

**CARDPRESSO**  
more than an application

# cardPresso **MANUAL**



AVAILABLE FOR:  Microsoft **Windows** 



# cardPresso

Ver 1.2.10

*by cardPresso, Lda*

**MORE THAN AN APPLICATION**

---

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners.

While every precaution has been taken in the preparation of this document, the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

May 2015

# Contents

	0
<b>Introduction</b>	<b>14</b>
<b>Technology / API Implementation Table</b>	<b>16</b>
<b>Objects</b>	<b>20</b>
<b>QExModule</b>	<b>20</b>
<b>Properties</b>	<b>21</b>
moduleName	21
productName	21
vendorName	21
settings	21
log	21
<b>Methods</b>	<b>22</b>
resourcesLocation	22
storageLocation	22
setProperty	22
property	22
sleep	23
abort	23
setLastError	23
lastError	23
enableScriptExceptions	23
scriptExceptions	24
<b>QExSettings</b>	<b>25</b>
<b>Methods</b>	<b>26</b>
value	26
setValue	26
setDefaultValue	26
settingsValue	26
section	26
keys	26
<b>QExSettings Related Objects</b>	<b>27</b>
QExSettingsValue	27
Methods	28
value	28
setValue	28
<b>QExEventLog</b>	<b>29</b>
<b>Methods</b>	<b>30</b>
log	30
logError	30
<b>QExError</b>	<b>30</b>
<b>QExError Properties</b>	<b>32</b>
isSet	32
severity	32
description	32
toString	32
text	32
<b>QExError Methods</b>	<b>33</b>
additionalInfo	33
addAdditionalInfo	33
reset	33
<b>QExScriptData</b>	<b>34</b>
<b>Methods</b>	<b>35</b>
isNull	35
append	35
rightJustified	35
leftJustified	35
rightJustified	35

leftJustified	35
left	35
right	36
mid	36
size	36
toLower	36
toUpper	36
reversed	36
toHex	36
toBase64	36
toUtf8	37
toString	37
toInt	37
toUInt	37
toBool	37
toByteArray	37
toVariant	37
fromHex	37
fromBase64	37
fromString	37
replace	38
writeToFile	38

## Modules

42

<b>Application</b>	<b>43</b>
addModule	43
addModule	43
exec	44
quit	44
log	44
debug	44
print	44
showMessage	44
showError	45
runScript	45
runScripts	45
runScriptWizard	45
loadCSS	45
loadCSSFolder	46
loadActions	46
applicationPath	46
applicationFilePath	46
<b>User Interface</b>	<b>47</b>
<b>Security</b>	<b>48</b>
login	48
isValid	48
<b>Encoding</b>	<b>48</b>
<b>Methods</b>	<b>49</b>
encoder	49
encoders	49
<b>Encoder Object</b>	<b>50</b>
Properties	51
name	51
vendor	51
model	51
apis	51
Methods	52
isValid	52
waitForTag	52
tagsInRange	53
<b>Generic Tag Object</b>	<b>54</b>
Generic Tag Properties	55

type	55
description	55
uid	55
isValid	55
read	55
security	55
isValid	55
login	55
data	56
isValid	56
size	56
read	56
write	56
config	57
isValid	57
size	57
read	57
write	57
Generic Tag Methods	58
isValid	58
Generic Tag Sample Script	59
<b>Contactless Tag Types</b>	<b>60</b>
MIFARE®	60
MIFARE Classic®	60
UID	62
isValid	62
read	62
Data	62
size	62
readBlock	62
writeBlock	62
read	62
write	62
formatMAD	62
isMADConfigured	62
getMADMultiApplication	63
setMADMultiApplication	63
getMADCardHolderInfo	63
setMADCardHolderInfo	63
removeMADCardPublisherInfo	63
getMADCardPublisherInfo	63
setMADCardPublisherInfo	64
getMADApplicationIdentifier	64
setMADApplicationIdentifier	64
isMADSectorFree	64
Config	64
size	64
read	64
write	64
readTrailer	65
writeTrailer	65
API	66
tagSectors	67
blocksPerSector	67
login	67
dataSize	67
configSize	67
readSectorBlock	67
writeSectorBlock	67
read	67
write	68
readSectorTrailer	68
writeSectorTrailer	68
Mifare Classic MAD (API)	69

formatMAD	69
readMADSettings	69
writeMADSettings	69
cpMifareMADSettings Object	69
getMADVersion	70
getMultiApplication	70
setMultiApplication	70
getMADAvailable	70
setMADAvailable	70
getCardHolderInfo	70
setCardHolderInfo	70
removeCardPublisherInfo	70
getCardPublisherInfo	71
setCardPublisherInfo	71
getApplicationIdentifier	71
setApplicationIdentifier	71
isSectorFree	71
MIFARE Classic Sample Script	71
MIFARE Classic Wizards	73
MIFARE® DESFire®	73
UID	76
isValid	76
read	76
Security	77
isValid	77
login	77
loginISO	77
loginISO24	77
loginAES	77
changeKey	77
changeKeyISO	77
changeKeyISO24	77
changeKeyAES	77
keyDiversificationAES	78
keyDiversificationAESAV1	78
keyDiversificationAESAV2	78
keyDiversification2TDEA	78
keyDiversification3TDEA	78
Data	79
isValid	79
size	79
formatCard	79
createApplication	79
deleteApplication	79
getApplicationIDS	79
getFreeMemory	79
getDFNames	80
selectApplication	80
createFile	80
createBackupFile	80
createValueFile	80
createLinearRecordFile	81
createCyclicRecordFile	81
deleteFile	81
getFileIDs	82
getISOFileIDs	82
write	82
read	82
getValue	82
credit	82
debit	83
limitedCredit	83
writeRecord	83
readRecords	83



clearRecordFile	84
commitTransaction	84
abortTransaction	84
Data (NFC)	84
formatNfc	84
w riteNfcText	84
w riteNfcUri	84
w riteNfcIcon	85
w riteNfcSmartPoster	85
prepareNfcVCard	85
addNfcVCardField	85
w riteNfcVCard	86
lockNfcApplication	86
Config	86
isValid	86
setConfiguration	86
getKeySettings	86
changeKeySettings	86
getFileSettings	86
changeFileSettings	87
getVersion	87
getKeyVersion	87
NFC MIFARE DESFire API	89
SetNfcDESFireVersion	89
GetNfcDESFireVersion	89
MIFARE DESFire Wizards	90
STMicroelectronics	90
SRTAG-D	90
UID	92
isValid	92
read	92
Security	92
isValid	92
login	92
loginRead	92
loginWrite	92
changeReadKey	92
changeWriteKey	92
Data	92
isValid	92
size	93
w rite	93
read	93
formatNfc	93
w riteNfcText	93
w riteNfcUri	93
w riteNfcIcon	93
w riteNfcSmartPoster	93
prepareNfcVCard	94
addNfcVCardField	94
w riteNfcVCard	94
lockNfcApplication	94
lockNfcApplicationForever	95
unlockNfcApplication	95
Config	95
isValid	95
SRTAG-D API	95
Verify	95
ChangeReferenceData	95
DisableVerificationRequirement	95
SelectApplication	95
SelectFile	95
CapabilityContainerSelect	95
NDEFSelect	95

SystemFileSelect	95
enablePermanentState	96
Contactless APIs	97
NFC	97
formatNfc	97
lockNfcApplication	97
createNDEFMessage	97
readNDEFMessage	98
writeNDEFMessage	98
createNfcTextRecord	98
createNfcUriRecord	98
createNfcIconRecord	98
createNfcSmartPosterRecord	98
createNfcvCardRecord	99
NFC Record Objects	99
NDEFTextRecord	100
getLocale	100
setLocale	100
getText	100
setText	100
getEncoding	100
setEncoding	100
NDEFUriRecord	100
getUri	101
setUri	101
NDEFIconRecord	101
getData	101
setData	101
NDEFSmartPosterRecord	101
hasTitle	102
getTitleCount	102
getTitleRecord	102
getTitleRecords	102
getTitle	102
addTitle	102
removeTitle	103
getUriRecord	103
getUri	103
setUri	103
hasAction	103
getAction	103
setAction	103
hasSize	103
getSize	103
setSize	103
hasTypeInfo	104
getTypeInfo	104
setTypeInfo	104
hasIcon	104
getIconCount	104
getIconRecord	104
getIconRecords	104
getIcon	104
addIcon	104
removeIcon	104
NDEFvCardRecord	104
addField	105
Common Record Methods	105
getTypeNameFormat	105
setTypeNameFormat	105
getType	105
setType	105
getId	105
setId	105

getPayload	105
setPayload	106
isEmpty	106
HID	106
ISO Prox	106
UID	106
isValid	106
read	106
security	106
isValid	106
Data	106
isValid	106
size	106
read	106
write	107
Config	107
isValid	107
size	107
<b>Contact Tag Types</b>	<b>107</b>
SLE4442	107
UID	109
read	109
isValid	109
security	109
login	109
verifyPin	109
changePin	109
login	109
verifyPin	109
changePin	110
readSecurityMemory	110
updateSecurityMemory	110
updateSecurityMemory	110
Data	110
size	110
read	110
write	111
Config	111
protectionMemorySize	111
securityMemorySize	111
readProtectionMemory	111
writeProtectionMemory	111
writeProtectionMemory	111
SLE4442 Sample Script	113



# Introduction

# Introduction

cardPresso 2.0 advanced script command reference.

Here you will find the reference to the advanced commands used in cardPresso.

This document is a draft version. The information found in this document is still under internal review. The contents of this document may be subject to changes. The accuracy of the information provided is not guaranteed. cardPresso, Lda. holds no liability for the consequences of the usage of the information in this document.

For support requests, find us at:

Website: <http://www.cardpresso.com>

Support: [support@cardpresso.com](mailto:support@cardpresso.com)

# **Technology / API Implementation Table**

# Technology / API Implementation Table

## TAG API's

Microsoft	HID	IDENTIVE/SCM		ACS	DUALi	CASTLES	GEM	ELATEC	RF IDEAS	ELICTYS	EWT	SPRING CARD
PCSC	OK 53x1	SDIO1X	CLOUD	ACR 38	DE - 620	EZ100PU	USB-TR CT30	TWN3 MULTI ISO	Pc Prox Plus	v 2.0	J68x	Crazy Writer

		Microsoft	HID	IDENTIVE/SCM	ACS	DUALi	CASTLES	GEM	ELATEC	RF IDEAS	ELICTYS	EWT	SPRING CARD	
GENERIC TAG	UID	M S	M S	M S	M S				M S			M S	M S	
	Security	M S	M S	M S	M S				M S			M S	M S	
	Data	M S	M S	M S	M S				M S			M S	M S	
	Config	M	M	M					M			M	M	
	MAD													
	NFC													
MIFARE	Classic	UID	M S W	M S W	M S W	M S W		M S W	M S	M	M S W	M S W	M S W	
		Security	M S W	M S W	M S W	M S W		M S W	M S		M S W	M S W	M S W	
		Data	M S W	M S W	M S W	M S W		M S W	M S		M S	M S	M S	
		Config	M W	M W	M W	M W		M W		M W		M W	M W	M W
		NFC												
	Desfire	MAD	M W	M W	M W	M W				M		M W	M	M W
		UID	M W	M W	M W	M W					M	M W		M W
		Security	M W	M W	M W	M W						M W		M W
		Data	M W	M W	M W	M W						M W		M W
		Config	M W	M W	M W	M W						M W		M W
Ultralight	NFC	M W	M W	M W	M W						M W		M W	
	UID	M	M	M	M					M				
	Security	M	M	M	M									
	Data	M	M	M	M									
SLE	SLE44x2 SLE55x2	Config	M	M	M	M								
		Data	M S	M S	M S	M S	M S		M S			M S		
		Security	M S	M S	M S	M S	M S		M S			M S		
ISO15693	ISO 15693	UID	M	M	M	M					M			
		Security	M	M	M	M								
		Data	M	M	M	M								
		Config	M	M	M									
		NFC												
HID	ISO PROX	UID	M	M							M			
		Security												



		Data													
		Config													
EM	410x	UID	M	M							M	M			
	4200	UID	M	M							M	M			
ST	SRTAG-D	UID	M	M	M	M									
		Security		M	M	M									
		Data		M	M	M									
		Config		M	M	M									
		NFC													



# Objects

# Objects

## 3.1.QExModule Module Reference

The **QExModule** is the base module from which other cardPresso modules derive. It also provides properties and methods to retrieve general information regarding the software and manage global configurations of other modules, properties and functions.

### Properties:

The **QExModule** properties provide general information regarding the software and software logs. Additionally, the [.settings](#) property provides access to the [Settings](#) object and it's inherent methods.

See more: [Settings](#).

#### Returns:

#### Property:

<i>QString</i>	<a href="#">.vendorName</a>
<i>QString</i>	<a href="#">.productName</a>
<i>QString</i>	<a href="#">.moduleName</a>
<i>QExSettings</i>	<a href="#">.settings</a>
<i>QExEventLog</i>	<a href="#">.log</a>

### Methods:

The **QtExModule** methods provide general information and configuration regarding the software settings, errors and basic software configuration.

#### Returns:

#### Method:

<i>String</i>	<a href="#">.resourcesLocation</a> ( <i>string section</i> )
<i>String</i>	<a href="#">.storageLocation</a> ( <i>Int location</i> )
<i>DataBlock</i>	<a href="#">.execAction</a> ( <i>String actionID, String scriptID</i> )
<i>QObject</i>	<a href="#">.createQObject</a> ( <i>String objectName, QScriptValue P1, QScriptValue P2, QScriptValue P3</i> )
<i>QScriptValue</i>	<a href="#">.createObject</a> ( <i>String objectName, QScriptValue P1, QScriptValue P2, QScriptValue P3</i> )
<i>Void</i>	<a href="#">setProperty</a> ( <i>String ID, value</i> )
<i>QVariant</i>	<a href="#">property</a> ( <i>String ID</i> )
<i>Bool</i>	<a href="#">sleep</a> ( <i>Int milliseconds</i> )
<i>Bool</i>	<a href="#">abort</a> ( <i>String errorMessage</i> )
<i>Void</i>	<a href="#">setLastError</a> ( <i>QExError error</i> )
<i>QExError</i>	<a href="#">lastError</a> ();
<i>Void</i>	<a href="#">enableScriptExceptions</a> ( <i>Bool enabled</i> )
<i>Bool</i>	<a href="#">scriptExceptions</a> ();

### 3.1.1.Properties

Enter topic text here.

#### **moduleName**

*QString* moduleName

Retrieves the name of the module as a String.

#### **productName**

*QString* productName

Returns the software product name as a String.

#### **vendorName**

*QString* vendorName

Returns the software vendor name as a String.

#### **settings**

*QExSettings* settings

Returns the software settings as a [Settings Object](#) Object.

Read more: [Settings Object](#)

#### **log**

*QExSettings* log

Returns the software log as an [EventLog Object](#).

Read more: [EventLog Object](#).

## 3.1.2.Methods

### resourcesLocation

*String* resourcesLocation(*string* section)

Returns the module resources location as a String.

If no section is specified, the default value will be automatically used.

### storageLocation

*String* storageLocation(*Int* location)

Required params: location

Returns the current user's operating system general storage folders as a string.

Valid **location** values are:

0. Current User Desktop Folder
1. Current User Documents folder
2. System Fonts folder
3. Current User Start menu Program folder
4. Current User Music folder
5. Current User Videos folder
6. Current User Pictures folder
7. cardPresso local Encoding folder
8. Current User local folder
9. Current User local cardPresso Installation folder
10. Local Cache folder

### setProperty

*Bool* setProperty(*String* property, *value*)

Required params: time.

Configures the specified **property** with the provided **value**.

The following properties are available.

Property	Description	Value Data Type	Default Value
TIMEOUTS.POOLING.TAG	Time to wait for a tag to be placed.	Integer (milliseconds)	30000
DETECTEDTAGS.IGNORED UPPLICATEUIDS	Enables or disables duplicate UID detection.	Bool (True/False)	True

### property

*QVariant* property(*String* id)

Required params: id

Returns the value of the specified property.

## sleep

```
Bool sleep(Int milliseconds)
```

Required params: milliseconds

Pauses the script at the point of the **sleep** method for the amount of time set in **milliseconds** (ms). Returns 'true' if the method is executed. Returns false if the method is interrupted by the user.

## abort

```
Bool abort(String errorMessage)
```

Required params: errorMessage

Interrupts the script at the point of the **abort** method and displays the specified **errorMessage** as an error message.

## setLastError

```
Void setLastError(QExError error)
```

Required params: error

Configures the [lastError](#) with the [Error Object](#) provided by the user or returned by an error event.

Note:

If an expected error occurs during script execution that error will be set as the lastError Object.

Unexpected errors are not set as a lastError Object.

If [scriptExceptions](#) are enabled, if an error occurs, the script stops being executed an error message is returned and the error is set as the [lastError](#).

If [scriptExceptions](#) are disabled, if an error occurs, the script keeps being executed and the error is set as the [lastError](#).

## lastError

```
QExError lastError()
```

Returns the [Error Object](#) currently set as the lastError by the [setLastError](#) method as an [Error Object](#).

Note:

If an expected error occurs during script execution that error will be set as the lastError Object.

Unexpected errors are not set as a lastError Object.

If [scriptExceptions](#) are enabled, if an error occurs, the script stops being executed an error message is returned and the error is set as the [lastError](#).

If [scriptExceptions](#) are disabled, if an error occurs, the script keeps being executed and the error is set as the [lastError](#).

