3GPP TS 31.131 V16.0.0 (2020-07)

Technical Specification

3rd Generation Partnership Project;

Technical Specification Group Core Network and Terminals;

C-language binding to (U)SIM API

(Release 16)

** 

The present document has been developed within the 3rd Generation Partnership Project (3GPP TM) and may be further elaborated for the purposes of 3GPP.   
The present document has not been subject to any approval process by the 3GPPOrganizational Partners and shall not be implemented.   
This Specification is provided for future development work within 3GPPonly. The Organizational Partners accept no liability for any use of this Specification.  
Specifications and reports for implementation of the 3GPP TM system should be obtained via the 3GPP Organizational Partners' Publications Offices.

Keywords

SIM, USIM, LTE, API

***3GPP***

Postal address

3GPP support office address

650 Route des Lucioles – Sophia Antipolis

Valbonne – FRANCE

Tel. : +33 4 92 94 42 00 Fax : +33 4 93 65 47 16

Internet

[http://www.3gpp.org](http://www.3gpp.org/)

***Copyright Notification***

No part may be reproduced except as authorized by written permission.  
The copyright and the foregoing restriction extend to reproduction in all media.

© 2020, 3GPP Organizational Partners (ARIB, ATIS, CCSA, ETSI, TSDSI, TTA, TTC).

All rights reserved.

UMTS™ is a Trade Mark of ETSI registered for the benefit of its members

3GPP™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners  
LTE™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners

GSM® and the GSM logo are registered and owned by the GSM Association

Contents

Foreword [6](#__RefHeading___Toc169974273)

1 Scope [7](#__RefHeading___Toc169974274)

2 References [7](#__RefHeading___Toc169974275)

3 Definitions and abbreviations [8](#__RefHeading___Toc169974276)

3.1 Definitions [8](#__RefHeading___Toc169974277)

3.2 Abbreviations [8](#__RefHeading___Toc169974278)

4 Description [9](#__RefHeading___Toc169974279)

4.1 Overview [9](#__RefHeading___Toc169974280)

4.2 Design Rationale and Upward Compatibility [10](#__RefHeading___Toc169974281)

4.3 Application Triggering [10](#__RefHeading___Toc169974282)

4.4 Proactive command handling [13](#__RefHeading___Toc169974283)

4.5 Application Loading [13](#__RefHeading___Toc169974284)

5 'C'-language binding for (U)SIM API [13](#__RefHeading___Toc169974285)

5.1 Overview [13](#__RefHeading___Toc169974286)

5.2 Toolkit Application Functions [14](#__RefHeading___Toc169974287)

5.2.1 main [14](#__RefHeading___Toc169974288)

