

# *mhw.card* PACKAGE

## **INTRODUCTION**

The package `mhw.card` provides objects representing various types of cards supported by the digital set-top box. Implementing these cards hides the card protocol. Each card class provides high-level methods for using it, and reading and writing information carried by it. Cards may be credit cards, conditional access cards and so on.

The package `mhw.card` manages notification of the insertion and extraction of a card in any available card slot. Applications or other packages can register themselves to the card manager object to receive events related to card insertion or extraction. Each time a card is inserted into a slot, the card manager creates an instance of a subclass of the class `card`. The subclass is dependent on the card type. When a card is extracted from a slot, the corresponding card object is set so that it can no longer be used by an application.

© 1998 - 19 9 CANAL+. This document is the intellectual property of CANAL+ and contains confidential information. All reproduction and communication to third parties is strictly prohibited without prior written consent from CANAL+.

## PACKAGE CONTENTS

NAME	DESCRIPTION
CLASSES	
<code>Card</code>	Defines an abstract card.
<code>CardEvent</code>	Defines an abstract event related to the card state.
<code>CardInsertedEvent</code>	Defines an event sent to indicate that a card has been inserted.
<code>CardManager</code>	Manages cards.
<code>CardRemovedEvent</code>	Defines an event sent to indicate that a card has been extracted.
<code>SECACard</code>	Defines a conditional access event card from the <b>SOCIETE EUROPEENE DE CONTROLE D'ACCESS</b> .
EXCEPTIONS	
<code>CardStatusException</code>	Indicates that a card refused a command.
<code>CardUnavailableException</code>	Indicates that the card is no longer available for use.



The classes and exceptions contained in the package *mhw.card* are presented below in general alphabetical order.

# Card

```
public class Card extends Object
```

## DESCRIPTION


**Card** describes an abstract card.


## HIERARCHY


```
java.lang.Object  
↳ mhw.card.Card
```

## ATTRIBUTES

The class **Card** contains the following attributes:

Attribute	CA0
<b>Description</b>	Describes a constant value for a type of module that supports a common interface.  The values 0x02 through 0xFE, inclusive, are reserved.
<b>Syntax</b>	<code>public static final int CA0</code>

Attribute	CA1
<b>Description</b>	Describes a constant value for a type of smart card module.  The values 0x02 through 0xFE, inclusive, are reserved.
<b>Syntax</b>	<code>public static final int CA1</code>

Attribute	PROPRIETARY
<b>Description</b>	Describes a constant value for a type of proprietary module.  The values 0x02 through 0xFE, inclusive, are reserved.
<b>Syntax</b>	<code>public static final int PROPRIETARY</code>

**METHODS**

The class `Card` contains the following methods:

<b>Method</b>	<code>getCardType()</code>
<b>Description</b>	Gets the card type.
<b>Syntax</b>	<code>public int getCardType()</code>
<b>Returns</b>	The card type.

<b>Method</b>	<code>getHistory(Buffer)</code>
<b>Description</b>	Gets the card history without resetting it.
<b>Syntax</b>	<code>public void reset(Buffer aHistory) throws CardStatusException, CardUnavailableException</code>
<b>Parameters</b>	<code>aHistory</code> The card history buffer, which is updated by the method.
<b>Throws</b>	<code>CardStatusException</code> If the method fails. <code>CardUnavailableException</code> If the card is no longer available.

<b>Method</b>	<code>getHistory(int)</code>
<b>Description</b>	Gets the card history without resetting it.
<b>Syntax</b>	<code>public void reset(int aHistorySize) throws ManagerException, CardStatusException</code>
<b>Parameters</b>	<code>aHistorySize</code> The card history size.
<b>Returns</b>	<code>byte[]</code> The byte containing the card history.
<b>Throws</b>	<code>ManagerException</code> If the buffer for the card history cannot be allocated. <code>CardStatusException</code> If the method fails.

Method	<code>getSlotNumber()</code>
<b>Description</b>	Gets the card connection slot number.
<b>Syntax</b>	<code>public synchronized int getSlotNumber () throws CardUnavailableException</code>
<b>Returns</b>	The card connection slot number.

Method	<code>getState()</code>
<b>Description</b>	Gets the card state.
<b>Syntax</b>	<code>public byte getState() throws CardStatusException</code>
<b>Returns</b>	The byte containing the card state.
<b>Throws</b>	<code>CardStatusException</code> If the method fails.

Method	<code>reset()</code>
<b>Description</b>	Resets the card and reads its history.
<b>Syntax</b>	<code>public void reset(Buffer aHistory) throws CardStatusException</code>
<b>Parameters</b>	<code>aHistory</code> The card history, which is updated by the method.
<b>Returns</b>	<code>void</code>
<b>Throws</b>	<code>CardStatusException</code> If the method fails.