## enableScriptExceptions

```
Void enableScriptExceptions(Bool enabled)
```

Required params: enabled

Enables [scriptExceptions](#) if set to true 'true' and disables [scriptExceptions](#) if set to 'false'.

Note:

If an expected error occurs during script execution that error will be set as the lastError Object.

Unexpected errors are not set as a lastError Object.

If [scriptExceptions](#) are enabled, if an error occurs, the script stops being executed an error message is returned

and the error is set as the [lastError](#).

If [scriptExceptions](#) are disabled, if an error occurs, the script keeps being executed and the error is set as the [lastError](#).

## scriptExceptions

```
Bool scriptExceptions()
```

Checks if script exceptions are enabled. Returns 'true' if enabled and 'false' if disabled.

See also: [enableScriptExceptions](#).



## 3.2.QExSettings Settings Object Reference

The **QExSettings** Object provides access to the methods that allow us to manage the module settings and configurations, stored by default on the application configuration file (.conf).

### Methods:

The **QtExSettings** Object provides the following methods:

Returns:	Method:
<i>String</i>	<a href="#">value</a> ( <i>String id</i> )
<i>Bool</i>	<a href="#">setValue</a> ( <i>String, id,String value</i> );
<i>Bool</i>	<a href="#">setDefaultValue</a> ( <i>String, id,String value</i> );
<a href="#">QExSettingsValue</a>	<a href="#">settingsValue</a> ( <i>String id, SaveMode</i> );
<i>QExSettings</i>	<a href="#">section</a> ( <i>String id</i> )
<i>KeysList</i>	<a href="#">keys</a> ()

### 3.2.1.Methods

#### value

```
String value(String id)
```

Required params: id

Returns the **value** currently set on the application configuration file setting identified by the **id**.

#### setValue

```
Bool setValue(String, id,String value);
```

Required params: id, value

Adds a new setting to the application configuration file with the specified **value** identified by the user submitted **id**.

#### setDefaultValue

```
Bool setDefaultValue(String, id,QVariant value);
```

Required params: id, value

Adds a session default value to a setting (**id**) with the specified **value**. If the setting (**id**) is not defined in the application configuration file, this default value will be used.

#### settingsValue

```
QExSettingsValue settingsValue(String, id, SaveMode);
```

Required params: id, value

Instantiates a [Settings Vlaue](#) object with the specified **id** and **SaveMode**.

#### section

```
QExSettings section(String id)
```

Required params: id

Allows us to use a [QExSettings](#) object as a section. Providing access to the methods that allow us to add settings under that same section.

```
encoding.settings.section("section").section("subsection").setValue("settingmeta","valuemeta");
```

#### keys

```
KeysList keys()
```

Returns a list of the currently configured settings and values in a [Settings](#) Object in an array.

## 3.2.2.QExSettings Related Objecs

### QExSettingsValue

The **SettingsValue** Object provides provides easier setting manipulation by allowing to create an instance of a specific setting in a unique object which saves itself in the application configuration file each time it's value is altered.

The QExSettingsValue object holds the ID of the setting, the value of the setting and the **SaveMode** of the object.

Available **SaveMode** values are:

**SaveAlways:** The setting is saved in the configuration file even if the value is not changed.

**OnlySavelfChanged:** The setting is only saved in the configuration file only if the value is changed.

### Methods:

The **SettingsValue** Object provides the following methods:

#### Returns:

*String*

*Bool*

#### Method:

[value](#)(string **id**)

[setValue](#)(string **id**, string **value**)

## Methods

value

```
String value(String id)
```

Required params: id

Returns the **value** currently set on the [Settings Value](#) Object identified by the **id**.

setValue

```
Bool setValue(String, id,String value);
```

Required params: id, value

Sets the **value** of the [Settings Value](#) Object identified by the specified **id**.

### 3.3.QExEventLog EventLog Object Reference

The **QExEventLog** Object provides access to the methods that allow us to handle event logs.

#### Methods:

The **QExEventLog** Object provides the following methods:

Returns:	Method:
<i>Void</i>	<a href="#">log</a> (LogLevel <b>level</b> ,String <b>msg</b> ,QExDataHash <b>data</b> ,DataType <b>type</b> );
<i>Void</i>	<a href="#">logError</a> (QExError <b>error</b> );
<i>Void</i>	<a href="#">logData</a> (LogLevel <b>level</b> ,String <b>dataID</b> ,QExDataHash <b>data</b> , DataType <b>type</b> );
<i>Void</i>	<a href="#">logDebug</a> (QVariant <b>object</b> ,QVariant <b>function</b> ,QVariant <b>description</b> ,QVariant <b>parameters</b> ,QExDataHash <b>data</b> ,DataType <b>type</b> );
<i>Void</i>	<a href="#">logWarning</a> (QVariant <b>object</b> ,QVariant <b>function</b> ,QVariant <b>description</b> ,QVariant <b>parameters</b> ,QExDataHash <b>data</b> ,DataType <b>type</b> );
<i>Void</i>	<a href="#">addHandler</a> (QExEventLogHandlerRef <b>handler</b> );

### 3.3.1.Methods

#### log

```
Void log(LogLevel level,String msg,QExDataHash data,DataType type)
```

Required params: level, msg

Logs a user specified message (**msg**) with the specified log level (**level**).

Specific data can also be logged with the **data** parameter.

A specific data type can also be specified with the **type** parameter.

Note:

If no **data** is specified, cardPresso will provide a value by default.

If no **type** is specified, cardPresso will use DATA\_TYPE\_NULL by default.

If a data **type** is specified , the **data** being handled must also be provided.

Valid **LogLevel** values:

**LOG\_OFF** - The log is off.

**LOG\_DEBUG** - The Debug log level.

**LOG\_WARNING** - The Warning log level.

**LOG\_ERROR** - The Error log level.

**LOG\_CRITICAL** - The Critical log level.

**LOG\_FATAL** - The Fatal log level.

**LOG\_MEMORY** - The Memory log level.

(QExEventLog.h)

Valid **DataType** values:

**DATA\_TYPE\_NULL** - No data was supplied or data type is unknown.

**DATA\_TYPE\_IN** - The data is related to incoming data (received).

**DATA\_TYPE\_OUT** - The data is related to outgoing data (sent).

**DATA\_TYPE\_ERROR** - The data is related to an error that occurred.

**DATA\_TYPE\_USER** - The data is user defined. Any program can use DATA\_TYPE\_USER + N to have different logic for different types of data.

#### logError

```
Void logError(QExError error)
```

Required params: error

Logs the specified [error](#) object.

## 3.4.QExError Error Object Reference

The **QExError** Object provides access to the methods that provide generic error information, regarding errors that have occurred. It also allows to export error information to external scripts.

#### Methods:

The **QExError** Object provides the following Properties and Methods:

**Returns:**

**Property:**

<i>Bool</i>	<a href="#">isSet</a>
<i>String</i>	<a href="#">severity</a>
<i>String</i>	<a href="#">description</a>
<i>String</i>	<a href="#">toString</a>
<i>String</i>	<a href="#">text</a>

**Returns:**

<i>String</i>	<a href="#">addicionalInfo</a> ( <i>String</i> key)
<i>Void</i>	<a href="#">addAdicionalInfo</a> ( <i>String</i> key, <i>QVariant</i> value);
<i>Void</i>	<a href="#">reset</a> ()

**Method:**

### 3.4.1.QExError Properties

#### **isSet**

*Bool* isSet

Verifies if there is an error object currently set. Returns 'true' if set or 'false' if not set.

#### **severity**

*String* severity

Returns the severity of the error as a string.

#### **description**

*String* description

Returns the error description as a string.

#### **toString**

*String* toString

Returns the error as a string.

#### **text**

*String* text

Returns the error as a string.



## 3.4.2.QExError Methods

### additionalInfo

```
String additionalInfo(String key)
```

Required params: key

Returns the available additional information identified by the **key** as a string.

Note:

Additional info may need to be previously added by using the method [addAdicionalInfo](#).

### addAdicionalInfo

```
Void addAdicionalInfo(String key, String value);
```

Required params: key, value

Allows the user to add a detailed description of the error object.  
More than one additional info may be added per error object.  
Each additional info entry must be identified by a unique **key**.

The additional info **value** of each **key** can then be return by using the [adicionalInfo](#) method.

### reset

```
Void reset()
```

Resets the current error object.

## 3.5.QScriptValue ScriptData Object Reference

The **QScriptValue** Object provides access to the methods that provide generic data manipulation. It is prepared to handle the most commonly used types of data and perform conversion operations between them.

### Methods:

The **QScriptValue** Object provides the following Properties and Methods:

Returns:	Method:
<i>bool</i>	<a href="#">isNull()</a>
<i>void</i>	<a href="#">append</a> ( <i>QScriptValue</i> <b>string</b> )
<i>QScriptValue</i>	<a href="#">rightJustified</a> ( <i>int</i> <b>width</b> , <i>string</i> <b>fill</b> )
<i>QScriptValue</i>	<a href="#">leftJustified</a> ( <i>int</i> <b>width</b> , <i>string</i> <b>fill</b> )
<i>QScriptValue</i>	<a href="#">rightJustified</a> ( <i>int</i> <b>width</b> , <i>quint8</i> <b>fill</b> )
<i>QScriptValue</i>	<a href="#">leftJustified</a> ( <i>int</i> <b>width</b> , <i>quint8</i> <b>fill</b> )
<i>QScriptValue</i>	<a href="#">left</a> ( <i>int</i> <b>length</b> );
<i>QScriptValue</i>	<a href="#">right</a> ( <i>int</i> <b>length</b> )
<i>QScriptValue</i>	<a href="#">mid</a> ( <i>int</i> <b>pos</b> , <i>int</i> <b>len</b> )
<i>int</i>	<a href="#">size</a> ()
<i>QScriptValue</i>	<a href="#">toLowerCase</a> ()
<i>QScriptValue</i>	<a href="#">toUpperCase</a> ()
<i>QScriptValue</i>	<a href="#">reversed</a> ()
<i>QString</i>	<a href="#">toHex</a> ()
<i>QString</i>	<a href="#">toBase64</a> ()
<i>QString</i>	<a href="#">toUtf8</a> ()
<i>QString</i>	<a href="#">toString</a> ()
<i>qint64</i>	<a href="#">toInt</a> ()
<i>qint64</i>	<a href="#">toUInt</a> ()
<i>QScriptValue</i>	<a href="#">toBool</a> ()
<i>QByteArray</i>	<a href="#">toByteArray</a> ()
<i>QVariant</i>	<a href="#">toVariant</a> ()
<i>QScriptValue</i>	<a href="#">fromHex</a> ( <i>string</i> <b>hex</b> )
<i>QScriptValue</i>	<a href="#">fromBase64</a> ( <i>string</i> <b>base64</b> )
<i>QScriptValue</i>	<a href="#">fromString</a> ( <i>string</i> <b>data</b> )
<i>void</i>	<a href="#">replace</a> ( <i>QScriptValue</i> <b>before</b> , <i>QScriptValue</i> <b>after</b> );
<i>bool</i>	<a href="#">writeToFile</a> ( <i>string</i> <b>fileName</b> )
<i>bool</i>	<a href="#">writeToFileD</a> ( <i>string</i> <b>fileName</b> )

## 3.5.1.Methods

Enter topic text here.

### isNull

```
Bool isNull()
```

Returns true if the ScriptData Object is null and false if the object is not null.

### append

```
void append (QExScriptValue string)
```

When working with strings, appends the user specified **string** to the ScriptData Object.

### rightJustified

```
QScriptValue rightJustified(int width,string fill)
```

When working with strings, it transforms the string to the specified length (**width**). If the string is shorter than the specified width, the empty characters will be filled with the character specified with the **fill** parameter. Characters will be added to the end of the string.

### leftJustified

```
QScriptValue leftJustified(int width,string fill)
```

When working with strings, it transforms the string to the specified length (**width**). If the string is shorter than the specified width, the empty characters will be filled with the character specified with the **fill** parameter. Characters will be added to the beginning of the string.

### rightJustified

```
QScriptValue rightJustified(int width,quint8 fill)
```

When working with bytes, it transforms the byte to the specified length (**width**). If the byte is shorter than the specified width, the empty bytes will be filled with the byte specified with the **fill** parameter. Fill bytes will be added to the end of the original byte.

### leftJustified

```
QScriptValue leftJustified(int width,quint8 fill)
```

When working with bytes, it transforms the byte to the specified length (**width**). If the byte is shorter than the specified width, the empty bytes will be filled with the byte specified with the **fill** parameter. Fill bytes will be added to the start of the original byte.

### left

```
QScriptValue left(int length);
```

When working with strings, it returns the characters in the length specified by the **length** parameter, starting from the left of the string.

Note:

Character count starts at the number 0.

## right

*QScriptValue* **right** (*int* length)

When working with strings, it returns the characters in the length specified by the **length** parameter, starting from the left of the string.

Note:

Character count starts at the number 0.

## mid

*QScriptValue* **mid** (*int* pos, *int* length )

When working with strings, it returns the characters in the length specified by the **length** parameter, starting from the position within the string, specified by the **pos** parameter.

Note:

Character count starts at the number 0.

## size

*int* **size**()

Returns the size of the ScriptData Object as an integer.

## toLowerCase

*QScriptValue* **toLowerCase**()

Returns the ScriptData Object in lowercase, and returns it as a ScriptData Object.

## toUpperCase

*QScriptValue* **toUpperCase**()

Returns the ScriptData Object in uppercase, and returns it as a ScriptData Object.

## reversed

*QScriptValue* **reversed**()

Returns the ScriptData Object reversed, and returns it as a ScriptData Object.

## toHex

*String* **toHex**()

Converts the ScriptData Object to hexadecimal, and returns it as a string.

## toBase64

*String* **toBase64**()

Converts the ScriptData Object to Base64, and returns it as a string.

## toUtf8

*String* **toUtf8()**

Converts the ScriptData Object to Utf8, and returns it as a string.

## toString

*QScriptValue* **toString()**

Converts the ScriptData Object to a string, and returns it as a string.

## toInt

*Quint64* **toInt()**

Converts the ScriptData Object to an integer, and returns it as *quint64*.

## toUInt

*Quint64* **toUInt()**

Converts the ScriptData Object to a UInt, and returns it as *quint64*.

## toBool

*QScriptValue* **toBool()**

Converts the ScriptData Object to bool, and returns it as a ScriptData Object.

## toByteArray

*QByteArray* **toByteArray()**

Converts the ScriptData Object to a ByteArray, and returns it as a QByteArray.

## toVariant

*QVariant* **toVariant()**

Converts the ScriptData Object to a Variant, and returns it as a QVariant.

## fromHex

*QScriptValue* **fromHex(string hex)**

Converts the provided **hex** string to a hexadecimal value and returns it as a ScriptData Object.

## fromBase64

*QScriptValue* **fromBase64(string base64)**

Converts the provided **base64** string to a Base64 value and returns it as a ScriptData Object.

## fromString

*QScriptValue* **fromString(string string)**

Converts the provided **string** string to a string value and returns it as a ScriptData Object.

## replace

```
void replace(QScriptValue before, QScriptValue after);
```

Replaces the value of the ScriptData Object specified in the parameter **before**, with the value of the ScriptData Object specified in the parameter **after**.

## writeToFile

```
bool writeToFile (string fileName)
```

Writes the ScriptData Object value to the file specified by **fileName**.

Note:

The **filename** string must contain the full path to the file to be written.







# Modules

# Modules

Modules are the main building blocks of any cardPresso solution. We use the term 'solution' because each situation calls for a different solution, and in that sense cardPresso 2.0 was envisioned as a set of building blocks developed to be completely modular so they can be adapted to any situation.

Being modular means that the user gets to decide which parts of the program are needed and should be used, and which parts are not.

cardPresso 2.0 is built of several modules that work independently of each other. A specific module can be called to perform a specific set of functions, as opposed to loading the entire program when your working situation only requires a specific operation, such as printing or encoding.

## 4.1.Application

The **Application** module is the base of all the major application related objects. It derives from the [QExModule](#) and inherits all it's methods.

The **Application** module provides the methods that allow us to create and customize the foundations upon which the application will be built.

Returns:	Method:
<i>Bool</i>	<a href="#">addModule(QObject module)</a>
<i>Bool</i>	<a href="#">addModule(QExModule module)</a>
<i>Int</i>	<a href="#">exec()</a>
<i>Void</i>	<a href="#">quit()</a>
<i>Void</i>	<a href="#">log(int logLevel,String message)</a>
<i>Void</i>	<a href="#">debug(String message)</a>
<i>Void</i>	<a href="#">print(String message)</a>
<i>Int</i>	<a href="#">showMessage(String errorMessage, String Buttons)</a>
<i>Void</i>	<a href="#">showError(String error)</a>
<i>QScriptValue</i>	<a href="#">runScript(String filename)</a>
<i>Bool</i>	<a href="#">runScripts()</a>
<i>QScriptValue</i>	<a href="#">runScriptWizard(String fileName,String requiredOption)</a>
<i>Void</i>	<a href="#">loadCSS(QScriptValue styleSheet)</a>
<i>Void</i>	<a href="#">loadCSSFolder(String styleSheetFolder)</a>
<i>Bool</i>	<a href="#">loadActions(String Folder)</a>
<i>String</i>	<a href="#">applicationPath()</a>
<i>String</i>	<a href="#">applicationFilePath()</a>

### 4.1.1.addModule

*Bool* **addModule(QObject module)**

Required params: module.

Loads the module from an existing QObject object providing access to all the methods from the respective module.

Returns 'true' if the operation was successful and 'false' if the module was unsuccessfully loaded.

### 4.1.2.addModule

*Bool* **addModule(QExModule module)**

Required params: module.