5.2.2 CatGetFrameworkEvent [15](#__RefHeading___Toc169974289)

5.2.3 CatExit [15](#__RefHeading___Toc169974290)

5.3 Registry [16](#__RefHeading___Toc169974291)

5.3.1 CatSetMenuString [16](#__RefHeading___Toc169974292)

5.3.2 CatNotifyOnFrameworkEvent [16](#__RefHeading___Toc169974293)

5.3.3 CatNotifyOnEnvelope [17](#__RefHeading___Toc169974294)

5.3.4 CatNotifyOnEvent [17](#__RefHeading___Toc169974295)

5.4 Man-Machine Interface [17](#__RefHeading___Toc169974296)

5.4.1 CatAddItem [17](#__RefHeading___Toc169974297)

5.4.2 CatSelectItem [17](#__RefHeading___Toc169974298)

5.4.3 CatEndSelectItem [18](#__RefHeading___Toc169974299)

5.4.4 CatDisplayText [18](#__RefHeading___Toc169974300)

5.4.5 CatGetInKey [18](#__RefHeading___Toc169974301)

5.4.6 CatGetInput [19](#__RefHeading___Toc169974302)

5.4.7 CatSetupIdleModeText [19](#__RefHeading___Toc169974303)

5.4.8 CatPlayTone [20](#__RefHeading___Toc169974304)

5.5 Timers [20](#__RefHeading___Toc169974305)

5.5.1 CatGetTimer [20](#__RefHeading___Toc169974306)

5.5.2 CatFreeTimer [20](#__RefHeading___Toc169974307)

5.5.3 CatStartTimer [20](#__RefHeading___Toc169974308)

5.5.4 CatGetTimerValue [21](#__RefHeading___Toc169974309)

5.6 Supplementary Card Reader Management [21](#__RefHeading___Toc169974310)

5.6.1 CatPowerOnCard [21](#__RefHeading___Toc169974311)

5.6.2 CatPowerOffCard [21](#__RefHeading___Toc169974312)

5.6.3 CatPerformCardAPDU [22](#__RefHeading___Toc169974313)

5.6.4 CatGetReaderStatus [22](#__RefHeading___Toc169974314)

5.7 UICC File Store Access [22](#__RefHeading___Toc169974315)

5.7.1 CatSelect [23](#__RefHeading___Toc169974316)

5.7.2 CatStatus [23](#__RefHeading___Toc169974317)

5.7.3 CatGetCHVStatus [23](#__RefHeading___Toc169974318)

5.7.4 CatReadBinary [23](#__RefHeading___Toc169974319)

5.7.5 CatUpdateBinary [24](#__RefHeading___Toc169974320)

5.7.6 CatReadRecord [24](#__RefHeading___Toc169974321)

5.7.7 CatUpdateRecord [24](#__RefHeading___Toc169974322)

5.7.8 CatSearch [25](#__RefHeading___Toc169974323)

5.7.9 CatIncrease [25](#__RefHeading___Toc169974324)

5.7.10 CatInvalidate [25](#__RefHeading___Toc169974325)

5.7.11 CatRehabilitate [25](#__RefHeading___Toc169974326)

5.8 Miscellaneous [26](#__RefHeading___Toc169974327)

5.8.1 CatGetTerminalProfile [26](#__RefHeading___Toc169974328)

5.8.2 CatMoreTime [26](#__RefHeading___Toc169974329)

5.8.3 CatPollingOff [26](#__RefHeading___Toc169974330)

5.8.4 CatPollInterval [26](#__RefHeading___Toc169974331)

5.8.5 CatRefresh [27](#__RefHeading___Toc169974332)

5.8.6 CatLanguageNotification [27](#__RefHeading___Toc169974333)

5.8.7 CatLaunchBrowser [27](#__RefHeading___Toc169974334)

5.9 Low-level Interface [28](#__RefHeading___Toc169974335)

5.9.1 CatResetBuffer [28](#__RefHeading___Toc169974336)

5.9.2 CatStartProactiveCommand [29](#__RefHeading___Toc169974337)

5.9.3 CatSendProactiveCommand [29](#__RefHeading___Toc169974338)

5.9.4 CatOpenEnvelope [29](#__RefHeading___Toc169974339)

5.9.5 CatSendEnvelopeResponse [29](#__RefHeading___Toc169974340)

5.9.6 CatSendEnvelopeErrorResponse [29](#__RefHeading___Toc169974341)

5.9.7 CatPutData [29](#__RefHeading___Toc169974342)

5.9.8 CatPutByte [30](#__RefHeading___Toc169974343)

5.9.9 CatPutTLV [30](#__RefHeading___Toc169974344)

5.9.10 CatPutBytePrefixedTLV [30](#__RefHeading___Toc169974345)

5.9.11 CatPutOneByteTLV [30](#__RefHeading___Toc169974346)

5.9.12 CatPutTwoByteTLV [30](#__RefHeading___Toc169974347)

5.9.13 CatGetByte [31](#__RefHeading___Toc169974348)

5.9.14 CatGetData [31](#__RefHeading___Toc169974349)

5.9.15 CatFindNthTLV [31](#__RefHeading___Toc169974350)

5.9.16 CatFindNthTLVInUserBuffer [31](#__RefHeading___Toc169974351)

5.10 Network Services [32](#__RefHeading___Toc169974352)

5.10.1 CatGetLocationInformation [32](#__RefHeading___Toc169974353)

5.10.2 CatGetTimingAdvance [32](#__RefHeading___Toc169974354)

5.10.3 CatGetIMEI [32](#__RefHeading___Toc169974355)

5.10.4 CatGetNetworkMeasurementResults [32](#__RefHeading___Toc169974356)

5.10.5 CatGetDateTimeAndTimeZone [33](#__RefHeading___Toc169974357)

5.10.6 CatGetLanguage [33](#__RefHeading___Toc169974358)

5.10.7 CatSetupCall [33](#__RefHeading___Toc169974359)

5.10.8 CatSendShortMessage [34](#__RefHeading___Toc169974360)

5.10.9 CatSendSS [35](#__RefHeading___Toc169974361)

5.10.10 CatSendUSSD [35](#__RefHeading___Toc169974362)

5.10.11 CatOpenCSChannel [36](#__RefHeading___Toc169974363)

5.10.12 CatOpenGPRSChannel [37](#__RefHeading___Toc169974364)

5.10.13 CatCloseChannel [39](#__RefHeading___Toc169974365)

5.10.14 CatReceiveData [39](#__RefHeading___Toc169974366)

5.10.15 CatSendData [40](#__RefHeading___Toc169974367)

5.10.16 CatGetChannelStatus [40](#__RefHeading___Toc169974368)

5.10.17 CatServiceSearch [40](#__RefHeading___Toc169974369)

5.10.18 CatGetServiceInformation [41](#__RefHeading___Toc169974370)

5.10.19 CatDeclareService [41](#__RefHeading___Toc169974371)

5.10.20 CatRunATCommand [41](#__RefHeading___Toc169974372)

5.10.21 CatSendDTMFCommand [42](#__RefHeading___Toc169974373)

5.11 Supporting Data Types [42](#__RefHeading___Toc169974374)