Method	<code>sendInputMessage(Buffer,Buffer)</code>	
Description	Sends an input message to the card, and returns the card state.	
Syntax	<code>public void sendInputMessage(Buffer aMessageBuffer, Buffer aReplyBuffer) throws CardStatusException</code>	
Parameters	<code>aMessageBuffer</code>	The input message.
	<code>aReplyBuffer</code>	The card status.
Throws	<code>CardStatusException</code>	If the method fails.

Method	<code>sendInputMessage(int,int,Buffer,Buffer)</code>	
Description	Sends an asynchronous input message to the card, and returns the card state.	
Syntax	<code>public void sendInputMessage(int anEventType, int anEventPriority, Buffer aMessageBuffer, Buffer aReplyBuffer) throws CardStatusException</code>	
Parameters	<code>anEventType</code>	The type of event sent to indicate the end of the command.
	<code>anEventPriority</code>	The event priority.
	<code>aMessageBuffer</code>	The input message.
	<code>aReplyBuffer</code>	The card status.
Throws	<code>CardStatusException</code>	If the method fails.

Method	<code>sendInputOutputMessage(Buffer, Buffer)</code>	
<b>Description</b>	Sends two consecutive messages to the card.	
<b>Syntax</b>	<code>public void sendInputOutputMessage(Buffer aMessageBuffer, Buffer aReplyBuffer) throws CardStatusException</code>	
<b>Parameters</b>	<code>aMessageBuffer</code>	The input and output message.
	<code>aReplyBuffer</code>	The reply and the card status.
<b>Throws</b>	<code>CardStatusException</code>	If the method fails.

Method	<code>sendInputOutputMessage(int, int, Buffer, Buffer)</code>	
<b>Description</b>	Sends two consecutive messages to the card.	
<b>Syntax</b>	<code>public void sendInputMessage(int anEventType, int anEventPriority, Buffer aMessageBuffer, Buffer aReplyBuffer) throws CardStatusException</code>	
<b>Parameters</b>	<code>anEventType</code>	The type of event sent to indicate the end of the command.
	<code>anEventPriority</code>	The event priority.
	<code>aMessageBuffer</code>	The input message.
	<code>aReplyBuffer</code>	The card status.
<b>Throws</b>	<code>CardStatusException</code>	If the method fails.

Method	<code>sendOutputMessage(Buffer,Buffer)</code>	
Description	Sends an output message to the card.	
Syntax	<code>public void sendOutputMessage(Buffer aMessageBuffer, Buffer aReplyBuffer) throws CardStatusException</code>	
Parameters	<code>aMessageBuffer</code>	The output message.
	<code>aReplyBuffer</code>	The card status.
Throws	<code>CardStatusException</code>	If the method fails.

Method	<code>sendOutputMessage(int,int,Buffer,Buffer)</code>	
Description	Sends an asynchronous output message to the card, and returns the card state.	
Syntax	<code>public void sendOutputMessage(int anEventType, int anEventPriority, Buffer aMessageBuffer, Buffer aReplyBuffer) throws CardStatusException</code>	
Parameters	<code>anEventType</code>	The type of event sent to indicate the end of the command.
	<code>anEventPriority</code>	The event priority.
	<code>aMessageBuffer</code>	The input message.
	<code>aReplyBuffer</code>	The card status.
Throws	<code>CardStatusException</code>	If the method fails.



# CardEvent

public abstract class **CardEvent** extends ResourceStatusEvent

## DESCRIPTION

**CardEvent** is an abstract card event that signals a change in the card state.

## HIERARCHY

```
java.lang.Object
↳ java.util.EventObject
    ↳ org.davic.resources.ResourceStatusEvent
        ↳ mhw.card.CardEvent
```

## CONSTRUCTORS

The class **CardEvent** contains the following constructors:

Constructor	<b>CardEvent ( Card , Object )</b>	
<b>Description</b>	Creates an instance of <b>CardEvent</b> .	
<b>Syntax</b>	<b>public CardEvent( Card card , Object source )</b>	
<b>Parameters</b>	<b>card</b>	The card.
	<b>source</b>	The object that created the event.

## METHODS

The class **CardEvent** contains the following methods:

Method	<b>getCard ( )</b>
<b>Description</b>	Gets the card related to the event.
<b>Syntax</b>	<b>public Card getCard ( )</b>
<b>Returns</b>	The card related to the event.