Loads the module from an existing QExModule object providing access to all the methods from the respective module.

Returns 'true' if the operation was successful and 'false' if the module was unsuccessfully loaded.

### 4.1.3.exec

*int* **exec()**

Required params: none.

Runs the application with the currently configured settings.

### 4.1.4.quit

*int* **quit()**

Required params: none.

Quits the application.

### 4.1.5.log

*Void* **log(int logLevel, String message)**

Required params: logLeve, message.

Logs the specified user **message** in the specified **logLevel**.

`app.log(1,"Test");`,Error:TypeError: Result of expression 'app.log' [QExEventLog(name = "")] is not a function.,  
Line:1

### 4.1.6.debug

*Void* **debug(String message)**

Required params: none.

Logs the user specified **message** to the output window.

### 4.1.7.print

*Void* **print(String message)**

Required params: none.

Prints the user specified **message** to the output window.

### 4.1.8.showMessage

*Int* **showMessage(String errorMessage, String buttons =QString?)**

Required params: errorMessage.

Displays a pop-up alert window with the user specified **errorMessage** and specified **buttons**.

Multiple buttons can be defined using the " | " separator in the string.

The **buttons** parameter supports the following standard buttons:

**PENDING TESTING**

Ok

Open

Save

Cancel

Close

Discard

Apply

Reset

RestoreDefaults

Help

SaveAll

Yes

YesToAll

No

NoToAll

Abort

Retry

Ignore

NoButton

### 4.1.9.showError

*Void* showError(*String* error)

Required params: error.

Shows a pop-up error message with the user specified **error** message.

### 4.1.10.runScript

*QScriptValue* runScript(*String* filename)

Required params: filename.

Runs the script file specified in the **filename** parameter.

### 4.1.11.runScripts

*Bool* runScripts()

Required params: none.

Runs the scripts in the -----  
**PROPERTY?**

### 4.1.12.runScriptWizard

*QScriptValue* runScriptWizard(*String* fileName,*String* requiredOption)

Required params: filename, requiredOption.

Runs the Script Wizard file specified by the **fileName** parameter and provides the required option in the **requiredOption** parameter.

### 4.1.13.loadCSS

*Void* loadCSS(*QScriptValue* styleSheet)

Required params: styleSheet.

Loads the CSS file specified by the **styleSheet** parameter.

### 4.1.14.loadCSSFolder

```
Void loadCSSFolder(String styleSheetFolder)
```

Required params: styleSheetFolder.

Loads the CSS files contained in the folder specified by the **styleSheetFolder** parameter.

### 4.1.15.loadActions

```
Bool loadActions(String folder)
```

Required params: styleSheet.

Loads the actions files contained in the folder specified by the **folder** parameter.

### 4.1.16.applicationPath

```
String applicationPath()
```

Required params: styleSheet.

Returns the application path as a string.

### 4.1.17.applicationFilePath

```
String applicationFilePath()
```

Required params: styleSheet.

Returns the application executable file path as a string.

## 4.2. User Interface

The **UI** module is the base module for all User Interface related objects. It derives from the [QExModule](#) and inherits all its methods.

The **UI** module provides the methods that allow to instantiate a window or a widget, which can then be used to build the user interface elements of the application.

To access the UI module methods the UI module must be loaded with the `addModule` method.

The following methods are available in the UI module:

**Returns:**

*QScriptValue*

*QScriptValue*

**Method:**

`createWidget(String widgetFile)`

`createDialog(String types)`

## 4.3.Security

### 4.3.1.login

Bool **login**(String **key**,QScriptValue **data**)

Required params: "key", "data".

Authenticates the current MIFARE Classic Tag with a KeyType and a KeyData. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

### 4.3.2.isValid

Bool **isValid**()

Returns "true" if the current Tag is successfully authenticated and returns "false" if the current is not authenticated.

## 4.4.cPEncoding encoding Object Reference:

The **cpEncoding** Object inherits from the [QExModule](#) and is the base of all encoding related objects, methods and properties, allowing the managing of both encoders and tags.

The **cpEncoding Module** provides the methods that allow us to instantiate an encoder as a [cpEncoder](#) Object, which will then be used to interact with both encoders and tags.

To instantiate an encoder as a [cpEncoder](#) Object is to create a connection between the cardPresso software and the physical encoder (or encoders) being used. This will in turn allow us to send and receive commands and data from the encoder trough cardPresso.

The following methods are available to create a [cpEncoder](#) instance:

#### Returns:

*cpEncoder*

*cpEncoderList*

#### Method:

[encoder](#)(String **encodername**)

[encoders](#)(String **types**)

#### Note:

By default, cardPresso creates an instance of the selected encoder as a [cpEncoder](#) Object with the name 'encoder'.



## 4.4.1.Methods

### encoder

*cpEncoder* **encoder**(*String* **encodername**)

Required params: encodername

Creates an instance of a single encoder, identified by the exact name of the encoder (**encodername**) specified by the user and provides access to the [cpEncoder](#) methods and properties.

Important:

The **encodername** parameter must be an exact match to the name of the encoder.

See also: [cpEncoder](#)

### encoders

*cpEncoderList* **encoders**(*String* **types**)

Required params: types

Enumerates and returns a cpEncoderList of the currently available encoders, of the specified **types**. A cpEncoderList Object contains an enumeration of all the detected encoders as [cpEncoder](#) Objects.

Supported **types** are:

- "CONTACT"
- "CONTACTLESS"

Each detected encoder is instantiated as a [cpEncoder](#) object and inherits all the respective properties and methods.

Each [cpEncoder](#) object can be retrieved from the cpEncoderList and iterated to gain access to the respective properties and methods:

See also: [cpEncoder](#)

## 4.4.2.cpEncoder encoder Object Reference:

The **cpEncoder** Object derives from the [cpEncoding](#) Object and is a virtual representation of a physical encoder. This is used by cardPresso to interact with the encoder.

Every method or property used by a **cpEncoder** Object will reflect on the physical encoder it represents.

Provided are several properties and methods which are used to either get information specific to the encoder or perform operations with the encoder itself, which includes interacting with Tags.

The **cpEncoder Module** also provides the methods that allow us to instantiate a tag as a [cpEncodeTag](#) Object, which will then be used to perform operations with tags.

### Properties:

The **cpEncoder** Object properties provide several methods which return general information specific to the encoder.

#### Returns:

*String*

*String*

*String*

*QObject*

#### Property:

[name](#)

[vendor](#)

[model](#)

[apis](#)

### Methods:

The **cpEncoder** Object methods allow us to validate if the encoder is correctly instantiated and also allows us to instantiate a tag as a [cpEncodeTag](#) object.

#### Returns:

Bool

EncodeTagList

EncodeTag

#### Method:

[isValid\(\)](#)

[tagsInRange](#)(String **type**)

[waitForTag](#)(String **type**)

## Properties

### **name**

String **name**

Returns the encoder name as a String.

### **vendor**

String **vendor**

Returns the encoder vendor name as a String.

### **model**

String **model**

Returns the encoder model name as a String.

### **apis**

QObject **apis**

The api provides direct access to the encoder manufacturer API methods .

## Methods

### isValid

#### Bool isValid()

Required params: none.

Returns (Bool) True if the encoder is correctly instantiated or false if the encoder was not found.

### waitForTag

#### EncodeTag waitForTag(*String type*)

Required params: *type*

Returns and instantiates the first encoder tag object of the specified **type** detected by the encoder and provides access to the [EncodeTag Object](#) methods and properties.

#### Note:

By default, cardPresso will wait 30 seconds for a valid tag to be placed in the encoder.

If no valid tag is placed in time, a timeout error will occur.

A different timeout value can be configured for the Encoding module with the [setProperty](#) method.

#### Important:

Valid tag **type** values depend on the encoder in use. Ex: "Q5", "MIFARE.1K"...

If no tag is detected by the encoder in the configured tag detection time (see: [setProperty](#)) the script will be interrupted and a timeout error will be returned.

The default tag detection time value is 30000 (30 seconds) by default.

## tagsInRange

EncodeTagList **tagsInRange**(*String* type)

Required params: type

Instantiates and returns a list of tags of the specified **type**, detected by the current [Encoder](#) as an EncodeTagList Object.

This instantiates each detected Tag and returns them enumerated in an EncodeTagList Object.

Each tag is instantiated as an [cpEncodeTag](#) Object and inherits all the respective properties and methods.

A specific [cpEncodeTag](#) Object can be retrieved from the EncodeTagList and iterated to gain access to the respective properties and methods.

Note:

The [setProperty](#) method can be used to configure the tag detection time in milliseconds.

The [setProperty](#) method can be used to detect and ignore duplicate cards (by UID).

See also: [cpEncodeTag](#) Object.

See also: [cpEncodeTag](#) Object.

Important:

Valid tag **type** values are encoder specific. Ex: "Q5", "MIFARE.1K"...

If no tag **type** is specified, all tag types detected by the encoder will be enumerated in the EncodeTagList object.

### 4.4.3.cpEncodeTag tag Object Reference:

A Tag Object derives from the [cpEncoder](#) Object. It's a representation of a Tag successfully detected by the encoder/reader and instantiated by cardPresso.

It provides several properties and methods that allow access and manipulation of the data stored in the Tag.

#### Properties:

The **cpEncodeTag** Object properties provide several methods which to make use of all the features each Tag supports.

##### Returns:

*String*

*String*

*QObject*

*QObject*

*QObject*

*QObject*

##### Property:

[.type](#)

[.description](#)

[.uid](#)

[.security](#)

[.data](#)

[.config](#)

#### Methods:

##### Returns:

Bool

##### Method:

[.isValid\(\)](#)

## Generic Tag Properties

### type

*String* type

Returns the Tag Type as a String.

### description

*String* description

Returns the Tag Description as a String.

### uid

isValid

*Bool* .uid.isValid()

Validades if the **.uid** property is implemented. Returns 'true' if it is implemented and 'false' if it's not implemented.

read

*DataBlock* read()

Returns the Tag UID as a [QScriptValue](#) object.

Note:

Read methods contain an **.error** property which will return 'undefined' if the read was successful or an error value if there was an error during the read process.

### security

isValid

*Bool* .security.isValid()

Validades if the **.security** property is implemented. Returns 'true' if it is implemented and 'false' if it's not implemented.

login

*Bool* .security.login(*DataBlock* data)

Required params: "data".

Authenticates the current Tag with a user specified **data** value. Returns (Bool) 'True' if successful or 'False' if

unsuccessful.

In the following example we will authenticate a MIFARE Classic Tag and if successfully authenticated we will log authentication result:

## data

isValid

*Bool* **.data.isValid()**

Validades if the **.data** property is implemented. Returns 'true' if it is implemented and 'false' if it's not implemented.

size

*Int* **.data.size()**

Required params: none.

Retrieves the total size of the current Tag as an integer value.

In the following example we will compare the size of the data we want to write with the size of the Tag. If the data is smaller the the Tag size, we will then write the data:

read

*QScriptValue* **.data.read(Int offset,Int size)**

Required params: offset, size.

Retrieves the data of a Tag in the specified **offset** and with the specified **size**. The sector and block are automatically calculated. Returns a QScriptValue Object which inherits the respective properties and methods.

### Note:

By default, if not set trough the parameters, the offset is 0 and the size is -1:

ex: myReadData = tag.data.read(); // default offset=0, default size=-1

If only the offset is set, the size defaults to -1:

ex: myReadData = tag.data.read(2); // from offset 2. size defaults to -1

To set a size, an offset must be set as well:

ex: myReadData = tag.data.read(2,4); // from offset 2. size 4.

Read methods contain an **.error** property which will return 'undefined' if the read was successfull or an error value if there was an error during the read process.

write

*Bool* **.data.write(Int offset,QScriptValue data)**

Required params: offset, data.

Writes the **data** defined by the user, in the specified **offset** in the tag as a QScriptValue. Returns (Bool) 'True' if successful or 'False' if unsuccessful.



## config

isValid

*Bool* **.config.isValid()**

Validates if the **.uid** property is implemented. Returns 'true' if it is implemented and 'false' if it's not implemented.

size

*Int* **.config.size()**

Required params: none.

Retrieves the total size of the Tag Configuration Memory.

read

*QScriptValue* **.config.read(Int offset, Int size)**

Available params: offset, size.

Retrieves the data from the Tag Configuration Memory in the specified **offset** and with the specified **size**. Returns a QScriptValue Object which inherits the respective properties and methods.

Note:

Read methods contain an **.error** property which will return 'undefined' if the read was successful or an error value if there was an error during the read process.

write

*Bool* **.config.write(Int offset, QScriptValue data)**

Required params: offset, data.

Writes the **data** defined by the user as a QScriptValue Object, in the specified **offset** in the Tag Configuration Memory. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

## Generic Tag Methods

### isValid

*Bool* .isValid()

Required params: none.

Returns (Bool) True if the Tag is correctly instantiated or false if no Tag was found.

## Generic Tag Sample Script

```
//return a list of the currently available contactless encoders
encoders = encoding.encoders("CONTACTLESS");

// for each of the detected encoders, run the following script:
for (var iEncoder in encoders) {

//return the current encoder and log it
  encoder = encoders[iEncoder];
  log("Detected Encoder : " + encoder.name); //log the Encoder name

//detect the Tag available in the encoder and log it
  tag = encoder.waitForTag();
  log("Detected Tag : " + tag.type); //log the Tag name

//login the MIFARE Classic tag and verify if login was successful
  //first we return an authentication attempt so we can validate if it was successful.
  tagSecurityLogin = tag.security.login("A",data.fromHex("FFFFFFFFFFFF"));

  //now we check if the authentication was successful. this means the code within the following if condition will
  only run if successfully authenticated
  if (authenticated = "True") {

    //IMPORTANT: Failing to authenticate a Tag several times will result in locking the Tag permanently,
    rendering it useless.

    //Now that we're in the section of the script that will run if the authentication is successful, we have more
    Tag operations available.
    //We will first write some sample data onto the Tag main memory and log it.

    //write sample data in the Tag main memory, from the offset (bit) 32
    myWriteData = tag.data.write(32, "Some sample data");

    //Now that we have successfully written data onto the Tag we will attempt to read it from the Tag and log it
    as well.

    //read the data to confirm it it was correctly saved in the Tag and if we can read from the Tag.
    myReadData = tag.data.read(32, 16);
  }
}
```

## 4.4.4. Contactless Tag Types

### MIFARE®

MIFARE Classic and MIFARE DESFire are registered trademarks of NXP B.V. and are used under license.

#### MIFARE Classic®

Property:	Returns:	Method:
UID	QScriptValue	<a href="#">read()</a>
UID	Bool	<a href="#">isValid()</a>
Security	Bool	<a href="#">login(String key, QScriptValue data)</a>
Security	Bool	<a href="#">isValid()</a>
Data	Int	<a href="#">size()</a>
Data	QScriptValue	<a href="#">readBlock(Int sector, Int block)</a>
Data	Bool	<a href="#">writeBlock(Int sector, Int block, String data)</a>
Data	QScriptValue	<a href="#">read(Int offset, Int size)</a>
Data	Bool	<a href="#">write(Int offset, String data)</a>
Data	Bool	<a href="#">formatMAD(String keyB, String keyA, Bool allSectors, Int madVersion)</a>
Data	Bool	<a href="#">isMADConfigured (Int madVersion)</a>
Data		<a href="#">getMADMULTIApplication(Int madVersion)</a>
Data		<a href="#">setMADMULTIApplication(String keyB, Bool multiApp, Int madVersion)</a>
Data		<a href="#">getMADCARDHOLDERINFO(Int madVersion)</a>
Data		<a href="#">setMADCARDHOLDERINFO(String keyB, Int sector, String givenName, String surname, String sex, String anyOtherInfo, Int madVersion)</a>
Data		<a href="#">removeMADCARDPUBLISHERINFO(String keyB, Int madVersion)</a>
Data		<a href="#">getMADCARDPUBLISHERINFO(Int madVersion)</a>
Data		<a href="#">setMADCARDPUBLISHERINFO(String keyB, Int sector, String publisherInfo, Int madVersion)</a>
Data		<a href="#">getMADAPPLICATIONIDENTIFIER(Int sector, Int madVersion)</a>
Data		<a href="#">setMADAPPLICATIONIDENTIFIER(String keyB, Int sector, String applicationCode, String functionClusterCode, Int madVersion)</a>
Data		<a href="#">isMADSECTORFREE(Int sector, Int madVersion)</a>
Config	Int	<a href="#">size()</a>
Config	QScriptValue	<a href="#">read(Int offset, Int size)</a>
Config	Bool	<a href="#">write(Int offset, String data)</a>

<i>Config</i>	<i>QScriptValue</i>	<a href="#">readTrailer</a> (Int sector)
<i>Config</i>	<i>Bool</i>	<a href="#">writeTrailer</a> (Int sector, String data)

UID

**Bool isValid()**

Returns "true" if the current UID property is implemented is valid and returns "false" if the current UID property is not implemented.

**String read()**

Required params: none.

Returns the Tag UID.

Data

**Int size()**

Required params: none.

Retrives the total size of the current MIFARE Classic Tag as an integer value.

**QScriptValue readBlock(Int sector,Int block)**

Required params: sector, block.

Retrieves the contents of a specific **block** in the MIFARE Classic Tag as a QScriptValue Object which inherits the respective properties and methods.

**Bool writeBlock(Int sector,Int block, QScriptValue data)**

Required params: sector, block, data.

Writes the **data** defined by the user in the specified **sector** and **block** in the MIFARE Classic Tag as a QScriptValue. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

**QScriptValue read(Int offset,Int size)**

Required params: offset, size.