5.11.1 CatRecordAccessMode [42](#__RefHeading___Toc169974375)

5.11.2 CatSearchMode [42](#__RefHeading___Toc169974376)

5.11.3 CatFrameworkEventType [42](#__RefHeading___Toc169974377)

5.11.4 CatEnvelopeTagType [43](#__RefHeading___Toc169974378)

5.11.5 CatEventType [43](#__RefHeading___Toc169974379)

5.11.6 CatTextString [43](#__RefHeading___Toc169974380)

5.11.7 CatAlphaString [43](#__RefHeading___Toc169974381)

5.11.8 CatIconIdentifier [43](#__RefHeading___Toc169974382)

5.11.9 CatIconOption [44](#__RefHeading___Toc169974383)

5.11.10 CatDCSValue [44](#__RefHeading___Toc169974384)

5.11.11 CatDisplayTextOptions [44](#__RefHeading___Toc169974385)

5.11.12 CatGetInKeyOptions [44](#__RefHeading___Toc169974386)

5.11.13 CatGetInputOptions [44](#__RefHeading___Toc169974387)

5.11.14 CatSelectItemOptions [45](#__RefHeading___Toc169974388)

5.11.15 CatTimeUnit [45](#__RefHeading___Toc169974389)

5.11.16 CatTone [45](#__RefHeading___Toc169974390)

5.11.17 CatRefreshOptions [45](#__RefHeading___Toc169974391)

5.11.18 CatGetReaderStatusOptions [45](#__RefHeading___Toc169974392)

5.11.19 CatDevice [46](#__RefHeading___Toc169974393)

5.11.20 CatGeneralResult [46](#__RefHeading___Toc169974394)

5.11.21 CatTimerValue [47](#__RefHeading___Toc169974395)

5.11.22 CatTimeInterval [47](#__RefHeading___Toc169974396)

5.11.23 CatFileStatus [47](#__RefHeading___Toc169974397)

5.11.24 CatLanguageNotificationOptions [48](#__RefHeading___Toc169974398)

5.11.25 CatLocationInformation [48](#__RefHeading___Toc169974399)

5.11.26 CatTimingAdvance [48](#__RefHeading___Toc169974400)

5.11.27 CatLaunchBrowserOptions [48](#__RefHeading___Toc169974401)

5.11.28 CatSetupCallOptions [48](#__RefHeading___Toc169974402)

5.11.29 CatTypeOfNumberAndNumberingPlanIdentifier [49](#__RefHeading___Toc169974403)

5.11.30 CatSendShortMessageOptions [49](#__RefHeading___Toc169974404)

5.11.31 CatSendDataOptions [49](#__RefHeading___Toc169974405)

5.11.32 CatMEInterfaceTransportLevelType [50](#__RefHeading___Toc169974406)

5.11.33 CatBearer [50](#__RefHeading___Toc169974407)

5.11.34 CatOpenChannelOptions [50](#__RefHeading___Toc169974408)

5.11.35 CatAddressType [50](#__RefHeading___Toc169974409)

5.11.36 CatFID [50](#__RefHeading___Toc169974410)

5.11.37 CatTextFormat [51](#__RefHeading___Toc169974411)

5.11.38 CatTextForegroundColour [51](#__RefHeading___Toc169974412)

5.11.39 CatTextBackgroundColour [51](#__RefHeading___Toc169974413)

Annex A (normative): Application executable architecture [52](#__RefHeading___Toc169974414)

Annex B (informative): Example [54](#__RefHeading___Toc169974415)

Annex C (informative): Change history [56](#__RefHeading___Toc169974416)

# Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

x the first digit:

1 presented to TSG for information;

2 presented to TSG for approval;

3 or greater indicates TSG approved document under change control.

y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.

z the third digit is incremented when editorial only changes have been incorporated in the document.

# 1 Scope

A Subscriber Identity Module Application Programming Interface (SIM API) has been defined in TS 42.019 [4] as a technology-independent API by which toolkit applications and (U)SIMs co-operate. That specification is independent of the programming language technology used to create the application, the platform used to host the application and the runtime environment used to execute the application.

The present document includes information applicable to (U)SIM toolkit application developers creating applications using the C programming language ISO/IEC 9899 [7]. The present document describes an interface between toolkit applications written in the C programming language and the (U)SIM in order to realize the co-operation set forth in TS 42.019 [4]. In particular, the API described herein provides the service of assembling proactive commands and disassembling the responses to these commands for the application programmer.

Software tools, integrated software development environments and software management systems that may be used to create application programs are explicitly out of scope of the present document.

# 2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

* References are either specific (identified by date of publication, edition number, version number, etc.) or non‑specific.
* For a specific reference, subsequent revisions do not apply.
* For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1] 3GPP TR 21.905: "Vocabulary for 3GPP Specifications".

[2] 3GPP TS 31.111: "USIM Application Toolkit (USAT)".

[3] 3GPP TS 23.048: "Security Mechanisms for the (U)SIM application toolkit; Stage 2".

[4] 3GPP TS 42.019: "Subscriber Identity Module Application Programming Interface (SIM API); Stage 1".

[5] ISO 639 (1988): "Code for the representation of names of languages".

[6] 3GPP TS 23.038: "Alphabets and language‑specific information".

[7] ISO/IEC 9899: "Programming Languages - C".

[8] 3GPP TS 11.14: "Specification of the SIM Application Toolkit for the Subscriber Identity Module - Mobile Equipment (SIM – ME) interface".