# CardInsertedEvent

```
public class CardInsertedEvent extends CardEvent
```

## DESCRIPTION

CardInsertedEvent is an event that indicates that a card has be inserted in a slot.

## HIERARCHY

```
java.lang.Object
↳ java.util.EventObject
  ↳ org.davic.resources.ResourceStatusEvent
    ↳ mhw.card.CardEvent
      ↳ mhw.card.CardInsertedEvent
```

## CONSTRUCTORS

The class CardInsertedEvent contains the following constructors:

Constructor	CardInsertedEvent ( Card , CardManager )	
Description	Creates an instance of CardInsertedEvent.	
Syntax	public CardInsertedEvent(Card card, CardManager cardManager)	
Parameters	card	The new card inserted into the slot.
	cardManager	The event source.

# CardManager

```
public class CardManager extends Object
    implements Runnable
```

## DESCRIPTION

**CardManager** performs the following tasks.

- It detects when a card is inserted in or removed from card devices.
- It sends events related to cards to registered listeners.
- It requests the card history each time a card is inserted.

Each time a card is inserted into a device, the following events occur:

- 1 **CardManager** creates an instance of a card subclass for the type of card, for example, **SECCard**.
- 2 The card is reset.
- 3 **CardManager** requests the card history to identify the card type.

## HIERARCHY

```
java.lang.Object
↳ mhw.card.CardEvent
```

## ATTRIBUTES

The class **CardManager** contains the following attributes:

Attribute	LEFT
Description	Indicates the constant value for the left card slot.
Syntax	<code>public static final int LEFT</code>

<b>Attribute</b>	<b>RIGHT</b>
<b>Description</b>	Indicates the constant value for the right card slot.
<b>Syntax</b>	<code>public static final int RIGHT</code>

## CONSTRUCTORS

The class `CardManager` contains the following constructors:

<b>Constructor</b>	<b><code>CardManager(int)</code></b>	
<b>Description</b>	Creates an instance of <code>CardManager</code> .	
<b>Syntax</b>	<code>public CardManager(int maxNumberOfSlots) throws CardException, ManagerException, IllegalArgumentException</code>	
<b>Parameters</b>	<b><code>maxNumberOfSlots</code></b>	The maximum number of cards that can be managed. If <code>maxNumberOfSlots</code> equals: <ul style="list-style-type: none"><li>■ 1, only the left card device is used.</li><li>■ 2, both the left and right card devices are used.</li></ul>
<b>Throws</b>	<b><code>CardException</code></b>	If the card manager exists already.
	<b><code>ManagerException</code></b>	If the manager can open a device channel, or if it cannot dialogue with card devices.
	<b><code>IllegalArgumentException</code></b>	If the parameter <code>maxNumberOfSlots</code> is not valid.

## METHODS

The class `CardManager` contains the following methods:

<b>Method</b>	<b><code>addResourceStatusEventListener(ResourceStatusListener)</code></b>	
<b>Description</b>	Implements the method <code>addResourceStatusEventListener</code> in the class <code>org.davic.resources.ResourceServer</code> .	
<b>Syntax</b>	<code>public synchronized void addResourceStatusEventListener(ResourceStatusListener l)</code>	
<b>Parameters</b>	<b>1</b>	The listener that wants to receive card events.

<b>Method</b>	<b>getCards ( )</b>
<b>Description</b>	Gets all available cards.
<b>Syntax</b>	<code>public synchronized Card[] getCards()</code>
<b>Returns</b>	All available cards, or null.

<b>Method</b>	<b>getInstance ( )</b>
<b>Description</b>	Gets the card manager instance.
<b>Syntax</b>	<code>public static CardManager getInstance()</code>
<b>Returns</b>	The card manager instance.

<b>Method</b>	<b>getMaxNumberOfCards ( )</b>
<b>Description</b>	Gets the maximum number of connectable cards.
<b>Syntax</b>	<code>public synchronized int getMaxNumberOfCards</code>
<b>Returns</b>	The maximum number of connectable cards.

<b>Method</b>	<b>getNumberOfCards ( )</b>
<b>Description</b>	Gets the number of connected cards.
<b>Syntax</b>	<code>public synchronized int getNumberOfCards</code>

Method	<code>removeResourceStatusEventListener()</code>	
Description	Implements the method <code>removeResourceStatusEventListener</code> in the class <code>org.davic.resources.ResourceServe</code> .	
Syntax	<code>public synchronized void removeResourceStatusEventListener (ResourceStatusListener l)</code>	
Parameters	1	The listener that no longer wants to receive card events.