Retrieves the data of a MIFARE Classic Tag in the specified **offset** and with the specified **size**. The sector and block are automatically calculated. Returns a QScriptValue Object which inherits the respective properties and methods.

Note:

Read methods contain an **.error** property which will return 'undefined' if the read was successful or an error value if there was an error during the read process.

**Bool write(Int offset,QScriptValue data)**

Required params: offset, data.

Writes the **data** defined by the user, in the specified **offset** in the MIFARE Classic Tag as a QScriptValue. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

**Bool formatMAD(String keyB, String keyA, Bool allSectors, Int madVersion);**

Required params: keyB.

Formats a MIFARE Classic card in the MAD standard. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

If no **keyA** value is provided the default **keyA** will be used.

If no **allSectors** value is provided the default value is 'true'.

If no **madVersion** value is provided, the default value will be '0' (zero).

**Bool isMadConfigured(Int madVersion);**

Required params: none;

Checks if a MIFARE Classic card is currently formatted in the MAD standard. Returns (Bool) 'True' if formatted or 'False' if not formatted.

If no **madVersion** value is provided, the default value will be '1' (one).

```
Bool getMADMultiApplication(Int madVersion);
```

Required params: keyB;

Returns if the mutiple application setting is enabled or disabled. Returns (Bool) 'True' if enabled or 'False' if not disabled.

If no **madVersion** value is provided, the default value will be '1' (one).

```
Bool setMADMultiApplication(String keyB, Bool multiApp, Int madVersion);
```

Required params: keyB;

Enables or disables multi application configuration for the MAD. Returns (Bool) 'True' if successful or 'False' if not successful.

If no **madVersion** value is provided, the default value will be '1' (one).

```
QScriptValue getMADCardHolderInfo(Int madVersion);
```

Required params: none;

Returns the current card holder info from the MAD in a QScriptData object with properties for each element of the card holder info:

The **surname** property.

The **givenName** property.

The **sex** property.

The **anyOtherInfo** property.

If no **madVersion** value is provided, the default value will be '1' (one).

```
QScriptValue setMADCardHolderInfo(String keyB, Int sector, String givenName, String surname, String sex, String anyOtherInfo, Int madVersion);
```

Required params: keyB, sector, givenName

Writes the card holder info in the MAD tag. Returns the written card holder info in a QScriptData with several properties for each element of the card holder info:

The **surname** property.

The **givenName** property.

The **sex** property.

The **anyOtherInfo** property.

If no **madVersion** value is provided, the default value will be '1' (one).

```
Bool removeMADCardPublisherInfo(String keyB, Int madVersion);
```

Required params: keyB;

Removes the data from the card publisher info. Returns (Bool) 'True' if successful or 'False' if not unsuccessful.

If no **madVersion** value is provided, the default value will be '1' (one).

```
Bool getMADCardPublisherInfo(Int madVersion);
```

Required params: none.

Returns the current publisher info stored in the Tag. Returns (Bool) 'True' if successful or 'False' if not

unsuccessful.

If no **madVersion** value is provided, the default value will be '1' (one).

```
Bool setMADCardPublisherInfo(String keyB, Int sector, String publisherInfo, Int madVersion);
```

Required params: keyB, sector, publisherInfo;

Writes the provided publisher info in the specified sector. Returns (Bool) 'True' if successful or 'False' if not unsuccessful.

If no **madVersion** value is provided, the default value will be '1' (one).

```
QScriptValue getMADApplicationIdentifier(Int sector, Int madVersion);
```

Required params: sector.

Returns a QScriptValue with the application identifier from the specified sector. The application identifier is composed of two properties: the **applicationCode** property and the **functionClusterCode** property.

If no **madVersion** value is provided, the default value will be '1' (one).

```
Bool setMADApplicationIdentifier(String keyB, Int sector, String applicationCode, String functionClusterCode, Int madVersion);
```

Required params: sector, applicationCode, functionClusterCode.

Writes the provided application identifier data in the specified sector. Returns (Bool) 'True' if successful or 'False' if not unsuccessful.

If no **madVersion** value is provided, the default value will be '1' (one).

```
Bool isMADSectorFree(Int sector, Int madVersion);
```

Required params: sector;

Returns if the specified MAD sector is free. Returns (Bool) 'True' if enabled or 'False' if not disabled.

If no **madVersion** value is provided, the default value will be '1' (one).

Config

```
Int size();
```

Required params: none.

Retrieves the total size of the MIFARE Classic Tag Configuration Sectors.

```
QScriptValue read(Int offset, Int size)
```

Available params: offset, size.

Retrieves the data from a MIFARE Classic Tag Configuration Block in the specified **offset** and with the specified **size**. The sector and block are automatically calculated. Returns a QScriptValue Object which inherits the respective properties and methods.

Note:

Read methods contain an **.error** property which will return 'undefined' if the read was successful or an error value if there was an error during the read process.

```
Bool write(Int offset, QScriptValue data)
```

Required params: offset, data.



Writes the **data** defined by the user as a QScriptValue Object, in the specified **offset** in the MIFARE Classic Tag Configuration Block. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

### *QScriptValue* **readTrailer(Int sector)**

Required params: sector.

Returns the Trailer data from the specified **sector** Trailer in the MIFARE Classic Tag as a QScriptValue Object, .

#### Note:

Read methods contain an **.error** property which will return 'undefined' if the read was successful or an error value if there was an error during the read process.

### *QScriptValue* **writeTrailer(Int sector,QScriptValue data)**

Required params: sector, data.

Writes the **data** defined by the user as a QScriptValue Object in the specified **sector** Trailer block in the MIFARE Classic Tag .

## apis.mifare Property

Provides methods to handle MIFARE Classic Tag API features.

The following MIFARE Classic Tag API methods are available:

### Returns:

*Int*

*Int*

*Bool*

*Int*

*Int*

*QScriptValue*

*Bool*

*QScriptValue*

*Bool*

*QScriptValue*

*QScriptValue*

### Method:

[tagSectors\(\)](#)

[blocksPerSector\(\*Int\* sector\)](#)

[login\(\*String\* key,\*QScriptValue\* data\)](#)

[dataSize\(\)](#)

[configSize\(\)](#)

[readSectorBlock\(\*Int\* sector,\*Int\* block\)](#)

[writeSectorBlock\(\*Int\* sector,\*Int\* block, \*QScriptValue\* data\)](#)

[read\(\*Int\* offset,\*Int\* size\)](#)

[write\(\*Int\* offset,\*QScriptValue\* data\)](#)

[readSectorTrailer\(\*Int\* sector\)](#)

[writeSectorTrailer\(\*Int\* sector,\*QScriptValue\* data\)](#)

**Int .apis.mifare.tagSectors()**

Required params: offset, size.

Returns the total amount of sectors the current MIFARE Classic Tag contains.

**Int .apis.mifare.blocksPerSector(Int sector)**

Required params: offset, size.

Returns the total amount of blocks a specified **sector** contains.

**Bool .apis.mifare.login(String key,QScriptValue data)**

Required params: "keyType", "keyData".

Authenticates the current MIFARE Classic Tag with the specified **Key** and **Data**. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

**Int .apis.mifare.dataSize()**

Required params: none.

Retrives the total size of the current MIFARE Classic Tag as an integer value.

**Int .apis.mifare.configSize()**

Required params: none.

Retrieves the total size of the MIFARE Classic Tag Configuration Sectors.

**QScriptValue .apis.readSectorBlock(Int sector,Int block)**

Required params: sector, block.

Retrieves the contents of a specific **block** in the MIFARE Classic Tag as a QScriptValue Object which inherits the respective properties and methods.

Note:

Read methods contain an **.error** property which will return 'undefined' if the read was successful or an error value if there was an error during the read process.

**Bool .apis.writeSectorBlock(Int sector,Int block, QScriptValue data)**

Required params: sector, block, data.

Writes the **data** defined by the user in the specified **sector** and **block** in the MIFARE Classic Tag as a *QScriptValue*. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

**QScriptValue .apis.read(Int offset,Int size)**

Required params: offset, size.

Retrieves the data of a MIFARE Classic Tag in the specified **offset** and with the specified **size**. The sector and block are automatically calculated. Returns a QScriptValue Object which inherits the respective properties and methods.

Note:

Read methods contain an **.error** property which will return 'undefined' if the read was successful or an error value if there was an error during the read process.

**Bool .apis.write(Int offset, QScriptValue data)**

Required params: offset, size.

Writes the **data** defined by the user, in the specified **offset** in the MIFARE Classic Tag as a *QScriptValue*. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

**QScriptValue .apis.readSectorTrailer(Int sector)**

Required params: offset, size.

Returns the Trailer data from the specified **sector** Trailer in the MIFARE Classic Tag as a QScriptValue Object, .  
Note:

Read methods contain an **.error** property which will return 'undefined' if the read was successful or an error value if there was an error during the read process.

**QScriptValue .apis.writeSectorTrailer(Int sector, QScriptValue data)**

Required params: offset, size.

Writes the **data** defined by the user as a QScriptValue Object in the specified **sector** Trailer block in the MIFARE Classic Tag .

## Mifare Classic MAD (API)

The MAD API provides methods to perform advanced operations on the MAD, specifically to manipulate [cpMifareMADSettings](#) objects.

The MAD API

Property	Returns:	Method:
<i>apis.mad</i>	<i>QScriptValue</i>	<a href="#">formatMAD</a>
<i>apis.mad</i>	<i>cpMifareMADSettings</i>	<a href="#">readMADSettings</a>
<i>apis.mad</i>	<i>NDEFMessage</i>	<a href="#">writeMADSettings</a>

```
Bool formatMAD(String keyB, String keyA, Bool allSectors, Int madVersion);
```

Required params: keyB.

Formats a MIFARE Classic card in the MAD format. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

If not **keyA** value is provided the default **keyA** will be used.

If no **allSectors** value is provided the default value is 'true'.

If no **madVersion** value is provided, the default value will be '0' (zero).

```
cpMifareMADSettings readMADSettings(Int madVersion);
```

Required params: keyB.

Returns a [cpMifareMADSettings](#) object which contains the current MAD settings and provides access to the [cpMifareMADSettings](#) object methods that allow changes the MAD settings, which can be saved to the MAD configuration sectors with the [writeMADSettings](#) method.

If no **madVersion** value is provided, the default value will be '1' (zero).

```
cpMifareMADSettings readMADSettings(Int madVersion);
```

Required params: keyB.

Writes a [cpMifareMADSettings](#) object to the MAD settings sectors.

If no **madVersion** value is provided, the default value will be '1' (zero).

The cpMifareMADSettings object contains the current MAD settings. Once instantiated, this object also provides access to methods that allow the manipulation of the current MAD settings.

The MAD settings can then be written to the tag, using the [writeMADSettings](#) method.

The cpMifareMADSettings object provides the following methods:

Returns:	Method:
<i>Int</i>	<a href="#">getMADVersion();</a>
<i>Bool</i>	<a href="#">getMultiApplication();</a>
<i>Bool</i>	<a href="#">setMultiApplication(Bool multiApp);</a>
<i>Bool</i>	<a href="#">getMADAvailable();</a>
<i>Bool</i>	<a href="#">setMADAvailable(Bool madAvailable);</a>
<i>QScriptData</i>	<a href="#">getCardHolderInfo();</a>
<i>Bool</i>	<a href="#">setCardHolderInfo(String keyB, Int sector, String givenName, String surname, String sex,</a>

```
String anyOtherInfo, Int madVersion);
```

```
Bool removeCardPublisherInfo();
```

```
QScriptData getCardPublisherInfo(Int madVersion);
```

```
Bool setCardPublisherInfo(Int sector, String publisherInfo);
```

```
Int getApplicationIdentifier(Int sector);
```

```
Bool setApplicationIdentifier(Int sector, Hex applicationCode, Hex functionClusterCode);
```

```
Bool isSectorFree(Int sector);
```

```
Bool getMADVersion();
```

Required params: none.

Returns the current MAD version configured in the tag info stored in the Tag.

```
Bool getMultiApplication();
```

Required params: none

Returns if the mutiple application setting is enabled or disabled. Returns (Bool) 'True' if enabled or 'False' if not disabled.

```
Bool setMultiApplication(Bool multiApp);
```

Required params: none.

Enables or disables multi application configuration for the MAD. Returns (Bool) 'True' if successful or 'False' if not successful.

```
Bool getMADAvailable();
```

Required params: none.

Returns if the MAD is available. Returns (Bool) 'True' if enabled or 'False' if not disabled.

```
Bool getMADAvailable(Bool madAvailable);
```

Required params: madAvailable

Sets if MAD is available or not available in the tag. Returns (Bool) 'True' if successful or 'False' if not successful.

```
QScriptValue getCardHolderInfo();
```

Required params: none;

Returns the current card holder info in a QScriptData object with properties for each element of the card holder info:

The **surname** property.

The **givenName** property.

The **sex** property.

The **anyOtherInfo** property.

```
QScriptValue setCardHolderInfo(Int sector, String givenName, String surname, String sex, String anyOtherInfo);
```

Required params: sector, givenName

Writes the card holder info in the MAD tag. Returns the current card holder info in a QScriptData with several properties for each element of teh card holder info:

```
Bool removeCardPublisherInfo();
```

Required params: none.

Removes the data from the card publisher info. Returns (Bool) 'True' if successful or 'False' if not unsuccessful.

```
Bool getCardPublisherInfo();
```

Required params: none.

Returns the current publisher info stored in the Tag.

```
Bool setCardPublisherInfo(Int sector, String publisherInfo);
```

Required params: sector, publisherInfo;

Writes the provided publisher info in the specified sector. Returns (Bool) 'True' if successful or 'False' if not unsuccessful.

```
QScriptValue getApplicationIdentifier(Int sector);
```

Required params: sector.

Returns a QScriptValue with the application identifier from the specified sector. The application identifier is composed of two properties: the **applicationCode** property and the **functionClusterCode** property.

```
Bool setApplicationIdentifier(Int sector, String applicationCode, String functionClusterCode);
```

Required params: sector, applicationCode, functionClusterCode.

Writes the provided application identifier data in the specified sector. Returns (Bool) 'True' if successful or 'False' if not unsuccessful.

```
Bool isSectorFree(Int sector);
```

Required params: keyB;

Returns if the specified MAD sector is free. Returns (Bool) 'True' if enabled or 'False' if not disabled.

```
MIFARE Classic Sample Script
```

```
//return a list of the currently available contact encoders  
encoders = encoding.encoders("Contactless");
```

```
// for each of the detected encoders, run the following script:  
for (var iEncoder in encoders) {
```

```
//return the current encoder and log it  
encoder = encoders[iEncoder];  
log("Detected Encoder : " + encoder.name); //log the Encoder name
```

```
//detect the Tag available in the encoder and log it  
tag = encoder.waitForTag();  
log("Detected Tag : " + tag.type); //log the Tag name
```

```
//login the MIFARE tag and verify if login was successful  
tagLogin = tag.security.login("A", data.fromHex("FFFFFFFFFFFF"));
```

```
if (!tagLogin = "True") {
```

```
//IMPORTANT: Failing to authenticate a Tag several times will result in locking the Tag permanently, rendering it useless.
```

```
//Write sample data in the tag Sector 2, Block 2 and log it.  
myWriteData = tag.data.writeBlock(2,2 , "Some sample data");  
log("Written data: " + myWriteData);
```

```
//read the data  
myReadData = tag.data.readBlock(2,2);
```

```
}
```

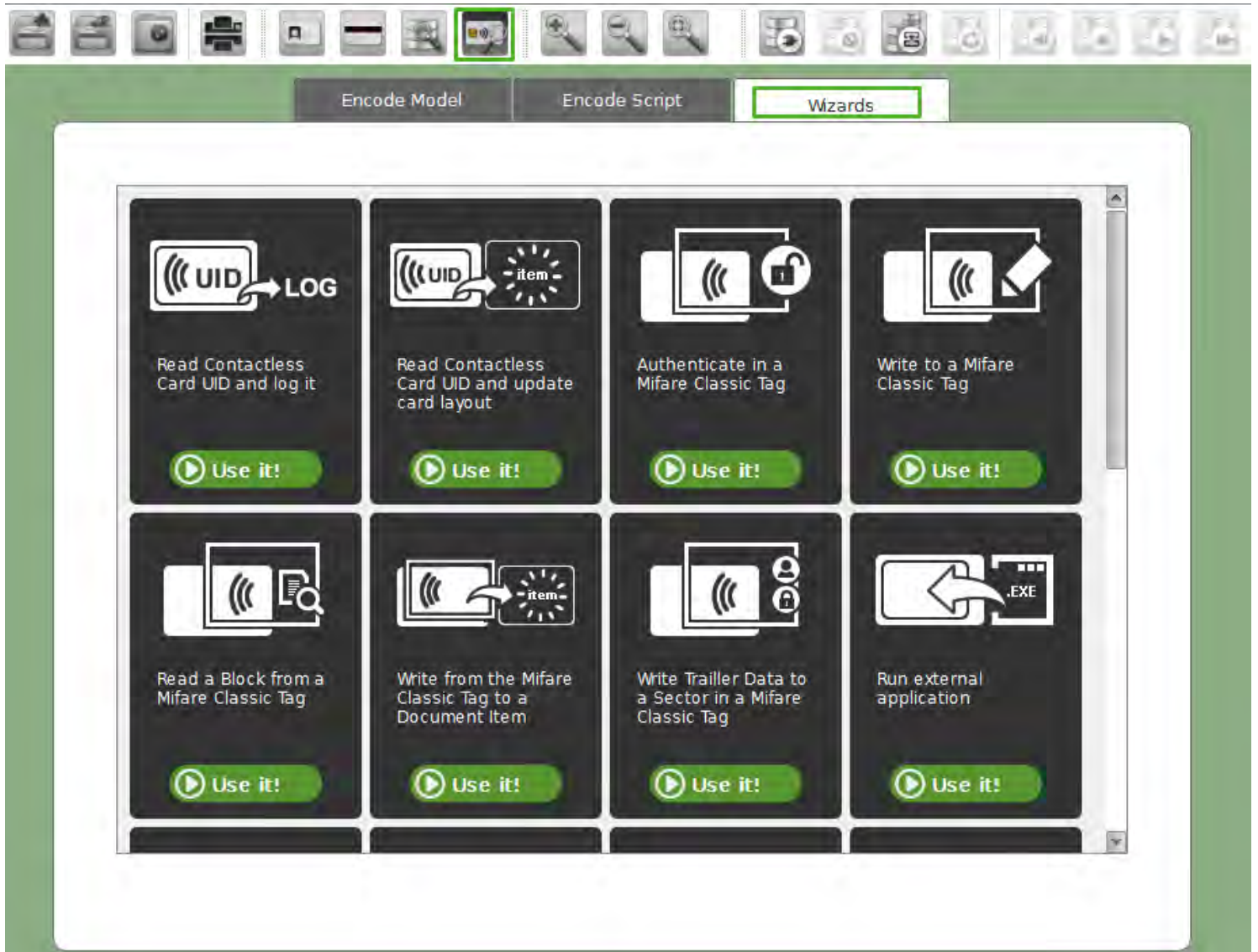
```
}
```



MIFARE Classic Wizards

cardPresso has several wizards available to help you to build your script.