[9] Tool Interface Standard (TIS) Executable and Linking Format Specification Version 1.2.

[10] SYSTEM V Application Binary Interface, Edition 4.1.

[11] 3GPP TS 51.011: "Specification of the Subscriber Identity Module - Mobile Equipment (SIM‑ME) interface".

[12] Void.

[13] 3GPP TS 31.115: "Secured packet structure for (U)SIM Toolkit applications".

[14] 3GPP TS 31.116: "Remote APDU Structure for (U)SIM Toolkit applications".

[15] 3GPP TS 31.102: "Characteristics of the USIM Application".

[16] 3GPP TS 31.101: "UICC-Terminal Interface, Physical and Logical Characteristics".

# 3 Definitions and abbreviations

## 3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

**application:** computer program that defines and implements a useful domain-specific functionality  
The term may apply to the functionality itself, to the representation of the functionality in a programming language, or to the realization of the functionality as executable code.

**application executable:** representation of an application as collection of executable codes

**application program:** representation of an application in a programming language such as assembly language, C, Java, WML or XHTML

**Application Programming Interface (API):** collection of entry points and data structures that an application program can access when translated into an application executable

**byte code:** processor-independent representation of a basic computer operation such as "increment by one" that is executed by computer program called a byte code interpreter

**data structure:** memory address that can be accessed by an application executable in order to read or write data

**entry point:** memory address that can be branched to by an application executable in order to access functionality defined by an application-programming interface  
Depending on the software technology, an entry point is also called a subroutine, a function or a method.

**executable code:** generic term for either byte code or native code

**framework:** defines a set of Application Programming Interface (API) functions for developing applications and for providing system services to those applications

**loadfile:** representation of an application executable that is transmitted from the terminal to the smart card operating system  
A loadfile typically includes information about the application executable in addition to the application executable itself.

**native code:** processor-dependent representation of a basic computer operation such as "increment by one" that is executed by the hardware circuitry of a computer's central processing unit

**toolkit application:** uses the commands described in TS 31.111 [2] and TS 11.14 [8]

## 3.2 Abbreviations

For the purpose of the present document, the following abbreviations apply:

APDU Application Protocol Data Unit

API Application Programming Interface

CAT Card Application Toolkit

CS Circuit Switched

DCS Digital Cellular System

DF Dedicated File

DTMF Dual Tone Multiple Frequency

EF Elementary File

ELF Executable and Linkable Format

FID File Identifier

GSM Global System for Mobile communications

ME Mobile Equipment

NAA Network Access Application (SIM or USIM)

OTA Over The Air

SIM Subscriber Identity Module

SMS Short Message Service

STK SIM ToolKit

SW Status Word

TAR Toolkit Application Reference

TLV Tag, Length, Value

TPDU Transport Protocol Data Unit

UICC (not an acronym)

URL Uniform Resource Locator

USIM Universal Subscriber Identity Module

USSD Unstructured Supplementary Services Data

# 4 Description

The (U)SIM Application consists of the following:

- APDU handlers for communicating with the ME;

- File system and file access control;

- Toolkit Framework that provides services to Toolkit applications.

The present document describes the C programming language binding for the interface between the (U)SIM application and toolkit applications described in TS 42.019 [4]. This API allows application programmers using the C programming language to access functions and data described in TS 31.111 [2] and TS 11.14 [8], such that the (U)SIM-based applications and the services they implement can be developed and loaded onto ICCs. If required and supported by the underlying smart card technology, toolkit applications can be loaded or deleted remotely, after the card has been issued.

## 4.1 Overview

The *'C'-binding for (U)SIM API* shall provide function calls for pro-active functions and transport functions. The figure below shows the interactions between a typical toolkit application (shown in blue) and the various functional blocks of the (U)SIM (shown in orange). The C-bindings for these APIs are presented in subclause 4.2.



Figure 1

## 4.2 Design Rationale and Upward Compatibility

Some functions on the C SIM API take parameters that correspond to optional TLVs in TS 31.111 [2] and TS 11.14 [8]. If the actual parameter value passed to the function is NULL, the corresponding TLV is not passed to the ME; an example of an optional parameter is CatIconIdentifier that corresponds to the ICON IDENTIFIER TLV.

Some proactive commands have a very large number of optional TLVs, such as SETUP CALL. Therefore, this API offers two variants that address this aspect, CatSetupCall and CatSetupCallEx. The first function, CatSetupCall, takes as parameters everything that is necessary to issue a successful SETUP CALL proactive command (i.e. everything required to construct the mandatory TLVs as required by TS 31.111 [2] and TS 11.14 [8]) and also includes optional user interface TLVs (title and icon) for ease of use.

The second function, CatSetupCallEx, takes a parameter block that can be extended in future versions of the present specification. The parameter block contains members that correspond to all mandatory and optional TLVs for the SETUP CALL proactive command.

The reason for introducing the "…Ex" variants is threefold:

- Rather than extend the parameter list of a function to take a large number of optional parameters for each call, it is preferable to set up the parameters using named structure members before issuing the call to the function.