Method	<code>run()</code>	
Description	Runs the card manager, and called when the thread that runs the card manager is started.	
Syntax	<code>public void run()</code>	

# CardRemovedEvent

```
public class CardRemovedEvent extends CardEvent
```

## DESCRIPTION

**CardRemovedEvent** indicates that a card was removed from a card device.

## HIERARCHY

```
java.lang.Object
↳ java.util.EventObject
    ↳ org.davic.resources.ResourceStatusEvent
        ↳ mhw.card.CardEvent
            ↳ mhw.card.CardRemovedEvent
```

## CONSTRUCTORS

The class **CardRemovedEvent** contains the following constructors:

Constructor	<b>CardRemovedEvent ( int , Card , CardManager )</b>	
<b>Description</b>	Creates an instance of <b>CardRemovedEvent</b> .	
<b>Syntax</b>	<b>public CardRemovedEvent(int slotNumber, Card card, CardManager cardManager)</b>	
<b>Parameters</b>	<b>slotNumber</b>	The slot number from which the card was removed.
	<b>card</b>	The Card object that was removed.
	<b>cardManager</b>	The CardManager object that is the event source.

## METHODS

The class **CardRemovedEvent** contains the following methods:

Method	<b>getSlotNumber ( )</b>
<b>Description</b>	Gets the slot number from which the card was removed.
<b>Syntax</b>	<b>public int getSlotNumber()</b>
<b>Returns</b>	The slot number from which the card was removed.

# CardStatusException

public class **CardStatusException** extends CardException

## DESCRIPTION

**CardStatusException** is thrown if a card refuses a command. It gives the identifier of the refused command and the status words returned by the card.

## HIERARCHY

```
java.lang.Object
↳ java.lang.Throwable
  ↳ java.lang.Exception
    ↳ mhw.device.DeviceException
      ↳ mhw.device.CardException
        ↳ mhw.card.CardStatusException
```

## CONSTRUCTORS

The class `CardStatusException` contains the following constructors:

Constructor	<code>CardStatusException(Card,String,byte,byte)</code>	
Description	Creates a <code>CardStatusException</code> without a message.	
Syntax	<code>public CardStatusException(Card card, String cmd, byte sw1, byte sw2)</code>	
Parameters	<code>card</code>	The card that refused the command.
	<code>cmd</code>	The command refused by the card.
	<code>sw1</code>	The first status word, that is, the most significant byte.
	<code>sw2</code>	The second status word, that is, the least significant byte.



<b>Constructor</b>	<b>CardStatusException(Card, String, short)</b>	
<b>Description</b>	Creates CardStatusException without a message.	
<b>Syntax</b>	<pre>public CardStatusException(Card card, String cmd, byte sw1,                            byte sw2)</pre>	
<b>Parameters</b>	<b>card</b>	The card that refused the command.
	<b>cmd</b>	The command refused by the card.
	<b>sw1</b>	The first status word, that is, the most significant byte.
	<b>sw2</b>	The second status word, that is, the least significant byte.
<b>Description</b>	Creates an instance of CardRemovedEvent with a message.	
<b>Syntax</b>	<pre>public CardRemovedEvent(int slotNumber, Card card,                         CardManager cardManager)</pre>	
<b>Parameters</b>	<b>slotNumber</b>	The slot number from which the card was removed.
	<b>card</b>	The Card object that was removed.
	<b>cardManager</b>	The CardManager object that is the event source.

## METHODS

The class CardStatusException contains the following methods:

<b>Method</b>	<b>getCard()</b>
<b>Description</b>	Gets the card.
<b>Syntax</b>	<pre>public Card getCard()</pre>
<b>Returns</b>	The instance of the card that is no longer available.

Method	<code>getCommand()</code>
Description	Gets the command that was refused.
Syntax	<code>public String getCommand()</code>
Returns	The command that was refused.

Method	<code>getMessage()</code>
Description	Gets the exception message.
Syntax	<code>public String getMessage()</code>
Returns	The exception message.
Overrides	<code>getMessage</code> in the class <code>DeviceException</code>

Method	<code>getStatus()</code>
Description	Gets the status words returned by the device.
Syntax	<code>public short getStatus()</code>
Returns	The status words,.

# CardUnavailableException

```
public class CardUnavailableException extends CardException
```

## DESCRIPTION

**CardUnavailableException** is thrown if an application tries to use an instance of a card that is no longer available, for example, because it has been removed from the card device.

## HIERARCHY

```
java.lang.Object
↳ java.lang.Throwable
    ↳ java.lang.Exception
        ↳ mhw.device.DeviceException
            ↳ mhw.device.CardException
                ↳ mhw.card.CardUnavailableException
```

## CONSTRUCTORS

The class `CardUnavailableException` contains the following constructors:

Constructor	<code>CardUnavailableException(Card)</code>
Description	Creates <code>CardUnavailableException</code> without a message.
Syntax	<code>public CardUnavailableException(Card card)</code>
Parameters	<code>card</code> The instance of the card that is not available.