To access to Wizards menu, please click in the **Encode View » Wizards** tab.



**MIFARE® DESFire®**

Property	Returns:	Method:
UID	Bool	<a href="#">isValid()</a>
UID	String	<a href="#">read()</a>
Security	Bool	<a href="#">login(Int keyNumber, String keyData);</a>
Security	Bool	<a href="#">loginISO(Int keyNumber, String keyData);</a>
Security	Bool	<a href="#">loginISO24(Int keyNumber, String keyData);</a>
Security	Bool	<a href="#">loginAES(Int keyNumber, String keyData);</a>
Security	Bool	<a href="#">changeKey(Int keyNumber, String newKeyData);</a>
Security	Bool	<a href="#">changeKeyISO(Int keyNumber, String newKeyData);</a>
Security	Bool	<a href="#">changeKeyISO24(Int keyNumber, String newKeyData);</a>
Security	Bool	<a href="#">changeKeyAES(Int keyNumber, String newKeyData);</a>

Security	ScriptValue	<a href="#">keyDiversificationAES</a> (String <b>masterKey</b> , Int <b>appld</b> =0,String <b>systemIdentifier</b> )
Security	ScriptValue	<a href="#">keyDiversificationAESAV1</a> (String <b>masterKey</b> , Int <b>appld</b> =0)
Security	ScriptValue	<a href="#">keyDiversificationAESAV2</a> (String <b>masterKey</b> , Int <b>appld</b> =0)
Security	ScriptValue	<a href="#">keyDiversification2TDEA</a> (String <b>masterKey</b> , Int <b>appld</b> =0,String <b>systemIdentifier</b> , Int <b>masterKeyVersion</b> )
Security	ScriptValue	<a href="#">keyDiversification3TDEA</a> (String <b>masterKey</b> , Int <b>appld</b> =0,String <b>systemIdentifier</b> , Int <b>masterKeyVersion</b> )
Data	Bool	<a href="#">isValid</a> ()
Data	Int	<a href="#">size</a> ()
Data	Bool	<a href="#">formatCard</a> ()
Data	Bool	<a href="#">createApplication</a> (Int <b>appld</b> , Int <b>numberOfKeys</b> , String <b>encryptionType</b> , Int <b>isold</b> , String <b>dfName</b> )
Data	Bool	<a href="#">deleteApplication</a> (Int <b>appld</b> )
Data	QScriptValue	<a href="#">getApplicationIDS</a> ()
Data	Int	<a href="#">getFreeMemory</a> ()
Data	QScriptValue	<a href="#">getDFNames</a> ()
Data	Bool	<a href="#">selectApplication</a> (Int <b>appld</b> )
Data	Bool	<a href="#">createFile</a> (Int <b>fileId</b> , String <b>commMode</b> , Int <b>readKey</b> ,Int <b>writeKey</b> , Int <b>readWriteKey</b> , Int <b>masterKey</b> , Int <b>fileSize</b> , Int <b>isoFileId</b> )
Data	Bool	<a href="#">createBackupFile</a> (Int <b>fileId</b> , Int <b>commMode</b> , Int <b>readKey</b> ,Int <b>writeKey</b> , Int <b>readWriteKey</b> , Int <b>masterKey</b> , Int <b>fileSize</b> , Int <b>isoFileId</b> )
Data	Bool	<a href="#">createValueFile</a> (Int <b>fileId</b> , String <b>commMode</b> , Int <b>readKey</b> ,Int <b>writeKey</b> , Int <b>readWriteKey</b> , Int <b>masterKey</b> ,Int <b>lowerLimit</b> , Int <b>upperLimit</b> , QScriptValue <b>initialValue</b> ,Int <b>limitedCreditEnabled</b> )
Data	Bool	<a href="#">createLinearRecordFile</a> (Int <b>fileId</b> , String <b>commMode</b> , Int <b>readKey</b> , Int <b>writeKey</b> , Int <b>readWriteKey</b> , Int <b>masterKey</b> , Int <b>recordSize</b> , Int <b>maxRecords</b> , Int <b>isoFileId</b> )
Data	Bool	<a href="#">createCyclicRecordFile</a> (Int <b>fileId</b> , String <b>commMode</b> , Int <b>readKey</b> , Int <b>writeKey</b> , Int <b>readWriteKey</b> , Int <b>masterKey</b> , Int <b>recordSize</b> , Int <b>maxRecords</b> , Int <b>isoFileId</b> )
Data	Bool	<a href="#">deleteFile</a> (Int <b>fileId</b> )
Data	QScriptValue	<a href="#">getFileIDs</a> ()
Data	QScriptValue	<a href="#">getISOFileIDs</a> ()
Data	Bool	<a href="#">write</a> (Int <b>fileId</b> , Int <b>offset</b> , String <b>data</b> )
Data	Bool	<a href="#">read</a> (Int <b>fileId</b> , Int <b>offset</b> , Int <b>maxLength</b> )
Data	QScriptValue	<a href="#">getValue</a> (Int <b>fileId</b> )
Data	Bool	<a href="#">credit</a> (Int <b>fileId</b> , Int <b>amount</b> )
Data	Bool	<a href="#">debit</a> (Int <b>fileId</b> , Int <b>amount</b> )
Data	Bool	<a href="#">limitedCredit</a> (Int <b>fileId</b> , Int <b>amount</b> )
Data	Bool	<a href="#">writeRecord</a> (Int <b>fileId</b> , Int <b>offset</b> , String <b>data</b> )

Data	Bool	<a href="#">readRecords</a> ( <i>Int fileId</i> , <i>Int fromRecord</i> , <i>Int maxRecordCount</i> , <i>Int recordSize</i> )
Data	Bool	<a href="#">clearRecordFile</a> ( <i>Int fileId</i> )
Data	Bool	<a href="#">commitTransaction</a> ()
Data	Bool	<a href="#">abortTransaction</a> ()
Data (NFC)	QScriptValue	<a href="#">formatNfc</a> ( <i>String masterKey</i> , <i>String nfcKey</i> )
Data (NFC)	QScriptValue	<a href="#">writeNfcText</a> ( <i>String nfcKey</i> , <i>String text</i> , <i>String locale</i> , <i>String encoding</i> )
Data (NFC)	QScriptValue	<a href="#">writeNfcUri</a> ( <i>String nfcKey</i> , <i>String uri</i> )
Data (NFC)	QScriptValue	<a href="#">writeNfcIcon</a> ( <i>String nfcKey</i> , <i>ScriptValue icon</i> )
Data (NFC)	QScriptValue	<a href="#">writeNfcSmartPoster</a> ( <i>String nfcKey</i> , <i>String title</i> , <i>String uri</i> , <i>String action</i> , <i>String locale</i> , <i>String encoding</i> )
Data (NFC)	Bool	<a href="#">prepareNfcVCard</a> ()
Data (NFC)	QScriptValue	<a href="#">addNfcVCardField</a> ( <i>String field</i> , <i>String data</i> )
Data (NFC)	QScriptValue	<a href="#">writeNfcVCard</a> ( <i>String nfcKey</i> )
Data (NFC)	QScriptValue	<a href="#">lockNfcApplication</a> ( <i>String nfcKey</i> )
Config	Bool	<a href="#">isValid</a> ()
Config	Bool	<a href="#">setConfiguration</a> ( <i>Byte option</i> , <i>String data</i> ) <b>WARNING: Use with caution.</b>
Config	QScriptValue	<a href="#">getKeySettings</a> ()
Config	Bool	<a href="#">changeKeySettings</a> ( <i>QScriptValue keySettings</i> )
Config	QScriptValue	<a href="#">getFileSettings</a> ( <i>Int fileId</i> )
Config	Bool	<a href="#">changeFileSettings</a> ( <i>Int fileId</i> , <i>String commMode</i> , <i>Int readKey</i> , <i>Int writeKey</i> , <i>Int readWriteKey</i> , <i>Int masterKey</i> )
Config	QScriptValue	<a href="#">getVersion</a> ()
Config	QScriptValue	<a href="#">getKeyVersion</a> ( <i>Int keyNumber</i> )

UID

**Bool isValid()**

Returns "true" if the current UID property is implemented is valid and returns "false" if the current UID property is not implemented.

**String read()**

Required params: none.

Returns the Tag UID if supported by the tag.

**Bool isValid()**

Returns "true" if the current Security property is implemented is valid and returns "false" if the current Security property is not implemented.

**Bool login(Int keyNumber, String keyData);**

Required params: keyNumber, keyData.

Authenticates in the currently selected application with the selected keyNumber and keyData. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

**Bool loginISO(Int keyNumber, String keyData);**

Required params: keyNumber, keyData.

Utilizes ISO authentication type to authenticate in the currently selected application with the selected keyNumber and keyData. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

**Bool loginISO24(Int keyNumber, String keyData);**

Required params: keyNumber, keyData.

Utilizes ISO24 authentication type to authenticate in the currently selected application with the selected keyNumber and keyData. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

**Bool loginAES(Int keyNumber, String keyData);**

Required params: keyNumber, keyData.

Utilizes AES authentication type to authenticate in the currently selected application with the selected keyNumber and keyData. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

**Bool changeKey(Int keyNumber, String keyData);**

Required params: keyNumber, keyData.

Changes the specified key of the currently selected application with the selected keyNumber and keyData. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

**Bool changeKeyISO(Int keyNumber, String keyData);**

Required params: keyNumber, keyData.

Changes the specified ISO key of the currently selected application with the selected keyNumber and keyData. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

**Bool changeKeyISO24(Int keyNumber, String keyData);**

Required params: keyNumber, keyData.

Changes the specified ISO24 key of the currently selected application with the selected keyNumber and keyData. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

**Bool changeKeyAES(Int keyNumber, String keyData, Int newkeyVersion);**

Required params: keyNumber, keyData.

Changes the specified AES key of the currently selected application with the selected keyNumber and keyData. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

Note:

If no **newkeyVersion** is specified, the default value will be "0".

*ScriptValue* **keyDiversificationAES**(String **masterKey**, Int **appld**,String **systemIdentifier**)

Required params: **masterKey**, **appld**, **systemIdentifier**.

Uses a plain AES algorithm to return a **16 bytes AES 128 bits diversified key** or a **24 bytes AES 192 bits diversified key**, depending on the provided **masterKey**.

The diversification input size can range from 1 to 31 bytes.

The provided **masterKey** must be a 16 bytes AES 128 bits or a 24 bytes AES 192 bits.

Note:

If no **appld** is specified, the default value used will be "0".

If no **systemIdentifier** is specified, the default value will be provided by cardPreso.

*ScriptValue* **keyDiversificationAESAV1**(String **masterKey**, Int **appld**)

Required params: **masterKey**, **appld**.

Uses a plain AES algorithm to return a **16 bytes AES 128 bits diversified key** as performed in MIFARE SAM AV1 based on the provided **masterKey** and **appld**.

The provided **masterKey** must be 16 bytes AES 128 bits.

*ScriptValue* **keyDiversificationAESAV2**(String **masterKey**, Int **appld**)

Required params: **masterKey**, **appld**.

Uses an AES 128 CMAC based algorithm to return a **16 bytes AES 128 bits diversified key** as performed in MIFARE SAM AV2 based on the provided **masterKey** and **appld**.

The provided **masterKey** must be 16 bytes AES 128 bits.

*ScriptValue* **keyDiversification2TDEA**(String **masterKey**, Int **appld**,String **systemIdentifier**, Int **masterKeyVersion**)

Required params: **masterKey**, **appld**, **systemIdentifier**.

Returns a **16 bytes 2TDEA diversified key**.

The diversification input size can range from 1 to 15 bytes.

The provided **masterKey** must be 16 bytes 2TDEA.

Note:

If no **appld** is specified, the default value used will be "0".

If no **systemIdentifier** is specified, the default value will be provided by cardPreso.

If no **masterKey** is specified, the default value used will be "0".

*ScriptValue* **keyDiversification3TDEA**(String **masterKey**, Int **appld**,String **systemIdentifier**, Int **masterKeyVersion**)

Required params: **masterKey**, **appld**, **systemIdentifier**.

Returns a **24 bytes 3TDEA diversified key**.

The diversification input size can range from 1 to 15 bytes.

The provided **masterKey** must be 24 bytes 3TDEA.

Note:

If no **appld** is specified, the default value used will be "0".

If no **systemIdentifier** is specified, the default value will be provided by cardPreso.

If no **masterKey** is specified, the default value used will be "0".

Data

### Bool isValid()

Returns "true" if the current Data property is implemented is valid and returns "false" if the current Data property is not implemented.

### Int size()

Required params: none.

Retrives the total size of the current DESFire Tag as an integer value.

### Bool formatCard()

Required params: none.

Resets the tag data memory to it's factory default removing all unprotected data from the tag data memory.

Note:

Must be authenticated with the master app key in application **0**.

### Bool createApplication(Int appld, Int numberOfKeys, String encryptionType, Int isold, String dfName)

Required params: appld, numberOfKeys.

Creates a new application in the current Tag with the specified app ID (**appld**) and number of keys (**numberOfKeys**).

Notes:

Each tag can have up to 28 applications.

If no **encryptionType** is specified, the application will have no encryption.

An application can have no keys if.

**isold** and **dfName** are optional.

If no **isold** is specified, the default value will be "0".

If no **dfName** is specified the default value will be an empty string.

### Bool deleteApplication(Int appld)

Required params: appld, numberOfKeys.

Deletes the application in the current Tag with the specified app ID (**appld**).

### QScriptValue getApplicationIDS()

Required params: , .

Retrieves a list of the IDs of the existing applications in the current tag.

Note:

Must be authenticated with the master app key in application "0".

### Int getFreeMemory()

Required params: none.

Retrieves the total size of the currently free Data memory in the tag as an integer value.

### *QScriptValue* **getDFNames()**

Required params: none.

Retrieves a list of the DF names of the existing application.

Note:

Must be authenticated with the master app key in application "0".

### *Bool* **selectApplication(Int appld)**

Required params: none.

Selects the application specified by the **appld**.

### *Bool* **createFile(Int fileId, String commMode, Int readKey, Int writeKey, Int readWriteKey, Int masterKey, Int fileSize, Int isoFileId)**

Required params: fileId, commMode, readKey, writeKey, readWriteKey, masterKey, fileSize.

Creates a new file in the currently selected and authenticated application.

Note:

If no **isoFileId** is specified, the default value will be "0".

Valid **commMode** values are:

"" = plain (empty)

"PLAIN" = Plain

"PLAINSECURED" = Maced

"FULLSECURED" = Enciphered

if no **commMode** is specified, Plain communication mode will be used.

### *Bool* **createBackupFile(Int fileId, String commMode, Int readKey, Int writeKey, Int readWriteKey, Int masterKey, Int fileSize, Int isoFileId)**

Required params: fileId, commMode, readKey, writeKey, readWriteKey, masterKey, fileSize.

Creates a new Backup file in the currently selected and authenticated application.

Note:

Backup File changes are only saved or aborted when the [commitTransaction](#) and [abortTransaction](#) methods are executed.

If no **isoFileId** is specified, the default value will be "0".

Valid **commMode** values are:

"" = plain (empty)

"PLAIN" = Plain

"PLAINSECURED" = Maced

"FULLSECURED" = Enciphered

if no **commMode** is specified, Plain communication mode will be used.

### *Bool* **createValueFile(Int fileId, String commMode, Int readKey, Int writeKey, Int readWriteKey, Int masterKey, Int lowerLimit, Int upperLimit, QScriptValue initialValue, Bool limitedCreditEnabled)**

Required params: fileId, commMode, readKey, writeKey, readWriteKey, masterKey, fileSize, lowerLimit, upperLimit, initialValue, limitedCreditEnabled.

Creates a new Value file in the currently selected and authenticated application.



Note:

**Cyclic Record File changes are only saved or aborted when the [commitTransaction](#) and [abortTransaction](#) methods are executed.**

The value for Limited Credit is limited to the sum of the debit commits, with the most recent transaction containing at least one debit.