- If a future version of TS 31.111 [2] or TS 11.14 [8] extends the optional parameters for a proactive command, the corresponding parameter block can be extended to encompass these parameters without changing the function prototype.

- Any source code written for an older version of this C SIM API can be recompiled with a later version without change and will remain upwardly compatible at the source as long as the suggested coding standards are adhered to.

## 4.3 Application Triggering

The application-triggering portion of the SIM Toolkit Framework is responsible for the activation of toolkit applications, based on the APDU received by the card.

The ME shall not be adversely affected by the presence of applications on the (U)ICC card. For instance a syntactically correct Envelope shall not result in an error status word in case of a failure of an application. The only application as seen by the ME is the (U)SIM application. As a result, a toolkit application may return an error, but this error will not be sent to the ME.

The difference between an application and a toolkit application is that a toolkit application does not typically handle APDUs directly. It will handle higher-level messages. Furthermore the execution of a function could span over multiple APDUs, in particular, the proactive protocol commands.

All the applications that have registered interest in the event are triggered in order of their priority.

- The current context is switched to the toolkit application.

- A pending transaction is aborted.

- The current file context of the toolkit application is the MF.

- The current file context of the current selected application is unchanged.

On termination of a toolkit application execution of CatExit():

- The context switches back to the context of the current selected application, the NAA application.

- A pending toolkit application transaction is aborted.

Here after are the events that can trigger a toolkit application:

*EVENT\_PROFILE\_DOWNLOAD*

Upon reception of the Terminal Profile command by the (U)SIM, the Toolkit Framework stores the ME profile and then triggers the registered toolkit application that may want to change their registry. A toolkit application may not be able to issue a proactive command.

*EVENT\_MENU\_SELECTION, EVENT\_MENU\_SELECTION\_HELP\_REQUEST*

A toolkit application might be activated upon selection in the ME's menu by the user, or request help on this specific menu.

In order to allow the user to choose in a menu, the Toolkit Framework shall have previously issued a SET UP MENU proactive command. When a toolkit application changes a menu entry of its registry object, the Toolkit Framework shall dynamically update the menu stored in the ME during the current card session. The SIM Toolkit Framework shall use the data of the EFsume file (TS 51.011 [11] and TS 31.102 [15]) when issuing the SET UP MENU proactive command.

The positions of the toolkit application menu entries in the item list, the requested item identifiers and the associated limits (e.g. maximum length of item text string) are defined at the loading of the toolkit application.

If at least one toolkit application registers to EVENT\_MENU\_SELECTION\_HELP\_REQUEST, the SET UP MENU proactive command sent by the Toolkit Framework shall indicate to the ME that help information is available. A toolkit application registered for one or more menu entries may be triggered by the event EVENT\_MENU\_SELECTION\_HELP\_REQUEST, even if it is not registered to this event. A toolkit application registered for one or more menu entries should provide help information.

*EVENT\_FORMATTED\_SMS\_PP\_ENV, EVENT\_UNFORMATTED\_SMS\_PP\_ENV,*

*EVENT\_FORMATTED\_SMS\_PP\_UPD, EVENT\_UNFORMATTED\_SMS\_PP\_UPD*

A toolkit application can be activated upon the reception of a short message. There are two ways for a card to receive an SMS: via the Envelope SMS-PP Data Download or the UpdateRecord EFsms instruction.

The reception of the SMS by the toolkit application cannot be guaranteed for the Update Record EFsms instruction.

The received SMS may be:

- formatted according to TS 23.048 [3] or an other protocol to identify explicitly the toolkit application for which the message is sent;

- unformatted or using a toolkit application specific protocol the Toolkit Framework will pass this data to all registered toolkit applications.

*EVENT\_FORMATTED\_SMS\_PP\_ENV*

This event is triggered by an envelope APDU containing an SMS\_DATADOWNLOAD BER TLV with an SMS\_TPDU simple TLV according to TS 23.048 [3].

The Toolkit Framework shall:

- verify the TS 23.048 [3] security of the SMS TPDU;

- trigger the toolkit application registered with the corresponding TAR defined at application loading;

- take the optional Application Data posted by the triggered toolkit application if present;

- secure and send the response packet.

The toolkit application will only be triggered if the TAR is known and the security verified. Application data will also be deciphered.

*EVENT\_UNFORMATTED\_SMS\_PP\_ENV*

The registered toolkit applications will be triggered by this event and get the data transmitted in the APDU envelope SMS\_DATADOWNLOAD.

*EVENT\_FORMATTED\_SMS\_PP\_UPD*

This event is triggered by Update Record EFsms with an SMS TP-UD field formatted according to TS 23.048 [3].

The Toolkit Framework shall:

- update the EFsms file with the data received, it is then up to the receiving toolkit application to change the SMS stored in the file (i.e. the toolkit application need to have access to the EFsms file);

- verify the TS 23.048 [3] security of the SMS TPDU;

- convert the Update Record EFsms in a TLV List, an EnvelopeHandler;

- trigger the toolkit application registered with the corresponding TAR defined at application loading.