## METHODS

The class `CardUnavailableException` contains the following methods:

Method	<code>getCard()</code>
Description	Gets the card instance that is related to an unavailable physical card.
Syntax	<code>public Card getCard()</code>
Returns	The instance of the card that is no longer available.

# SECACard

```
public class SECACard extends Event
```

## DESCRIPTION

**SECACard** identifies the conditional access card of the **SOCIETE EUROPEENNE DE CONTROLE D'ACCESS**.

## HIERARCHY

```
java.lang.Object
↳ mhw.card.Card
  ↳ mhw.card.SECACard
```

## ATTRIBUTES

The class `SECACard` contains the following attributes:

Attribute	ACK
Description	Indicates the constant value for the card that accepted the command.
Syntax	<code>public static final short ACK</code>

Attribute	AINT
Description	Indicates an error in the set-top box, smart card or data change protocol.
Syntax	<code>public static final short AINT</code>

Attribute	APMP
Description	Indicates the a user password is required.
Syntax	<code>public static final short APMP</code>

<b>Attribute</b>	<b>AUC</b>
<b>Description</b>	Indicates that the set-top box must rebuild its tables, for example, the generator table.
<b>Syntax</b>	<code>public static final short AUC</code>

<b>Attribute</b>	<b>CABS</b>
<b>Description</b>	Indicates an error in the set-top box, smart card or data change protocol.
<b>Syntax</b>	<code>public static final short CABS</code>

<b>Attribute</b>	<b>CADE</b>
<b>Description</b>	Indicates that the smart card does not contain valid rights.
<b>Syntax</b>	<code>public static final short CADE</code>

<b>Attribute</b>	<b>CADV</b>
<b>Description</b>	Indicates that the smart card does not contain valid rights.
<b>Syntax</b>	<code>public static final short CADV</code>

<b>Attribute</b>	<b>CAGEO</b>
<b>Description</b>	Indicates that a geographical restriction caused the descrambling to fail.
<b>Syntax</b>	<code>public static final short CAGEO</code>

Attribute	CANM
Description	Indicates that a morality level error caused the descrambling to fail.
Syntax	<code>public static final short CANM</code>

Attribute	CDNS
Description	Indicates that the smart card is unknown.
Syntax	<code>public static final short CDNS</code>

Attribute	CLNS
Description	Indicates that the smart card is unknown.
Syntax	<code>public static final short CLNS</code>

Attribute	CTABS
Description	Indicates that certification is not present on the smart card.
Syntax	<code>public static final short CTABS</code>

Attribute	DNA
Description	Indicates an error in the set-top box, smart card or data change protocol.
Syntax	<code>public static final short DNA</code>

Attribute	EABS
<b>Description</b>	Indicates an error in the set-top box, smart card or data change protocol.
<b>Syntax</b>	<code>public static final short EABS</code>

Attribute	ECTX
<b>Description</b>	Indicates that the previous command is bad.
<b>Syntax</b>	<code>public static final short ECTX</code>

Attribute	IVER
<b>Description</b>	Indicates that the command is locked.
<b>Syntax</b>	<code>public static final short IVER</code>

Attribute	lgLength
<b>Description</b>	Indicates the number of bytes used for coding the field <code>Lg</code> , which is the buffer returned by the device, and contains the length of the smart card reply.
<b>Syntax</b>	<code>public static byte lgLength</code>

Attribute	LOCK
<b>Description</b>	Indicates that the smart card is unknown.
<b>Syntax</b>	<code>public static final short LOCK</code>

<b>Attribute</b>	<b>MAX</b>
<b>Description</b>	Indicates that there are no rights for the pay-per-view session.
<b>Syntax</b>	<code>public static final short MAX</code>

<b>Attribute</b>	<b>MAX_OPERATORS</b>
<b>Description</b>	Indicates the maximum number of operators supported by the smart card.
<b>Syntax</b>	<code>public static final byte MAX_OPERATORS</code>

<b>Attribute</b>	<b>ME2P</b>
<b>Description</b>	Indicates a change in the contents of the EEPROM.
<b>Syntax</b>	<code>public static final short ME2P</code>

<b>Attribute</b>	<b>MPIC</b>
<b>Description</b>	Indicates that the password is incorrect.
<b>Syntax</b>	<code>public static final short PMIC</code>