Valid **commMode** values are:

"" = plain (empty)

"PLAIN" = Plain

"PLAINSECURED" = Maced

"FULLSECURED" = Enciphered

if no **commMode** is specified, Plain communication mode will be used.

```
Bool createLinearRecordFile(Int fileId, String commMode, Int readKey, Int writeKey, Int readWriteKey, Int masterKey, Int recordSize, Int maxRecords, Int isoFileId)
```

Required params: fileId, commMode, readKey, writeKey, readWriteKey, masterKey, recordSize, maxRecords.

Creates a new Linear Record file in the currently selected and authenticated application.

Note:

**Linear Record File changes are only saved or aborted when the [commitTransaction](#) and [abortTransaction](#) methods are executed.**

If no **isoFileId** is specified, the default value will be "0".

Valid **commMode** values are:

"" = plain (empty)

"PLAIN" = Plain

"PLAINSECURED" = Maced

"FULLSECURED" = Enciphered

if no **commMode** is specified, Plain communication mode will be used.

```
Bool createCyclicRecordFile(Int fileId, String commMode, Int readKey, Int writeKey, Int readWriteKey, Int masterKey, Int recordSize, Int maxRecords, Int isoFileId)
```

Required params: fileId, commMode, readKey, writeKey, readWriteKey, masterKey, recordSize, maxRecords.

Creates a new Cyclic Record file in the currently selected and authenticated application.

Note:

**Cyclic Record File changes are only saved or aborted when the [commitTransaction](#) and [abortTransaction](#) methods are executed.**

If no **isoFileId** is specified, the default value will be "0".

Valid **commMode** values are:

"" = plain (empty)

"PLAIN" = Plain

"PLAINSECURED" = Maced

"FULLSECURED" = Enciphered

if no **commMode** is specified, Plain communication mode will be used.

```
Bool deleteFile(Int fileId)
```

Required params: fileId.

Deletes the file specified by the **fileId** parameter in the currently authenticated application.

**QScriptValue getFileIDs()**

Required params: none.

Retrieves a list of existing file IDs in the currently authenticated application.

**QScriptValue getISOFileIDs()**

Required params: none.

Retrieves a list of ISO file IDs for the files in the currently authenticated application which have a specified ISO file ID.

**Bool write(Int fileId, Int offset, String data)**

Required params: fileId, offset, data.

Writes the **data** defined by the user, in the file specified by the **fileId** and the specified **offset** in the Tag. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

Note:

This method is specific to regular files created by the [createFile](#) and [createBackupFile](#) methods.

Backup File changes are only saved or aborted when the [commitTransaction](#) and [abortTransaction](#) methods are executed.

**QScriptData read(Int fileId, Int offset, Int maxLength)**

Required params: fileId, offset, maxLength.

Retrieves the data from the file specified by the **fileId** in the specified **offset** and with the specified **maxLength**. Returns a QScriptValue Object which inherits the respective properties and methods.

Note:

This method is specific to regular files created by the [createFile](#) and [createBackupFile](#) method.

Backup File changes are only saved or aborted when the [commitTransaction](#) and [abortTransaction](#) methods are executed.

Read methods contain an **.error** property which will return 'undefined' if the read was successful or an error value if there was an error during the read process.

**QScriptData getValue(Int fileId)**

Required params: fileId.

Retrieves the current value from the Value File specified by the **fileId**. Returns a QScriptValue Object which inherits the respective properties and methods.

Note:

This method is specific to Value files created by the [createValueFile](#) method.

**Bool credit(Int fileId, Int amount)**

Required params: fileId, amount.

Adds the value specified by the **amount** parameter to the Value File specified by the **fileId**. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

Note:

This method is specific to Value files created by the [createValueFile](#) method.

Read methods contain an **.error** property which will return 'undefined' if the read was successful or an error value if there was an error during the read process.

### *Bool* **debit**(*Int fileId, Int amount*)

Required params: fileId, amount.

Subtracts the value specified by the **amount** parameter to the Value File specified by the **fileId**. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

Note:

This method is specific to Value files created by the [createValueFile](#) method.

Read methods contain an **.error** property which will return 'undefined' if the read was successful or an error value if there was an error during the read process.

### *Bool* **limitedCredit**(*Int fileId, Int amount*)

Required params: fileId, amount.

Adds the value specified by the **amount** parameter to the Value File specified by the **fileId**. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

Note:

The value for Limited Credit is **limited to the sum of the debit commits**, with the most recent transaction containing at least one debit.

This method is specific to Value files created by the [createValueFile](#) method.

### *Bool* **writeRecord**(*Int fileId, Int offset, String data*)

Required params: fileId, offset, data.

Adds the **data** defined by the user, in the file specified by the **fileId** and the specified **offset** in the Record File. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

Note:

If no [commitTransaction](#) command is sent after a writeRecord command, the next writeRecord command to the same file writes to the already created record.

After sending a [commitTransaction](#) command, a new writeRecord command will create a new record in the record file. An [abortTransaction](#) command will invalidate all changes.

This method is specific to regular files created by the [createLinearRecordFile](#) and [createCyclicRecordFile](#) methods.

### *QScriptData* **readRecords**(*Int fileId, Int fromRecord, Int maxRecordCount, Int recordSize*)

Required params: fileId, fromRecord, maxRecordCount, recordSize.

Retrieves the **data**, in the file specified by the **fileId** and the specified **offset** in the Record File. Returns a QScriptData Object which inherits the respective properties and methods.

Note:

This method is specific to regular files created by the [createLinearRecordFile](#) and [createCyclicRecordFile](#) methods.

**Bool clearRecordFile(Int fileId)**

Required params: fileId.

Clears all the **data** in the file specified by the **fileId**. Returns a QScriptData Object which inherits the respective properties and methods. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

Note:

This method is specific to regular files created by the [createLinearRecordFile](#) and [createCyclicRecordFile](#) methods.

**Bool commitTransaction()**

Required params: none.

Saves the the **data** changes on Backup Files. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

Note:

This method is specific to regular files created by the [createBackupFile](#) methods.

**Bool abortTransaction()**

Required params: none.

Aborts the **data** changes on Backup Files. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

Note:

This method is specific to regular files created by the [createBackupFile](#) methods.

Data (NFC)

**Bool formatNfc(String masterKey,String nfcKey)**

Required params: masterKey.

Configures the MIFARE DESFire card to support NFC data.

Note: This command creates an NFC application where the NFC data will be saved.

**masterKey** refers to the PICC master key.

**nfcKey** refers to the key to be attributed to the NFC application

If no **nfcKey** is provided the PICC master key will be used.

**QScriptValue writeNfcText(String nfcKey, String text, String locale, String encoding)**

Required params: appKey, text.

Writes a plain text object with the user defined **text** to an NFC formatted MIFARE DESFire tag.

The **locale** parameter defines the language being used.

If no **locale** is provided the default locale will be "en-US"

The **encoding** parameter defines the character set being used.

If no **encoding** is provided the default locale will be "UTF8"

Supported charsets are **UTF8** and **UTF16**.

**QScriptValue writeNfcUri(String nfcKey, String uri)**

Required params: appKey, uri.

Writes a URI object with the user defined **uri** to an NFC formatted MIFARE DESFire tag.

### QScriptValue **writeNfcIcon**(*String nfcKey, QScriptValue icon*)

Required params: appKey, icon.

Writes an **icon** object (image) to an NFC formatted MIFARE DESFire tag.

### QScriptValue **writeNfcSmartPoster**(*String nfcKey, String title, String uri, String action, String locale, String encoding*)

Required params: appKey, icon.

Writes a smart poster object with the user defined **title** and **uri** to an NFC formatted MIFARE DESFire tag.

The **action** parameter suggests a course of action that the device (reader) should do with the content.

The following actions are supported:

**DO**: runs/opens the provided data (send an sms / open a browser / perform a telephone call).

**SAVE**: saves the provided data (store an sms / save the URI in a bookmark / save the telephone number in contacts).

**EDIT**: opens the provided data for editing (open an sms in an sms editor / open an URI in a URI editor / open the telephone number for editing).

Note: The action will only be executed if the device (reader) supports the provided data requested action.

Note:

The **locale** parameter defines the language being used.

If no **locale** is provided the default locale will be "en-US"

The **encoding** parameter defines the character set being used.

If no **encoding** is provided the default locale will be "UTF8"

Supported charsets are **UTF8** and **UTF16**.

### QScriptValue **prepareNfcVCard**()

Required params: none.

Creates a new vCard record with several commonly used vCard Fields ready for usage.

New vCard Fields can be added with the [addNfcVCardField](#) method.

After editing, the [writeNfcVCard](#) command must be used, to save the vCard data in the tag.

The default VCard fields created by the **prepareNfcVCard** method are:

first_name	middle_name	family_name	prefix_name	suffix_name	nickname	email
email_alternative	pager	fax	home_phone	cell_phone	business_phone	pro_Address1
pro_Address2	pro_Post_Code	pro_Region_State	pro_Town	pro_Country	address1	address2
post_Code	region_State	town	country	birthday	photo	role
title	company	note				

### QScriptValue **addNfcVCardField**(*String field, String data*)

Required params: field, data.

Adds a new field to an existing NFC vCard (see [prepareNfcVCard](#)).

The field is identified by the **field** parameter and will contain the data provided in the **data** parameter.

QScriptValue **writeNfcVCard**(*String nfcKey*)

Required params: nfcKey.

Saves the data from an existing NFC vCard object (see [prepareNfcVCard](#)) in the tag.

QScriptValue **lockNfcApplication**(*String nfcKey*)

Required params: nfcKey.

Locks the NFC Application, preventing further changes to the saved data.

Config

Bool **isValid**()

Returns "true" if the current Config property is implemented is valid and returns "false" if the current Data property is not implemented.

Bool **setConfiguration**(*Byte option, String data*)

Configures the specified tag **option** with the user specified configuration **data**. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

**WARNING: Using this command can make changes in card permanent and possibly render it useless. Use with caution.**

QScriptValue **getKeySettings**()

When authenticated in an application, retrieves a list of the existing keys and their current configuration.

Returns a QScriptValue with the following properties (case sensitive):

KeySettings

MaxNoOfkeys

Bool **changeKeySettings**(*QScriptValue keySettings*)

When authenticated in an application, allows the key settings to be modified with the user provided **keySettings**. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

QScriptValue **getFileSettings**(*Int fileId*)

When authenticated in an application, retrieves a QScriptValue wich contains all the settings for the key settings to be modified with the provided **keySettings**. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

Returns a QScriptValue with the following properties:

**FileType**

**CommMode**

**AccessRights**

**ReadKey**

**WriteKey**

**ReadWriteKey**

## MasterKey

**Backup** data files also contain the following properties:

**FileSize**

**Value Files** also contain the following properties:

**LowerLimit**

**UpperLimit**

**LimitedCreditValue**

**LimitedCreditEnabled**

**Linear Record** files and **Cyclic Record** files also contain the following properties

**RecordSize**

**MaxNumberOfRecords**

**CurrentNumberOfRecords**

```
Bool changeFileSettings(Int fileId, String commMode, Int readKey, Int writeKey, Int readWriteKey, Int masterKey)
```

When authenticated in an application, allows to change the settings in the available parameters. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

```
QScriptValue getVersion()
```

When authenticated in an application, returns a QScriptValue object with the MIFARE DESFire Software Version of the current tag (EV0/EV1).

The QScriptValue object also contains the the following properties:

**hwVendorID** (hardware vendor ID)

**hwType** (hardware type)

**hwSubType** (hardware sub type)

**hwMajorVersion** (hardware major version)

**hwMinorVersion** (hardware minor version)

**hwStorageSize** ( hardware storage size)

**hwProtocol** (hardware protocol)

**swVendorID** (software vendor ID)

**swType** (software type)

**swSubType** (software sub type)

**swMajorVersion** (software major version)

**swMinorVersion** (software minor version)

**swStorageSize** (software storage size)

**swProtocol** (software protocol)

**UID** (tag UID)

**batchNo** (batch number)

**CWProduction** (Production CW)

**YearProduction** (production year)

```
QScriptValue getKeyVersion(Int keyNumber)
```

When authenticated in an application, returns the current version of the key as a QScriptValue object.

Note:

Only applies to keys with AES encryption who have been previously changed ([changeKeyAES](#)) and attributed a key version (**newkeyVersion**).



## NFC MIFARE DESFire API

Property	Returns:	Method:
<i>apis.nfc</i>	<i>Bool</i>	<a href="#">SetNfcDESFireVersion</a> (String nfcDesfireVersion)
<i>apis.nfc</i>	<i>String</i>	<a href="#">GetNfcDESFireVersion</a> ()

**Bool SetNfcDESFireVersion**(String nfcDesfireVersion)

Required params: nfcDesfireVersion.

Configures the NFC DESFire version to the user specified version.

Valid versions are **EVO** and **EV1**.

**String GetNfcDESFireVersion**()

Required params: none.

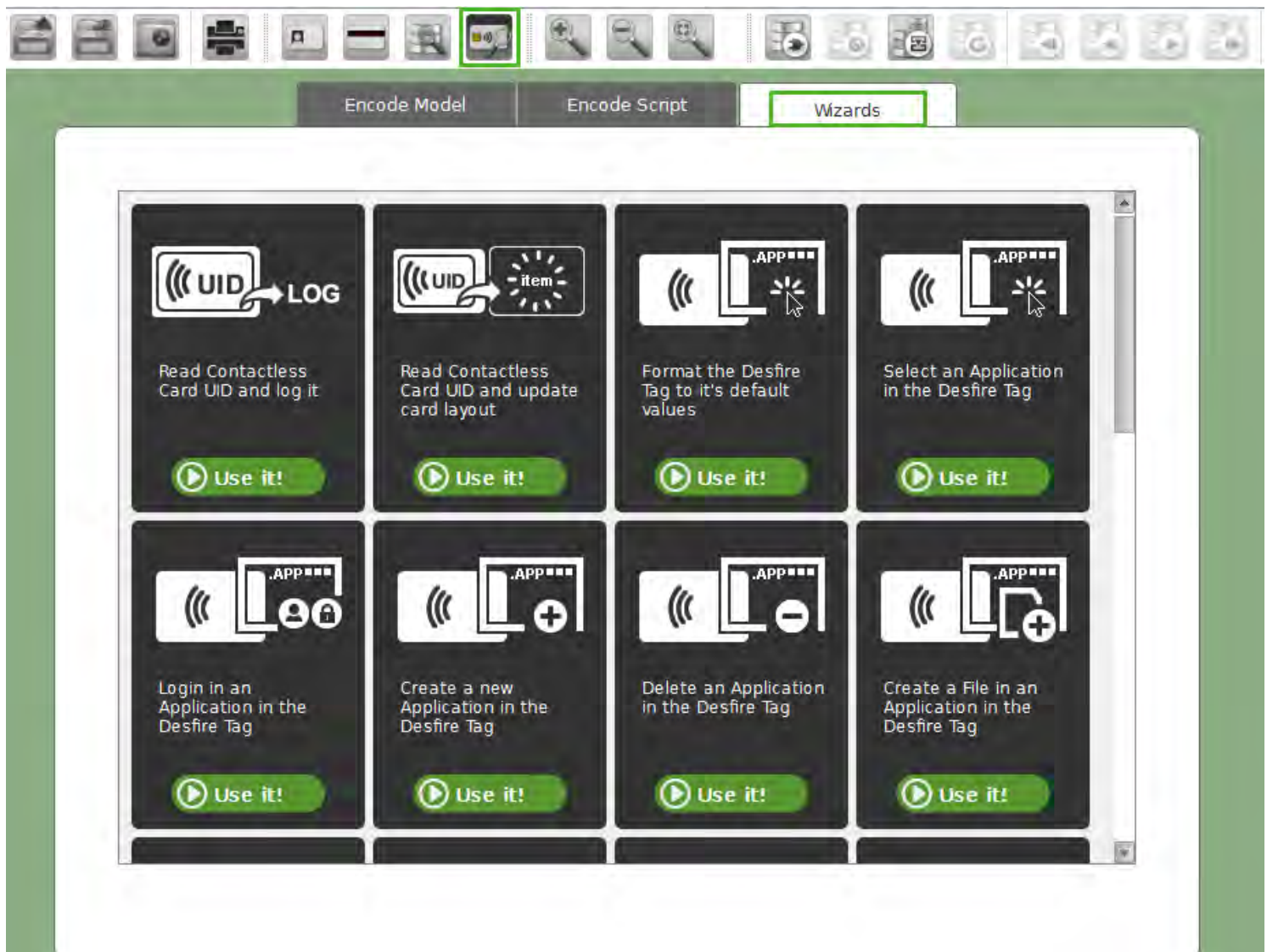
Returns the currently configured NFC DESFire version.

Valid versions are **EVO** and **EV1**.

## MIFARE DESFire Wizards

cardPresso has several wizards available to help you to build your script.

To access to Wizards menu, please click in the **Encode View » Wizards** tab.