*EVENT\_UNFORMATTED\_SMS\_PP\_UPD*

The SIM Toolkit Framework will first update the EFsms file, convert the received APDU as described above, and then trigger all the registered toolkit applications. All of them may modify the content of EFsms (i.e. the toolkit applications need to have access to the EFsms file).

*EVENT\_UNFORMATTED\_SMS\_CB*

When the ME receives a new cell broadcast message, the cell broadcast page may be passed to the card using the envelope command. e.g. the application may then read the message and extract a meaningful piece of information that could be displayed to the user, for instance.

*EVENT\_CALL\_CONTROL\_BY\_SIM*

When the NAA is in call control mode and when the user dials a number, this number is passed to the Toolkit Framework. Only one toolkit application can handle the answer to this command: call barred, modified or accepted.

*EVENT\_EVENT\_DOWNLOAD\_MT\_CALL, EVENT\_EVENT\_DOWNLOAD\_CALL\_CONNECTED,*

*EVENT\_EVENT\_DOWNLOAD\_CALL\_DISCONNECTED, EVENT\_EVENT\_DOWNLOAD\_LOCATION\_STATUS,*

*EVENT\_EVENT\_DOWNLOAD\_USER\_ACTIVITY, EVENT\_EVENT\_DOWNLOAD\_IDLE\_SCREEN\_AVAILABLE,*

*EVENT\_EVENT\_DOWNLOAD\_CARD\_READER\_STATUS*

The toolkit application will be triggered by the registered event download trigger, upon reception of the corresponding Envelope command. In order to allow the toolkit application to be triggered by these events, the Toolkit Framework shall have previously issued a SET UP EVENT LIST proactive command. When a toolkit application changes one or more of these requested events of its registry, the Toolkit Framework shall dynamically update the event list stored in the ME during the current card session.

*EVENT\_MO\_SHORT\_MESSAGE\_CONTROL\_BY\_SIM*

Before sending an SMS MO entered by the user, the SMS is submitted to the Toolkit framework. Only one toolkit application can register to this event.

*EVENT\_TIMER\_EXPIRATION*

This event is registered when the application executes a successful Toolkit CatGetTimer(). The toolkit application can then manage this (these) timer(s), and it will be triggered at the reception of the APDU Envelope TIMER EXPIRATION. The Toolkit Framework shall reply busy to this Envelope APDU if it cannot guaranty to trigger the corresponding toolkit application.

*EVENT\_UNRECOGNIZED\_ENVELOPE*

The application registered to this event shall be triggered by the framework if the BER-TLV tag contained in the ENVELOPE APDU is not defined in the associated release of TS 31.111 [2] and TS 11.14 [8] and if no corresponding constant is defined in the list of the ToolkitConstants interface. By providing the means to transfer an arbitrary block of data, the Unrecognized Envelope Event will allow a toolkit application to handle the evolution of the specifications TS 31.111 [2] and TS 11.14 [8].

*EVENT\_STATUS\_COMMAND*

At reception of a STATUS APDU command, the SIM Toolkit Framework shall trigger the registered toolkit application.

A range of events is reserved for experimental and proprietary usage (from -128 to -1). As the definition of these events is not standardized, the use of these events may make the toolkit application behave differently on different platforms.

The toolkit application shall be triggered for the registered events upon reception, and shall be able to access to the data associated to the event using OpenEnvelope() or the low-level functions.

The order of triggering the toolkit application shall follow the priority level of each toolkit application defined at its loading. If several toolkit applications have the same priority level, the last loaded toolkit application takes precedence.

## 4.4 Proactive command handling

The (U)SIM application toolkit protocol (i.e. 91xx, Fetch, Terminal Response) is handled by the network access application and the Toolkit Framework. The toolkit application shall not handle those events.

The network access application and the Toolkit Framework shall handle the transmission of the proactive command to the ME, and the reception of the response. The Toolkit Framework will then return in the toolkit application just after the proactive command. It shall then provide to the toolkit application the values as indicated in the function parameters. It also provides the raw return information so that the toolkit application can analyse the response.

The proactive command is sent to the ME as defined and constructed by the toolkit library without any check of the Toolkit Framework.

The toolkit application shall not issue the following proactive commands: SET UP MENU, SET UP EVENT LIST, POLL INTERVAL, POLLING OFF; as those are system proactive commands that will affect the services of the Toolkit Framework.

## 4.5 Application Loading

Applications compliant to the present document are represented for loading as loadfiles in the Executable and Linkable Format (ELF) described in Tool Interface Standard (TIS) Executable and Linking Format Specification [9] and SYSTEM V Application Binary Interface [10]. The application executable in the ELF loadfile may be either native code or byte code that has been created through a process of compiling the representation of the application program in the C programming language.

The e\_machine entry in the ELF header is set to according to the table in annex A and indicates the architecture for which the application executable in the loadfile has been prepared.

Coding for other processors, processor instruction set extensions and byte code interpreters will be defined as needed processor-specific or interpreter-specific supplements to SYSTEM V Application Binary Interface [10] may also be provided as needed.

Loadfile linkers, loaders and installers, whether on-card or off-card, return an error condition if the application representation in the loadfile cannot be accommodated or if resources requested by the application are not available.