<b>Attribute</b>	<b>MSAT</b>
<b>Description</b>	Indicates an error in the set-top box, smart card or data change protocol.
<b>Syntax</b>	<code>public static final short MSAT</code>



<b>Attribute</b>	<b>OTHER</b>
------------------	--------------

<b>Description</b>	Indicates the card status returned in reply to a command.
--------------------	---

<b>Syntax</b>	<code>public static final short OTHER</code>
---------------	--

<b>Attribute</b>	<b>P1IC</b>
------------------	-------------

<b>Description</b>	Indicates that the command must be resent.
--------------------	--

<b>Syntax</b>	<code>public static final short P1IC</code>
---------------	---

<b>Attribute</b>	<b>P2IC</b>
------------------	-------------

<b>Description</b>	Indicates that the command must be resent.
--------------------	--

<b>Syntax</b>	<code>public static final short P2IC</code>
---------------	---

<b>Attribute</b>	<b>PBEL</b>
------------------	-------------

<b>Description</b>	Indicates that the smart card corrupted.
--------------------	--

<b>Syntax</b>	<code>public static final short PBEL</code>
---------------	---

<b>Attribute</b>	<b>PBLG</b>
------------------	-------------

<b>Description</b>	Indicates that the command must be resent.
--------------------	--

<b>Syntax</b>	<code>public static final short PBLG</code>
---------------	---

Attribute	PDEC
Description	Indicates that the token no longer exists.
Syntax	<code>public static final short PDEC</code>

Attribute	PME2P
Description	Indicates that the EEPROM has not been modified.
Syntax	<code>public static final short PME2P</code>

Attribute	PNT
Description	Indicates the card status returned in reply to a command.
Syntax	<code>public static final short PNT</code>

Attribute	PPLE
Description	Indicates that the purse is full.
Syntax	<code>public static final short PPLE</code>

Attribute	PRW
Description	Indicates a preview of a pay-per-view event.
Syntax	<code>public static final short PRW</code>

Attribute	PVID
Description	Indicates that the purse is empty.
Syntax	<code>public static final short PVID</code>

Attribute	QMD
Description	Indicates an error in the set-top box, smart card or data change protocol.
Syntax	<code>public static final short QMD</code>


Attribute	SINC
Description	Indicates that there are no rights in the smart card to unscramble the streams.
Syntax	<code>public static final short SINC</code>

Attribute	VINC
Description	Indicates the card status returned in reply to a command.
Syntax	<code>public static final short VINC</code>

Attribute	ZABS
Description	Indicates that the command must be resent.
Syntax	<code>public static final short ZABS</code>

METHODS

The class SECACard contains the following methods:

Method	changeDescrambling(ElementaryStream,byte,short,short,short)	
Description	Changes the elementary stream descrambling by changing how the ECMs are filtered before sending them to the smart card.	
Syntax	public void changeDescrambling(ElementaryStream stream, byte filteringMode, short session, short index, short operatorId) throws CAException, CardUnavailableException	
Parameters	stream	The elementary stream to be descrambled.
	filteringMode	The bitmap specifying how to filter the ECM.
	session	The filtering session number.If the filtering is on, a session number is required.
	index	The filtering index number. If the filtering is on, an index number is required.
	operatorId	The filtering operator identification.
		The parameters session, index and operatorId are used if the scrambled event is a pay-per-view event.
Throws	CAException	If the descrambling changes fail.
	CardUnavailableException	If the card is not available.

Method	<code>checkSession(short,int)</code>	
<b>Description</b>	Determines if a pay-per-view session, which corresponds to the given operator identifier and session number, is present in the smart card. If it is, the associated index number is returned.	
<b>Syntax</b>	<pre>public synchronized int checkSession(short operatorId,                                      int sessionNumber) throws CardStatusException, CardException,                                      CardUnavailableException</pre>	
<b>Parameters</b>	<code>operatorId</code>	The operator identifier.
	<code>sessionNumber</code>	The session number.
<b>Throws</b>	<code>CardStatusException</code>	If the smart card refuses an input or output message.
	<code>CardException</code>	If an EMM cannot be sent to the smart card.
	<code>CardUnavailableException</code>	If the card is not available.
<b>Returns</b>	The index of the pay-per-view session corresponding to the operator identification and session number.	

Method	<code>descrambleStream(ElementaryStream)</code>	
<b>Description</b>	Descrambles an elementary stream using the PID of the stream carrying ECMs, which must be used to descramble the stream.	
<b>Syntax</b>	<pre>public synchronized byte descrambleStream(ElementaryStream stream)     throws CAException, CardUnavailableException</pre>	
<b>Parameters</b>	<code>stream</code>	The elementary stream to be descrambled.
<b>Returns</b>	The descrambling identifier.	
<b>Throws</b>	<code>CAException</code>	If descrambling fails to start.
	<code>CardUnavailableException</code>	If the card is unavailable.