## STMicroelectronics

## SRTAG-D

Property	Returns:	Method:
UID	Bool	<a href="#">isValid()</a>
UID	String	<a href="#">read()</a>
Security	Bool	<a href="#">isValid()</a>
Security	Bool	<a href="#">login</a> (String keyType, String keyData);
Security	Bool	<a href="#">loginRead</a> (String keyData);
Security	Bool	<a href="#">loginWrite</a> (String keyData);
Security	Bool	<a href="#">changeReadKey</a> (String oldReadKey, String newReadKey);
Security	Bool	<a href="#">changeWriteKey</a> (String oldWriteKey, String newWriteKey);

<i>Data</i>	<i>Bool</i>	<a href="#">isValid()</a>
<i>Data</i>	<i>Int</i>	<a href="#">size()</a>
<i>Data</i>	<i>Bool</i>	<a href="#">write</a> ( <i>Int fileId</i> , <i>Int offset</i> , <i>String data</i> )
<i>Data</i>	<i>QScriptValue</i>	<a href="#">read</a> ( <i>Int fileId</i> , <i>Int offset</i> , <i>Int maxLength</i> )
<i>Data (NFC)</i>	<i>QScriptValue</i>	<a href="#">formatNfc</a> ( <i>String writeKey</i> , <i>String readKey</i> )
<i>Data (NFC)</i>	<i>QScriptValue</i>	<a href="#">writeNfcText</a> ( <i>String writeKey</i> , <i>String text</i> , <i>String locale</i> , <i>String encoding</i> , <i>String readKey</i> )
<i>Data (NFC)</i>	<i>QScriptValue</i>	<a href="#">writeNfcUri</a> ( <i>String writeKey</i> , <i>String uri</i> , <i>String readKey</i> )
<i>Data (NFC)</i>	<i>QScriptValue</i>	<a href="#">writeNfcIcon</a> ( <i>String writeKey</i> , <i>String icon</i> , <i>String readKey</i> )
<i>Data (NFC)</i>	<i>QScriptValue</i>	<a href="#">writeNfcSmartPoster</a> ( <i>String writeKey</i> , <i>String title</i> , <i>String uri</i> , <i>String action</i> , <i>String locale</i> , <i>String encoding</i> , <i>String readKey</i> )
<i>Data (NFC)</i>	<i>Bool</i>	<a href="#">prepareNfcVCard</a> ()
<i>Data (NFC)</i>	<i>Bool</i>	<a href="#">addNfcVCardField</a> ( <i>String field</i> , <i>String data</i> )
<i>Data (NFC)</i>	<i>Bool</i>	<a href="#">writeNfcVCard</a> ( <i>String writeKey</i> , <i>String readKey</i> )
<i>Data (NFC)</i>	<i>Bool</i>	<a href="#">lockNfcApplication</a> ( <i>String writeKey</i> , <i>String readKey</i> )
<i>Data (NFC)</i>	<i>QScriptValue</i>	<a href="#">lockNfcApplicationForever</a> ( <i>String writeKey</i> , <i>String readKey</i> )
<i>Data (NFC)</i>	<i>QScriptValue</i>	<a href="#">unlockNfcApplication</a> ( <i>String writeKey</i> , <i>String readKey</i> )
<i>Config</i>	<i>Bool</i>	<a href="#">isValid()</a>

UID

**Bool isValid()**

Returns "true" if the current UID property is implemented is valid and returns "false" if the current UID property is not implemented.

**String read()**

Required params: none.

Returns the Tag UID.

Security

**Bool isValid()**

Returns "true" if the current Tag is successfully authenticated and returns "false" if the current is not authenticated.

**Bool login(String keyType,String keyData)**

Required params: "key", "data".

Authenticates the current Tag with the provided KeyType and a KeyData. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

**Bool loginRead(String keyData)**

Required params: "keyData".

Authenticates the current Tag with read only permissions with the provided KeyData. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

**Bool loginWrite(String keyData)**

Required params: keyData.

Authenticates the current Tag with read and write permissions with the provided KeyData. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

**Bool loginWrite(String oldReadKey, String newReadKey)**

Required params: oldReadKey, newReadKey.

Changes the current Read Key with the provided **newReadKey**. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

**Bool changeWriteKey(String oldReadKey, String newReadKey)**

Required params: oldReadKey, newReadKey.

Changes the current Write Key with the provided **newReadKey**. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

Data

**Bool isValid()**

Returns "true" if the current Data property is implemented is valid and returns "false" if the current Data property is not implemented.

**Int size()**

Required params: none.

Retrives the total size of the current Tag as an integer value.

**Bool write(Int fileId, Int offset, String data)**

Required params: fileId, offset, data.

Writes the **data** defined by the user, in the file specified by the **fileId** and the specified **offset** in the Tag. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

**QScriptData read(Int fileId, Int offset, Int maxLength)**

Required params: fileId, offset, maxLength.

Retrieves the data from the file specified by the **fileId** in the specified **offset** and with the specified **maxLength**. Returns a QScriptValue Object which inherits the respective properties and methods.

Note:

Read methods contain an **.error** property which will return 'undefined' if the read was successful or an error value if there was an error during the read process.

**Bool formatNfc(String writeKey, String readKey)**

Required params: writeKey, readKey.

Formats the Tag to NFC data.

**writeKey** refers to the key that allows writing in the tag.

**readKey** refers to the key that allows reading the tag

**QScriptValue writeNfcText(String writeKey, String text, String locale, String encoding, String readKey)**

Required params: writeKey, text, locale, encoding, readKey.

Writes a plain text object with the user defined **text** to an NFC formatted tag.

The **locale** parameter defines the language being used.

If no **locale** is provided the default locale will be "en-US"

The **encoding** parameter defines the character set being used.

If no **encoding** is provided the default locale will be "UTF8"

Supported charsets are **UTF8** and **UTF16**.

**QScriptValue writeNfcUri(String writeKey, String uri, String readKey)**

Required params: writeKey, uri, readKey.

Writes a URI object with the user defined **uri** to an NFC formatted tag.

**QScriptValue writeNfcIcon(String writeKey, String icon, String readKey)**

Required params: writeKey, icon, readKey.

Writes an **icon** object (image) to an NFC formatted tag.

**QScriptValue writeNfcSmartPoster(String writeKey, String title, String uri, String action, String locale, String encoding, String readKey)**

Required params: appKey, icon.

Writes a smart poster object with the user defined **title** and **uri** to an NFC formatted tag.

The **action** parameter suggests a course of action that the device (reader) should do with the content.

The following actions are supported:

**DO**: runs/opens the provided data (send an sms / open a browser / perform a telephone call).

**SAVE**: saves the provided data (store an sms / save the URI in a bookmark / save the telephone number in contacts).

**EDIT**: opens the provided data for editing (open an sms in an sms editor / open an URI in a URI editor / open the telephone number for editing).

Note: The action will only be executed if the device (reader) supports the provided data requested action.

Note:

The **locale** parameter defines the language being used.

If no **locale** is provided the default locale will be "en-US"

The **encoding** parameter defines the character set being used.

If no **encoding** is provided the default locale will be "UTF8"

Supported charsets are **UTF8** and **UTF16**.

### QScriptValue **prepareNfcVCard()**

Required params: none.

Creates a new vCard record with several commonly used vCard Fields ready for usage.

New vCard Fields can be added with the [addNfcVCardField](#) method.

After editing, the [writeNfcVCard](#) command must be used, to save the vCard data in the tag.

The default VCard fields created by the **prepareNfcVCard** method are:

first_name	middle_name	family_name	prefix_name	suffix_name	nickname	email
email_alternative	pager	fax	home_phone	cell_phone	business_phone	pro_Address1
pro_Address2	pro_Post_Code	pro_Region_State	pro_Town	pro_Country	address1	address2
post_Code	region_State	town	country	birthday	photo	role
title	company	note				

### QScriptValue **addNfcVCardField(String field, String data)**

Required params: field, data.

Adds a new field to an existing NFC vCard (see [prepareNfcVCard](#)).

The field is identified by the **field** parameter and will contain the data provided in the **data** parameter.

### QScriptValue **writeNfcVCard(String writeKey, String readKey)**

Required params: nfcKey.

Saves the data from an existing NFC vCard object (see [prepareNfcVCard](#)) in the tag.

### QScriptValue **lockNfcApplication(String writeKey, String readKey)**

Required params: nfcKey.

Locks the NFC Application, preventing further changes to the saved data until the application is unlocked with the [unlockNfcApplication](#) command.

**QScriptValue lockNfcApplication**(*String writeKey, String readKey*)

Required params: nfcKey.

Permanently locks the NFC Application, preventing further changes to the saved data.

**Bool lockNfcApplication**(*String writeKey, String readKey*)

Required params: nfcKey.

Unlocks a previously locked NFC Application, allowing further changes to the saved data.

Config

**Bool isValid()**

Returns "true" if the current Config property is implemented is valid and returns "false" if the current Data property is not implemented.

SRTAG-D API

**Bool Verify**(*String passId, String keyData*)

Required params: passId, keyData.

Verifies if

NFC APP EXISTS?

**Bool ChangeReferenceData**(*String passId, String keyData*)

Required params: passId, keyData.

Changes the reference data to the user provided **keyData**.**Bool DisableVerificationRequirement**(*String passId*)

Required params: passId.

Disables the verification requirements of the tag.

**Bool SelectApplication**(*String appId*)

Required params: appId.

Selects the specified application.

**Bool SelectFile**(*String fileId*)

Required params: fileId.

Selects the specified file within the currently selected application.

**Bool CapabilityContainerSelect**()

Required params: none.

Selects the NFC Capability Container file.

**Bool NDEFSelect**()

Required params: none.

Selects the NDEF message file.

**Bool SystemFileSelect**()

Required params: none.

Selects the NFC system file.

Bool **enablePermanentState**(*String* securityFlag)

Required params: fileId.

Enables permanent state for the current tag.



## Contactless APIs

### NFC

The NFC Tag API provides methods to perform advanced NFC data handling, specifically to manipulate NDEFMessage objects, where several NFC record objects (text, uri, icon, smart poster, vcard) can be saved. This allows for a more customizable data structure development when working with NFC formatted data.

Property	Returns:	Method:
<i>apis.nfc</i>	QScriptValue	<a href="#">formatNfc</a> (String <b>masterKey</b> ,String <b>nfcKey</b> )
<i>apis.nfc</i>	QScriptValue	<a href="#">lockNfcApplication</a> (String <b>nfcKey</b> )
<i>apis.nfc</i>	NDEFMessage	<a href="#">createNDEFMessage</a> ()
<i>apis.nfc</i>	QScriptValue	<a href="#">readNDEFMessage</a> (String <b>appKey</b> );
<i>apis.nfc</i>	QScriptValue	<a href="#">writeNDEFMessage</a> (String <b>appKey</b> ,cpNDEFMessage <b>ndefMsg</b> );
<i>apis.nfc</i>	NDEFTextRecord	<a href="#">createNfcTextRecord</a> (String <b>nfcKey</b> , String <b>text</b> , String <b>locale</b> , String <b>encoding</b> )
<i>apis.nfc</i>	NDEFUriRecord	<a href="#">createNfcUriRecord</a> (String <b>nfcKey</b> , String <b>uri</b> )
<i>apis.nfc</i>	NDEFIconRecord	<a href="#">createNfcIconRecord</a> (String <b>nfcKey</b> , QScriptValue <b>icon</b> )
<i>apis.nfc</i>	cpNDEFSmartPosterRecord	<a href="#">createNfcSmartPosterRecord</a> (String <b>nfcKey</b> , String <b>title</b> , String <b>uri</b> , String <b>action</b> , String <b>locale</b> , String <b>encoding</b> )
<i>apis.nfc</i>	cpNDEFvCardRecord	<a href="#">createNfcvCardRecord</a> ()

`formatNfc`

QScriptValue **formatNfc**(String **masterKey**,String **nfcKey**)

Required params: **masterKey**.

Configures the DESFire card to receive NFC data.

Note: This command creates an NFC application where the NFC data will be saved.

**masterKey** refers to the PICC master key.

**nfcKey** refers to the key of the NFC application

If no **nfcKey** is provided the PICC master key will be used.

`lockNfcApplication`

QScriptValue **lockNfcApplication**(String **nfcKey**)

Required params: **nfcKey**.

Locks the NFC Application, preventing further changes to the saved data.

`createNDEFMessage`

NDEFMessage **createNDEFMessage**()

Required params: none.

Creates a new NDEFMessage object where NFC records can be added.

The NDEFMessage object contains an **addRecord** command which can be used to store existing NFC records in the NDEFMessage object.

Void **addRecord**(nfcRecord recordObject)

readNDEFMessage

QScriptValue **readNDEFMessage**(String appKey);

Required params: none.

Reads the NDEFMessage currently saved in the NFC application.

writeNDEFMessage

QScriptValue **writeNDEFMessage**(String appKey, cpNDEFMessage ndefMsg);

Required params: none.

Writes the specified **ndefMsg** object in the NFC application.

createNfcTextRecord

NDEFTextRecord **createNfcTextRecord**(String nfcKey, String text, String locale, String encoding)

Required params: appKey, text.

Creates a plain text NFC record object with the user defined **text**.

This record file can be added to an existing NDEFMessage object using the [addRecord](#) method.

The **locale** parameter defines the language being used.  
If no **locale** is provided the default locale will be "en-US"

The **encoding** parameter defines the character set being used.  
If no **encoding** is provided the default locale will be "UTF8"  
Supported charsets are **UTF8** and **UTF16**.

createNfcUriRecord

NDEFUriRecord **createNfcUriRecord**(String nfcKey, String uri)

Required params: appKey, uri.

Creates a URI record object with the user defined **uri**.

This record file can be added to an existing NDEFMessage object using the [addRecord](#) method.

createNfcIconRecord

NDEFIconRecord **createNfcIconRecord**(String nfcKey, QScriptValue icon)

Required params: appKey, icon.

Creates an **icon** record object (image)..

This record file can be added to an existing NDEFMessage object using the [addRecord](#) method.

createNfcSmartPosterRecord

cpNDEFSmartPosterRecord **createNfcSmartPosterRecord**(String nfcKey, String title, String uri, String action, String locale, String encoding)

Required params: appKey, icon.

Creates a smart poster record object with the user defined **title** and **uri**.

This record file can be added to an existing NDEFMessage object using the [addRecord](#) method.

The **action** parameter suggests a course of action that the device (reader) should do with the content.

The following actions are supported:

**DO:** runs/opens the provided data (send an sms / open a browser / perform a telephone call).

**SAVE:** saves the provided data (store an sms / save the URI in a bookmark / save the telephone number in contacts).

**EDIT:** opens the provided data for editing (open an sms in an sms editor / open an URI in a URI editor / open the telephone number for editing).

Note: The action will only be executed if the device (reader) supports the provided data requested action.

Note:

The **locale** parameter defines the language being used.

If no **locale** is provided the default locale will be "en-US"

The **encoding** parameter defines the character set being used.

If no **encoding** is provided the default locale will be "UTF8"

Supported charsets are **UTF8** and **UTF16**.

```
createNfcvCardRecord
```

cpNDEFvCardRecord **createNfcvCardRecord()**

Required params: none.

Creates a new VCard record with several commonly used VCard Fields.

This record file can be added to an existing NDEFMessage object using the [addRecord](#) method.

New VCard Fields can be added with the [addNfcVCardField](#) method.

After editing, the [writeNfcVCard](#) command must be used, to save the VCard data in the tag.

The default VCard fields created by the **createNfcvCardRecord** method are:

first_name	middle_name	family_name	prefix_name	suffix_name	nickname	email
email_alternative	pager	fax	home_phone	cell_phone	business_phone	pro_Address1
pro_Address2	pro_Post_Code	pro_Region_State	pro_Town	pro_Country	address1	address2
post_Code	region_State	town	country	birthday	photo	role
title	company	note				

NFC Record Objects

NFC Record Objects provide access to methods that allow advanced operations with NFC Objects.

The Following NFC Objects are available:

#### Object:

<a href="#">NDEFTextRecord</a>	Holds and Handles NFC Text Record data
<a href="#">NDEFUriRecord</a>	Holds and Handles NFC Uri Record data
<a href="#">NDEFIconRecord</a>	Holds and Handles NFC Icon Record data
<a href="#">NDEFSmartPosterRec</a>	Holds and Handles NFC Smart Poster Record data

[ord](#)[NDEFvCardRecord](#)

Holds and Handles NFC Card Record data

NDEFTextRecord objects can be added to NDEFMessage objects ([createNDEFMessage](#)). They provide access to the methods which allow the user to store and manipulate plain text data in the record.

NDEFTextRecord objects support every [Common Record Methods](#) and the following methods:

**Returns:****Method:***String***getText()***Void***setText(String text)***String***getEncoding()***Void***setEncoding(String encoding)***Int***getLocale()***Void***setLocale(String locale)****ScriptValue getLocale()**

Required params: none.

Returns the locale of the Text record object.

**Bool setLocale(String locale)**

Required params: locale.

Returns the Type Name Format of the text record object.

**ScriptValue getText()**

Required params: none.

Returns the text of the Text record object.

**Bool setText(String text)**

Required params: text.

Sets the text value of the text record object.

**ScriptValue getEncoding()**

Required params: none.

Returns the encoding of the Text record object.

**Bool setEncoding(String encoding)**

Required params: encoding.

Sets the encoding of the text record object.

NDEFUriRecord objects can be added to NDEFMessage objects ([createNDEFMessage](#)). They provide access to the methods which allow the user to store and manipulate Uri data in the record.

NDEFUriRecord objects support every [Common Record Methods](#) and the following methods:

**Returns:****Method:***String*[getUri\(\)](#)

*Void*      [setUri\(String Uri\)](#)

ScriptValue **getUri()**

Required params: none.

Returns the Uri value of the Uri record object.

*Bool* **setUri(String uri)**

Required params: uri.

Sets the Uri value of the Uri record object.

NDEFIconRecord objects can be added to NDEFMessage objects ([createNDEFMessage](#)). They provide access to the methods which allow the user to store and manipulate Icon data in the record.

NDEFIconRecord objects support every [Common Record Methods](#) and the following methods:

**Returns:**

**Method:**

*String*

[getData\(\)](#)

*Void*

[setData\(String data\)](#)

ScriptValue **getData()**

Required params: none.

Returns the icon data stored in the icon record object.

*Bool* **setData(String data)**

Required params: data.