The over-the-air application loading mechanism, protocol and application life cycle are defined in TS 23.028 [3].

# 5 'C'-language binding for (U)SIM API

## 5.1 Overview

This subclause presents the 'C'-language binding to (U)SIM API. It is divided into sections as follows:

- Toolkit application entry and exit.

- Man-Machine Interface.

- Timers.

- Supplementary card reader.

- UICC file store access.

- Registry.

- Miscellaneous.

- Low-level functions.

- Network services.

- Supporting data types.

For each function, the prototype is given followed by a table describing the parameters and whether they are input [in] or output [out] parameters. There is explanatory text which explains the function's purpose and whether it is a proactive command or not.

## 5.2 Toolkit Application Functions

Toolkit applications will start by executing the application-defined function *main*. There are no arguments to *main*, nor are there any return results. The application can find out why it was invoked using the *CatGetFrameworkEventt*  function. The Framework events that can cause an application to be invoked can be split into the following groups:

- Command monitoring.

- ME monitor events.

- Application lifecycle change.

Command monitoring enables applications to be invoked when the framework receives commands from the ME. Currently supported commands that can be monitored are:

- TERMINAL PROFILE: monitoring this command enables an application to be invoked when the ME is powered on.

- STATUS: monitoring this command enables an application to be invoked when the ME polls for proactive commands.

- ENVELOPE: monitoring this command enables the application to be informed of specific envelope type arrival for example call control envelopes can be monitored.

ME monitor events are events that the framework can ask the ME to monitor; for example an event can be sent on call connection. ME monitored events are delivered to the application in the EVENT DOWNLOAD envelope as received from the ME.

The application lifecycle event enables the framework to invoke an application when the application status has changed. This is mainly to enable an application to be run at installation time so that it can set up its registry entries. The details of the application lifecycle events are provided in TS 31.116 [14].

### 5.2.1 main

**void  
main (void);**

The main function is the application entry point. The application should not return from *main*; it must call the *CatExit* function.

An example main function is given below:

void main(void)  
 {  
 switch (CatGetFrameworkEvent())  
 {  
 case EVENT\_APPLICATION\_LIFECYCLE\_INSTALL:  
 // set up registry for this application  
 CatSetMenuString(…..  
 CatNotifyOnEnvelope(SMS\_PP\_DOWNLOAD\_TAG,1);  
 CatNotifyOnEvent(CARD\_READER\_STATUS,1);  
 break;  
 case EVENT\_ENVELOPE\_COMMAND:  
 {  
 BYTE length;  
 switch (CatOpenEnvelope(&length))  
 {  
 case MENU\_SELECTION\_TAG:  
 // search for help request …..  
 break;  
 case SMS\_PP\_DOWNLOAD\_TAG:  
 …..  
 break;  
 case EVENT\_DOWNLOAD\_TAG:  
 // search for card reader status event …..  
 break;  
 default:  
 CatExit();  
 }  
 }  
 break;  
 default:  
 CatExit();  
 break;  
 }  
 CatExit();  
 }

### 5.2.2 CatGetFrameworkEvent

**CatFrameworkEventType  
CatGetFrameworkEvent(void);**

|  |  |  |
| --- | --- | --- |
| *RETURN* |  | Framework event type that caused the application to run; see [CatFrameworkEventType](#_SimEventType) for details. |

### 5.2.3 CatExit

**void  
CatExit (void) ;**

|  |  |  |
| --- | --- | --- |
|  |  |  |

*CatExit* causes the application to terminate execution and return control to the framework. When the application is restarted, it enters at *main*.

## 5.3 Registry

The menu entry(ies) of the application, together with the set of framework events that the application is interested in, may be registered using the functions defined in this subclause.

### 5.3.1 CatSetMenuString

**void  
CatSetMenuString (BYTE** *MenuID*,  
 **BYTE** *MenuStringLength*, **const void \****MenuString*,  
 **const CatIconIdentifier** **\****IconIdentifier*,  
 **BYTE** *HelpAvailable,*   
 **BYTE** *NextAction***)**;

|  |  |  |
| --- | --- | --- |
| *MenuID* | [in] | The menu ID by which this entry is known. |
| *MenuStringLength* | [in] | The length, in bytes, of MenuString. |
| *MenuString* | [in] | The menu entry to be placed in the registry. If MenuString is NULL or MenuStringLength is zero, any existing menu entry associated with MenuID is removed and is not displayed by the ME. |
| *IconIdentifier* | [in] | Optional icon identifier; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If IconIdentifier is NULL or if IconIdentifier.UseIcon is zero, no icon identifier is sent to the ME. |
| *HelpAvailable* | [in] | If non zero the application can supply help. |
| *NextAction* | [in] | The (optional) next action value |

*CatSetMenuString* allows the application to define a menu entry together with an icon. A non-zero value can be supplied if a next action indicator is required. This function will implicitly request that the application is notified of menu selection envelopes i.e. there is no requirement to call the *CatNotifiyOnEnvelope* function. An application can have several menu entries and must examine the menu selection envelope to decide which menu selection caused it to be invoked.