Method	ECMRejected(Object)	
<b>Description</b>	Indicates that an ECM was rejected by the smart card. This method tries to use another PID of the stream carrying the ECMs, or to use another module. If this is not successful, the method notifies the module that the unscrambling failed.	
<b>Syntax</b>	<code>public synchronized void ECMRejected(Object info)</code>	
<b>Parameters</b>	<code>info</code>	The device event that: <ul style="list-style-type: none"> <li>■ identifies the rejected ECM</li> <li>■ indicates the reason for the rejection.</li> </ul>

Method	getCAInfo()	
<b>Description</b>	Gets the conditional access information of the module, for example, the operator table.	
<b>Syntax</b>	<code>public Object getCAInfo() throws CardUnavailableException</code>	
<b>Returns</b>	The conditional access module information.	
<b>Throws</b>	<code>CardUnavailableException</code>	If the card is unavailable.

Method	getCASystemId()	
<b>Description</b>	Gets the conditional access module system identifier.	
<b>Syntax</b>	<code>public int getCASystemId</code>	
<b>Returns</b>	The conditional access module system identifier.	

Method	getDescramblingId()	
<b>Description</b>	Gets the descrambling identifier.	
<b>Syntax</b>	<code>public byte getDescramblingId(ElementaryStream stream) throws CardUnavailableException</code>	
<b>Parameters</b>	<code>stream</code>	The scrambled elementary stream.
<b>Returns</b>	The descrambling identifier, or -1, if the stream is not unscrambled by this module.	
<b>Throws</b>	<code>CardUnavailableException</code>	If the card is unavailable.

Method	getECMPid(ElementaryStream)	
<b>Description</b>	Gets the PID of the stream that carries the ECM used to unscramble the given stream.	
<b>Syntax</b>	<code>public int getECMPid(ElementaryStream stream) throws CardUnavailableException</code>	
<b>Parameters</b>	<code>stream</code>	The elementary stream to be descrambled.
<b>Returns</b>	The ECM PID, or 0xFF, if the stream was not unscrambled by this module.	
<b>Throws</b>	<code>CardUnavailableException</code>	If the card is unavailable.

Method	getKey()	
<b>Description</b>	Gets the smart card key.	
<b>Syntax</b>	<code>public synchronized int getKey()</code>	
<b>Returns</b>	The smart card key.	

Method	getOffer (byte)	
Description	Gets the bitmap offer for an operator.	
Syntax	<pre>public synchronized long getOffer(byte zonePointer)     throws CardUnavailableException, IllegalArgumentException</pre>	
Parameters	zonePointer	The zone pointer, which must be a value between 0 and MAX_OPERATORS-1.
Returns	The 64 bits corresponding to the offer, or 0.	
Throws	CardUnavailableException	If the card is not available.
	IllegalArgumentException	If the zone pointer does not correspond to a valid zone.

Method	getOffer (short)	
Description	Gets the bitmap offer for an operator.	
Syntax	<pre>public synchronized long getOffer(short operatorID)     throws CardUnavailableException</pre>	
Parameters	operatorID	The operator identifier.
Returns	The 64 bits corresponding to the offer, or 0, if the table does not contain the operator.	
Throws	CardUnavailableException	If the card is not available.

Method	getOfferDate (byte)	
Description	Gets the end date of an operator offer.	
Syntax	<pre>public synchronized short getOfferDate(byte zonePointer)     throws IllegalArgumentException, CardUnavailableException</pre>	
Parameters	zonePointer	The zone pointer, which must be a value between 0 and MAX_OPERTORS-1.
Throws	IllegalArgumentException	If the zone pointer value is not a valid zone.
	CardUnavailableException	If the card is not available.

© 1998 - 19 9 CANAL+. This document is the intellectual property of CANAL+ and contains confidential information. All reproduction and communication to third parties is strictly prohibited without prior written consent from CANAL+.



Method	getOfferDate ( short )	
<b>Description</b>	Gets the end date of an operator offer.	
<b>Syntax</b>	<pre>public synchronized short getOfferDate(short operatorId)     throws CardUnavailableException</pre>	
<b>Parameters</b>	operatorId	The operator identifier.
<b>Returns</b>	The 16 bits corresponding to the offer date, or 0, if the table does not contain the operator.	
<b>Throws</b>	CardUnavailableException	If the card is not available.