Writes the icon data in the icon record object.

NDEFSmartPosterRecord objects are composed of several [NFC Record Objects](#). After edited, NDEFSmartPosterRecord objects can be added to NDEFMessage objects ([createNDEFMessage](#)). They provide access to the methods which allow the user to store and manipulate Smart Poster data in the record.

NDEFSmartPosterRecord objects support every [Common Record Methods](#) and the following methods:

**Returns:**

**Method:**

*String*

[hasTitle\(String locale\)](#)

*Void*

[getTitleCount\(\)](#)

[NDEFTextRecord](#)

[getTitleRecord\(Int index\)](#)

[NDEFTextRecord](#) list

[getTitleRecords\(\)](#)

*String*

[getTitle\(String locale\)](#)

*Bool*

[addTitle\(String text, String locale, String encoding\)](#)

*Bool*

[removeTitle\(String indexOrLocale\)](#)

[NDEFUriRecord](#)

[getUriRecord\(\)](#)

*String*

[getUri\(\)](#)

*Bool*

[setUri\(string Url\)](#)

*Bool*

[hasAction\(\)](#)

*String*

[getAction\(\)](#)

Bool	<a href="#">setAction</a> ( <i>string action</i> )
Bool	<a href="#">hasSize</a> ()
Int	<a href="#">getSize</a> ()
Bool	<a href="#">setSize</a> ( <i>Int size</i> )
Bool	<a href="#">hasTypeInfo</a> ()
String	<a href="#">getTypeInfo</a> ()
Bool	<a href="#">setTypeInfo</a> ( <i>Int type</i> )
Bool	<a href="#">hasIcon</a> ( <i>String mimeType</i> )
Int	<a href="#">getIconCount</a> ()
<a href="#">NDEFIconRecord</a>	<a href="#">getIconRecord</a> ( <i>Int index</i> )
<a href="#">NDEFIconRecord</a> list	<a href="#">getIconRecords</a> ()
	<a href="#">getIcon</a> ( <i>String mimeType</i> )
	<a href="#">addIcon</a> ( <i>String type, QScriptValue iconData</i> )
	<a href="#">removeIcon</a> ( <i>String iconOrType</i> )

### Bool **hasTitle**(*String locale*)

Required params: none.

Returns if the Smart Poster Record has a title. A locale value can be specified to detect if a specific locale has a designated title.

### Bool **getTitleCount**()

Required params: none.

Returns the amount of title records the Smart Poster Record has.

### ScriptValue **getTitleRecord**(*Int index*)

Required params: none.

Returns an [NDEFTextRecord](#) saved in the [NDEFSmartPosterRecord](#) from the specified **index** in the Smart Poster record object.

If no **index** is specified 0 (zero) will be used as the default value.

### ScriptValue **getTitleRecords**()

Required params: none.

Returns an indexed list with all the title records in the Smart Poster record object.

### String **getTitle**(*String locale*)

Required params: none.

Returns if the Smart Poster Record has a title. A locale value can be specified to detect if a specific locale has a designated title.

### Bool **addTitle**(*String text, String locale, String encoding*)

Required params: text.

Adds a Title to the Smart Poster record.

The **locale** parameter defines the language being used.

If no **locale** is provided the default locale will be "en-US"

The **encoding** parameter defines the character set being used.

If no **encoding** is provided the default locale will be "UTF8"

Supported charsets are **UTF8** and **UTF16**.

Note: There may be several Title records in a single Smart Poster record. However there can only be one Title record for each language identifier (locale).

Bool **removeTitle**(*String indexOrLocale*)

Required params: none.

Removes the title from the Smart Poster record. Titles to be removed can be specified by either the index or the locale.

ScriptValue **getUriRecord**()

Required params: none.

Returns the [NDEFUriRecord](#) saved in the [NDEFSmartPosterRecord](#).

String **getUri**()

Required params: none.

Returns if the current Uri saved in the Uri record.

String **setUri**(*string Url*)

Required params: Url.

Sets the Uri in the uriRecord object.

Bool **hasAction**()

Required params: none.

Returns if the Smart Poster record has an action set.

String **getAction**()

Required params: none.

Returns the action currently configured in the Smart Poster record.

String **setAction**(*string action*)

Required params: Url.

Sets the action in the Smart Poster object.

The **action** parameter suggests a course of action that the device (reader) should do with the content.

The following actions are supported:

**DO**: runs/opens the provided data (send an sms / open a browser / perform a telephone call).

**SAVE**: saves the provided data (store an sms / save the URI in a bookmark / save the telephone number in contacts).

**EDIT**: opens the provided data for editing (open an sms in an sms editor / open an URI in a URI editor / open the telephone number for editing).

Bool **hasSize**()

Required params: none.

Returns if the Smart Poster record has a size record set.

String **getSize**()

Required params: none.

Returns the size currently configured in the size record of the Smart Poster record.

String **setSize**(*Int size*)

Required params: Url.

Sets the size record of the Smart Poster object with the user provided size value.

**Bool** `hasTypeInfo()`

Required params: none.

Returns if the Smart Poster record has a Type record set.

**String** `getTypeInfo()`

Required params: none.

Returns the type currently configured in the size record of the Smart Poster record.

**String** `setTypeInfo(Int type)`

Required params: Url.

Sets the type record of the Smart Poster object with the user provided type value.

The type value describes the type of object that can be reached trough the URI

**Bool** `hasIcon(String mimeType)`

Required params: none.

Returns if the Smart Poster record has any icon records set.

A specific MIME type can be specified when using this method.

**Int** `getIconCount()`

Required params: none.

Returns if the ammount of icon records currently saved in the Smart Poster record.

**String** `getIconRecord(Int index)`

Required params: none.

Returns the icon currently configured in the Smart Poster record.

**ScriptValue** `getIconRecords()`

Required params: none.

Returns an indexed list with all the icon records in the Smart Poster record object.

**String** `getIcon(String mimeType)`

Required params: none.

Returns the icon currently configured in the icon record of the Smart Poster record.

A specific MIME type can be specified when using this method.

**QScriptValue** `addIcon(String type, QScriptValue iconData)`

Required params: appKey, icon.

Adds an **icon** record (image) to the Smart Poster record.

**Bool** `removeIcon(String iconOrType)`

Required params: none.

Removes the icon from the Smart Poster record. Icons to be removed can be specified by either the icon index or the MIME type.

NDEFvCardRecord objects can be added to NDEFMessage objects ([createNDEFMessage](#)). They provide access to the methods which allow the user to store and manipulate vCard data in the record.

NDEFvCardRecord objects support every [Common Record Methods](#) and the following methods:



**Returns:***String***Method:**[addField](#)(*String name*, *String value*)**QScriptValue** [addField](#)(*String name*, *String value*)

Required params: name, value.

Adds a new field to an existing vCard record.

The field is identified by the **name** parameter and will contain the data provided in the **value** parameter.

NDEF record objects follow the NDEF specification. The following common methods are available for all NDEF record objects:

**Returns:***String***Method:**[getTypeNameFormat](#)()*Void*[setTypeNameFormat](#)(*String typeNameFormat*)*String*[getType](#)()*Void*[setType](#)(*String type*)*Int*[getId](#)()*Void*[setId](#)(*String id*)*ScriptData*[getPayload](#)()*Void*[setPayload](#)(*String payload*)*ScriptValue*[isEmpty](#)()**ScriptValue** [getTypeNameFormat](#)();

Required params: none.

Returns the Type Name Format of the record object.

**Void** [setTypeNameFormat](#)(*String typeNameFormat*);

Required params: typeNameFormat.

Sets the Type Name Format of the record object.

**ScriptData** [getType](#)();

Required params: none.

Returns the Type of the record object.

**Void** [setType](#)(*String type*);

Required params: type.

Sets the Type of the record object.

**ScriptData** [getId](#)();

Required params: none.

Returns the ID of the record object.

**Void** [setId](#)(*String id*);

Required params: id.

Sets the ID of the record object.

**ScriptData** [getPayload](#)();

Required params: none.

Returns the Payload of the record object.

```
Void setPayload(String payload);
```

Required params: payload.

Sets the payload of the record object.

```
Bool isEmpty();
```

Required params: none.

Returns if the record object is empty.

## HID

## ISO Prox

Enter topic text here.

UID

```
Bool isValid()
```

Returns "true" if the tag UID property is implemented for this tag type and "false" if not implemented.

```
String read()
```

Required params: none.

Returns the Tag UID.

Note:

If the Tag type does not support UID features the returned value will be undefined.

security

```
Bool isValid()
```

Returns "true" if the tag Security property is implemented for this tag type and "false" if not implemented.

Data

```
Bool isValid()
```

Returns "true" if the tag Data property is implemented for this tag type and "false" if not implemented.

```
Int size()
```

Required params: none.

Retrives the total size of the current Tag's memory as an integer value.

```
QScriptValue read(Int offset, Int size)
```

Required params: offset, size.

Retrieves the data from the Tag in the specified **offset** and with the specified **size**. Returns a QScriptValue object which inherits the respective properties and methods.

Note:

Read methods contain an **.error** property which will return 'undefined' if the read was successful or an error value if there was an error during the read process.

**QScriptValue write(Int offset, QScriptValue data)**

Required params: offset, data.

Writes the **data** defined by the user, in the specified **offset** in the Tag as a QScriptValue object. Returns (QScriptValue) 'True' if successful or 'False' if unsuccessful.

Config

**Bool isValid()**

Returns "true" if the tag Config property is implemented for this tag type and "false" if not implemented.

**Int size()**

Required params: none.

Retrieves the total size of the current Tag configuration memory as an integer value.

## 4.4.5.Contact Tag Types

### SLE4442

Property	Returns:	Method:
UID	QScriptValue	<a href="#">read</a> ()
UID	Bool	<a href="#">isValid</a> ()
Security	QScriptValue	<a href="#">login</a> (Byte PSC1,Byte PSC2,Byte PSC3)
Security	QScriptValue	<a href="#">verifyPin</a> (Byte PSC1,Byte PSC2,Byte PSC3)
Security	QScriptValue	<a href="#">changePin</a> (Byte newPSC1,Byte newPSC2,Byte newPSC3 )
Security	QScriptValue	<a href="#">login</a> (DataBlock Pin)
Security	QScriptValue	<a href="#">verifyPin</a> (DataBlock Pin)
Security	QScriptValue	<a href="#">changePin</a> (DataBlock Pin)
Security	QScriptValue	<a href="#">readSecurityMemory</a> ()
Security	QScriptValue	<a href="#">updateSecurityMemory</a> (DataBlock data)
Security	QScriptValue	<a href="#">updateSecurityMemory</a> (Int offset,DataBlock data)
Data	Int	<a href="#">size</a> ()
Data	QScriptValue	<a href="#">read</a> (Int offset,int size)
Data	QScriptValue	<a href="#">write</a> (Int offset,QScriptValue data)
Config	Int	<a href="#">protectionMemorySize</a> ()
Config	Int	<a href="#">securityMemorySize</a> ()
Config	QScriptValue	<a href="#">readProtectionMemory</a> ()
Config	QScriptValue	<a href="#">writeProtectionMemory</a> (QScriptValue data)

*Config**QScriptValue*[writeProtectionMemory](#)(*Int* **offset**, *QScriptValue* **data**)

## UID

read

### *String* read()

Required params: none.

Returns the Tag UID as a string.

Note:

If the Tag type does not support UID features the returned value will be undefined.

isValid

### *Bool* isValid()

Returns "true" if the current UID value is valid and returns "false" if the current UID value is false.

## security

login

### *QScriptValue* login(*Byte* PSC1,*Byte* PSC2,*Byte* PSC3)

Required params: PSC1, PSC2, PSC3

Authenticates the current Tag Pins with three separate Byte values (**PSC1,PSC2,PSC3**) in either Integer or Hexadecimal format. Returns (Bool) 'True' if successful or 'False' if unsuccessful.

verifyPin

### *QScriptValue* .security.verifyPin(*Byte* PSC1,*Byte* PSC2,*Byte* PSC3)

Required params: PSC1, PSC2, PSC3

Validates the current Tag authentication Pins using three separate Bytes values (**PSC1,PSC2,PSC3**) in either Integer or Hexadecimal format. Returns (QScriptValue) 'True' if successful or 'False' if unsuccessful.. Returns (QScriptValue) 'True' if successful or 'False' if unsuccessful.

changePin

### *QScriptValue* changePin(*Byte* newPSC1,*Byte* newPSC2,*Byte* newPSC3)

Required params: newPSC1, newPSC2, newPSC3

Changes the current Tag authentication Pins using three separate Bytes values (**newPSC1,newPSC2,newPSC3**) in either Integer or Hexadecimal format. Returns (QScriptValue) 'True' if successful or 'False' if unsuccessful.

login

### *QScriptValue* login(*QScriptValue* Pin)

Required params: Pin

Authenticates the current Tag with a user specified **Pin**. Returns (QScriptValue) 'True' if successful or 'False' if unsuccessful.

verifyPin

### *QScriptValue* verifyPin(*QScriptValue* Pin)

Required params: Pin

Validates the current Tag authentication Pin with a user specified **Pin**. Returns (QScriptValue) 'True' if successful or 'False' if unsuccessful.

changePin

*QScriptValue* **changePin(QScriptValue Pin)**

Required params: Pin

Changes the current Tag authentication Pin with a user specified **Pin**. Returns (QScriptValue) 'True' if successful or 'False' if unsuccessful.

readSecurityMemory

*QScriptValue* **readSecurityMemory()**

Required params: none

Retrieves the data from the Tag Security memory. Returns a QScriptValue Object which inherits the respective properties and methods.

Note:

Read methods contain an **.error** property which will return 'undefined' if the read was successful or an error value if there was an error during the read process.

updateSecurityMemory

*QScriptValue* **updateSecurityMemory(QScriptValue data)**

Required params: data

Writes **data** defined by the user in the the Tag Security memory. Returns (QScriptValue) 'True' if successful or 'False' if unsuccessful.

updateSecurityMemory

*QScriptValue* **updateSecurityMemory(Int, offset, QScriptValue data)**

Required params: offset, data

Writes the **data** defined by the user, in the specified **offset** in the Tag as a QScriptValue Object. Returns (QScriptValue) 'True' if successful or 'False' if unsuccessful.

## Data

size

*Int* **size()**

Required params: none.

Retrives the total size of the current Tag as an integer value.

read

*QScriptValue* **read(Int offset, Int size)**

Required params: offset, size.

Retrieves the data from the Tag in the specified **offset** and with the specified **size**. Returns a QScriptValue Object which inherits the respective properties and methods.

Note:

Read methods contain an **.error** property which will return 'undefined' if the read was successful or an error value

if there was an error during the read process.

write

**QScriptValue write(Int offset, QScriptValue data)**

Required params: offset, data.

Writes the **data** defined by the user, in the specified **offset** in the Tag as a QScriptValue. Returns (QScriptValue) 'True' if successful or 'False' if unsuccessful.

## Config

protectionMemorySize

**Int protectionMemorySize()**

Required params: none.

Retrieves the total size of the Tag Protection memory.

securityMemorySize

**Int securityMemorySize()**

Required params: none.

Retrieves the total size of the Tag Security memory.

readProtectionMemory

**QScriptValue readProtectionMemory()**

Required params: none.

Retrieves the data from the Tag Protection memory. Returns a QScriptValue Object which inherits the respective properties and methods.

Note:

Read methods contain an **.error** property which will return 'undefined' if the read was successful or an error value if there was an error during the read process.

writeProtectionMemory

**QScriptValue writeProtectionMemory(Int offset, QScriptValue data)**

Required params: data.

Writes the **data** defined by the user, in the specified **offset** in the Tag Protection memory as a QScriptValue. Returns (QScriptValue) 'True' if successful or 'False' if unsuccessful.

**WARNING:** This method will compare the input bytes with the data of the assigned address in memory card. If they are both the same, the protected bit of the address will be set to 0. If the protected bit is 0, the address will not be allowed to update any more and the protected bit will never be set to 1.

writeProtectionMemory

**QScriptValue writeProtectionMemory(QScriptValue data)**

Required params: data.

Writes the **data** defined by the user, in the Tag Protection memory as a QScriptValue Object. Returns (QScriptValue) 'True' if successful or 'False' if unsuccessful.

**WARNING:** This method will compare the input bytes with the data of the assigned address in memory card. If they are both the same, the protected bit of the address will be set to 0. If the protected bit is 0, the address will not be allowed to update any more and the protected bit will never be set to 1.



## SLE4442 Sample Script

```
//return a list of the currently available contact encoders
encoders = encoding.encoders("CONTACT");

// for each of the detected encoders, run the following script:
for (var iEncoder in encoders) {

    ///return the current encoder and log it
    encoder = encoders[iEncoder];
    log("Detected Encoder : " + encoder.name); //log the Encoder name

    //detect the Tag available in the encoder and log it
    tag = encoder.waitForTag("SLE.4442");
    log("Detected Tag : " + tag.type); //log the Tag name

    //login the SLE4442 tag and verify if login was successful
    authenticated = tag.security.login(data.fromHex("FFFFFF"));
    if (authenticated = "True") {

        //IMPORTANT: Failing to authenticate a Tag several times will result in locking the Tag permanently,
        rendering it useless.

        //write sample data in the Tag main memory, from the offset (bit) 32 which is no longer part of the protection
        memory
        myWriteData = tag.data.write(32, "Some sample data");

        //read the data to confirm it it was correctly saved in the Tag and if we can read from the Tag.
        myReadData = tag.data.read(32, 16);

    }
}
```



# CARDPRESSO

more than an application



[www.cardPresso.com](http://www.cardPresso.com)