card.card.ElementaryFile`
Since	1.0
Purpose	Type of a Calypso Elementary File.

Value	Since	Description
`LINEAR`	1.0	Linear EF (1 to several records).
`BINARY`	1.0	Binary EF (single continuous sequence of data bytes).
`CYCLIC`	1.0	Cyclic EF (records organised in a cycle, from the most recent to the oldest).
`COUNTERS`	1.0	Counters EF (single record containing K counters of three bytes each).
`SIMULATED_COUNTERS`	1.0	Simulated counter EF (linear file with a single record, kept for Calypso Rev 2 compatibility).

5.4.3. Get Data Tag

Name	`GetDataTag`
Kind	Enumeration
Namespace	`calypso.card`
Since	1.0
Purpose	Output data tags retrievable through the **Get Data** command. MAY NOT be applicable to all products.

Value	Since	Description
`FCP_FOR_CURRENT_FILE`	1.0	FCP for the current file, as returned by **Select File**.
`FCI_FOR_CURRENT_DF`	1.0	FCI for the current DF, as returned by **Select Application**.
`EF_LIST`	1.1	List of EFs in the current DF.
`TRACEABILITY_INFORMATION`	1.1	Product traceability information.
`CARD_PUBLIC_KEY`	2.1	Card public key.
`CARD_CERTIFICATE`	2.1	Card certificate.
`CA_CERTIFICATE`	2.1	CA certificate.

5.4.4. Put Data Tag

Name	`PutDataTag`
Kind	Enumeration
Namespace	`calypso.card`
Since	2.1
Purpose	Input data tags injectable through the **Put Data** command. MAY NOT be applicable to all products.

Value	Since	Description
`CARD_KEY_PAIR`	2.1	Card key pair.
`CARD_CERTIFICATE`	2.1	Card certificate.
`CA_CERTIFICATE`	2.1	CA certificate.

5.4.5. Secure Session Type

Name	`SecureSessionType`
Kind	Enumeration
Namespace	`calypso.card.transaction`
Since	3.0
Purpose	Cryptographic nature of a Calypso secure session.

Value	Since	Description
`SYMMETRIC`	3.0	PSO/SAM sessions, Regular and Extended modes.
`ASYMMETRIC`	3.0	PKI mode.

This enumeration reflects the **cryptographic nature** of the session and not the **application mode** (Regular / Extended). Both symmetric modes share the same underlying

cryptography and only differ in their configuration at the `TransactionManager` level; the integrator who needs to know the application mode obtains it directly via the sub-type of `TransactionManager` that has been instantiated.

5.4.6. Select File Control

Name	`SelectFileControl`
Kind	Enumeration
Namespace	`calypso.card`
Since	1.0
Purpose	Expected behaviour of the **Select File** command (cf. ISO/IEC 7816-4 and Calypso specifications).

Value	Since	Description
`FIRST_EF`	1.0	The first EF of the current Calypso DF.
`NEXT_EF`	1.0	The next EF of the current Calypso DF.
`CURRENT_DF`	1.0	The current Calypso DF.

5.4.7. SV Action

Name	`SvAction`
Kind	Enumeration
Namespace	`calypso.card.transaction`
Since	1.0
Purpose	Direction of an SV action.

Value	Since	Description
`DO`	1.0	In a `RELOAD`, load a positive amount; in a `DEBIT`, debit a positive amount.
`UNDO`	1.0	In a `RELOAD`, load a negative amount; in a `DEBIT`, cancel (totally or partially) a previous debit.

5.4.8. SV Operation

Name	`SvOperation`
Kind	Enumeration
Namespace	`calypso.card.transaction`
Since	1.0
Purpose	Type of Stored Value operation.

Value	Since	Description
`RELOAD`	1.0	Increase the SV balance.
`DEBIT`	1.0	Decrease the SV balance.

5.4.9. Write Access Level

Name	`WriteAccessLevel`
Kind	Enumeration
Namespace	`calypso.card`
Since	1.0
Purpose	Write access level for a Calypso secure session. Each level induces the use of a different session-key role.

Value	Since	Description
`PERSONALIZATION`	1.0	Personalisation, load and debit operations — uses the **issuer** key.
`LOAD`	1.0	Load and debit operations — uses the **load** key.
`DEBIT`	1.0	Debit operations only — uses the **debit** key.

5.5. Exceptions

5.5.1. Card Signature Not Verifiable Exception

Name	`CardSignatureNotVerifiableException`
Kind	Runtime exception
Namespace	`calypso.card.transaction`
Since	1.2
Purpose	Indicates that the card has correctly closed the secure session but that the authenticity of the session cannot be verified because the crypto module is no longer available (timeout, network problem, etc.).

5.5.2. Crypto Exception

Name	`CryptoException`
Kind	Runtime exception
Namespace	`calypso.card.transaction`
Since	2.0
Purpose	Indicates that an error occurred when computing a crypto operation.

5.5.3. Crypto IO Exception

Name	`CryptoIOException`
Kind	Runtime exception
Namespace	`calypso.card.transaction`
Since	2.0
Purpose	Indicates a communication error with the crypto module.

5.5.4. Inconsistent Data Exception

Name	`InconsistentDataException`
Kind	Runtime exception

Namespace	`calypso.card.transaction`
Since	1.2
Purpose	Indicates the detection of inconsistent data.

The inconsistency falls into one of the following cases:

- a **de-synchronisation of the APDU exchanges**, that is, a number of APDU responses different from the number of APDU requests;
- an **inconsistency in the card data**, which can happen for example when the data read outside the secure session differs from the data read inside it.

5.5.5. Invalid Card Signature Exception

Name	`InvalidCardSignatureException`
Kind	Runtime exception
Namespace	`calypso.card.transaction`
Since	1.2
Purpose	Indicates that the card signature is incorrect. For a transaction secured by **symmetric** cryptography (e.g. SAM), this indicates that the card has correctly closed the secure session, but that the card session is **not authentic** because the MAC it produced is wrong. This typically happens when the **Digest Authenticate** or **SV Check** status word is `6988h`. For a transaction secured by **asymmetric** cryptography (e.g. PKI), this indicates only that the card signature is incorrect.

5.5.6. Invalid Certificate Exception

Name	`InvalidCertificateException`
Kind	Runtime exception
Namespace	`calypso.card.transaction`
Since	2.1
Purpose	Indicates that a certificate failed validation (signature, validity period, issuer or subject details, constraints, extensions, expiration, revocation).

5.5.7. Invalid PIN Exception

Name	`InvalidPinException`
Kind	Runtime exception
Namespace	`calypso.card.transaction`
Since	2.0
Purpose	Indicates that the provided PIN is invalid.

5.5.8. Select File Exception

Name	`SelectFileException`
Kind	Runtime exception

Namespace	`calypso.card.transaction`
Since	1.4
Purpose	Indicates that file selection failed because the file was not found (status word `6A82h`).

5.5.9. Session Buffer Overflow Exception

Name	`SessionBufferOverflowException`
Kind	Runtime exception
Namespace	`calypso.card.transaction`
Since	1.2
Purpose	Indicates that the secure session cannot be performed atomically because the modification buffer capacity is insufficient.

5.5.10. Unauthorized Key Exception

Name	`UnauthorizedKeyException`
Kind	Runtime exception
Namespace	`calypso.card.transaction`
Since	1.0
Purpose	Indicates that the card requires an unauthorised session key.