Method	getOpenZones ( )	
<b>Description</b>	Gets an array of open zones.	
<b>Syntax</b>	<pre>public synchronized byte[] getOpenZones()     throws CardUnavailableException</pre>	
<b>Returns</b>	An array of open zones, or null.	
<b>Throws</b>	CardUnavailableException	If the card is not available.

Method	getOperatorName ( byte )	
<b>Description</b>	Gets the name of an operator.	
<b>Syntax</b>	<pre>public synchronized String getOperatorName(byte zonePointer)     throws IllegalArgumentException, CardUnavailableException</pre>	
<b>Parameters</b>	zonePointer	The zone pointer, which must be a value between 0 and MAX_OPERATORS-1.
<b>Returns</b>	The operator name, or null if the table does not contain the operator.	
<b>Throws</b>	IllegalArgumentException	If the zone pointer does not correspond to a valid zone.
	CardUnavailableException	If the card is not available.

<b>Method</b>	<b>getOperatorName ( short )</b>	
<b>Description</b>	Gets the name of an operator.	
<b>Syntax</b>	<pre>public synchronized String getOperatorName(short operatorId)     throws CardUnavailableException</pre>	
<b>Parameters</b>	<b>operatorId</b>	The operator identifier.
<b>Throws</b>	<b>CardUnavailableException</b>	If the card is not available.

<b>Method</b>	<b>getOperators ( )</b>	
<b>Description</b>	Gets the array of operator identifiers present in the smart card.	
<b>Syntax</b>	<pre>public synchronized short[] getOperators()     throws CardUnavailableException</pre>	
<b>Returns</b>	The array of operator identifiers present in the smart card, or null.	
<b>Throws</b>	<b>CardUnavailableException</b>	If the card is not available.

<b>Method</b>	<b>getProductCode ( )</b>	
<b>Description</b>	Gets the smart card product code.	
<b>Syntax</b>	<pre>public synchronized short getProductCode()</pre>	
<b>Returns</b>	The smart card product code.	

<b>Method</b>	<b>getSerialNumber ( )</b>	
<b>Description</b>	Get the smart card serial number.	
<b>Syntax</b>	<pre>public synchronized int getSerialNumber()</pre>	
<b>Returns</b>	The smart card serial number.	

Method	getZonePointer ( short )	
<b>Description</b>	Gets the zone pointer that corresponds to an operator in the operator table.	
<b>Syntax</b>	<pre>public synchronized byte getZonePointer(short operatorId)     throws CardUnavailableException</pre>	
<b>Parameters</b>	operatorId	The operator identifier.
<b>Returns</b>	The zone pointer, or the value -1, if no zone pointer exists for the operator.	
<b>Throws</b>	CardUnavailableException	If the card is not available.

Method	refresh ( )	
<b>Description</b>	Refreshes the conditional access information tables.	
<b>Syntax</b>	<pre>public synchronized void refresh() throws CardStatusException,     CardException, CardUnavailableException</pre>	
<b>Throws</b>	CardStatusException	If the smart card refused an input or an output message.
	CardException	If the method fails for any other reason.
	CardUnavailableException	If the card is unavailable.

Method	stopDescrambling ( )	
<b>Description</b>	Stops all descrambling.	
<b>Syntax</b>	<pre>public synchronized void stopDescrambling()     throws CardUnavailableException</pre>	
<b>Throws</b>	CardUnavailableException	If the card is unavailable.

Method	<code>stopDescramblingStream(ElementaryStream)</code>	
Description	Stops descrambling an elementary stream.	
Syntax	<code>public synchronized void stopDescramblingStream (ElementaryStream stream) throws CardUnavailableException</code>	
Parameters	<code>stream</code>	The elementary scrambled stream.
Throws	<code>CardUnavailableException</code>	If the card is not available.

Method	<code>tryAgainToDescramble()</code>	
Description	Restarts descrambling the elementary stream for which entitlements were not available previously. This method is called by the module manager.	
Syntax	<code>public synchronized void tryAgainToDescramble() throws CAException</code>	
Throws	<code>CAException</code>	If the descrambling cannot be restarted or fails.

Method	<code>writeEMM(byte[])</code>	
Description	Presents an EMM to the smart card in order to write a session or to modify the subscribing offer of the smart card.	
Syntax	<code>public synchronized void writeEMM(byte EMM[]) throws CardStatusException, CardException, CardUnavailableException</code>	
Parameters	<code>EMM</code>	The EMM bytes containing the additional parameters.
Throws	<code>CardStatusException</code>	If the smart card refuses an input or output message.
	<code>CardException</code>	If an EMM cannot be sent to the smart card.
	<code>CardUnavailableException</code>	If the card is unavailable.