The ordering of menu entries within a menu presented by the ME is based on increasing integer values of identifiers selected by the application. Note that any application's menu item ordering may be further overridden by an external source, e.g. card issuer, via a request to the SIM Toolkit framework this mechanism is beyond the scope of the present document.

### 5.3.2 CatNotifyOnFrameworkEvent

**void  
CatNotifyOnFrameworkEvent(CatFrameworkEventType Event, BYTE Enabled);**

|  |  |  |
| --- | --- | --- |
| *Event* | [in] | A framework event the application is interested in, see [CatFrameworkEventType](#_SimFrameworkEventType) for details. |
| *Enabled* | [in] | If non-zero the framework event is monitored otherwise the framework event isn't monitored. By default only application lifecycle events are monitored. |

*CatNotifyOnFrameworkEvent* enables the application to add/remove a framework event to/from the set of framework events that it is interested in.

### 5.3.3 CatNotifyOnEnvelope

**void  
CatNotifyOnEnvelope(CatEnvelopeTagType Tag, BYTE Enabled);**

|  |  |  |
| --- | --- | --- |
| *Tag* | [in] | The particular envelope type to monitor; see [CatEnvelopeTagType](#_SimEnvelopeTagType) for details. |
| *Enabled* | [in] | If non-zero the envelope type is monitored otherwise the envelope type isn't monitored. |

*CatNotifyOnEnvelope* enables the application to add/remove an envelope monitoring event to/from the set of the envelope monitoring events it is interested in. Note that the monitoring of MENU SELECTION, TIMER EXPIRATION and EVENT DOWNLOAD envelopes is handled by the framework.

### 5.3.4 CatNotifyOnEvent

**void  
CatNotifyOnEvent(CatEventType EventType, BYTE Enabled);**

|  |  |  |
| --- | --- | --- |
| *EventType* | [in] | The particular event type to monitor; see [CatEventType](#_SimEventType_1) for details. |
| *Enabled* | [in] | If non-zero the event type is monitored otherwise the event isn't monitored. |

*CatNotifyOnEvent* enables the application to add/remove an ME monitored event to/from the set of ME monitored events it is interested in.

## 5.4 Man-Machine Interface

### 5.4.1 CatAddItem

**void  
CatAddItem(BYTE** *ItemTextLength*, **const void \****ItemText*, **BYTE** *ItemIdentifier***)**;

|  |  |  |
| --- | --- | --- |
| ItemTextLength | [in] | The length in bytes of the following ItemText field. |
| *ItemText* | [in] | Text associated with item. |
| *ItemIdentifier* | [in] | Specifies a unique identifier to be associated with this selection. This value is returned in the SelectedItem parameter of [CatSelectItem](#_GsmSelectItem) if this item is selected from the menu. |

*CatAddItem* adds an item to a list for the user to select. See *CatSelectItem* below for details on the construction of a display list.

### 5.4.2 CatSelectItem

**void  
CatSelectItem (BYTE** *TitleLength*, **const void \****Title*,  
 **CatSelectItemOptions** *Options***)**;

|  |  |  |
| --- | --- | --- |
| *Title*Length | [in] | The length in bytes of Title. |
| *Title* | [in] | Title of the list of choices. |
| Options | [in] | Acceptable values for this parameter are listed in [CatSelectItemOptions](#_SimSelectItemOptions). |

*CatSelectItem* initiates the construction of a list of items to be displayed to the user and from which the user is expected to select exactly one entry. After *CatSelectItem* has been called, entries are added to the list one at a time using the *CatAddItem* entry point above. When all items have been added to the list, the list is sent to the ME using the *CatEndSelectItem* entry point below. *CatEndSelectItem* causes the list to be displayed and returns to the caller the item selected.

### 5.4.3 CatEndSelectItem

**CatGeneralResult  
CatEndSelectItem (BYTE \****SelectedItem,* **const CatIconIdentifier \****IconIdentifier***)**;

|  |  |  |
| --- | --- | --- |
| *SelectedItem* | [out] | Index of item selected by user. |
| *IconIdentifier* | [in] | Optional icon identifier; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If IconIdentifier is NULL or if IconIdentifier.UseIcon is zero, no icon identifier is sent to the ME. |
| *RETURN* |  | The GeneralResult code of the SELECT ITEM proactive command. |

*CatEndSelectItem* issues the proactive command SELECT ITEM that displays on the ME a list of items for the user to choose from. The terminal response is parsed and if successful the *SelectedItem* parameter is set to the index of the item chosen. See *CatSelectItem* above for details on the construction of a display list.

### 5.4.4 CatDisplayText

**CatGeneralResult  
CatDisplayText (CatDCSValue** *TextDCS*, **BYTE** *TextLength*, **const void \****Text*,  
 **CatDisplayTextOptions Text***Options*,

**CatTextFormat TextFormat,**

**CatForegroundColor ForegroundColour,**

**CatBackgroundColour BackgroundColour,  
 const CatIconIdentifier \****IconIdentifier*,  
 **BYTE** *ImmediateResponse***)**;

|  |  |  |
| --- | --- | --- |
| TextDCS | [in] | The data coding scheme for Text. Acceptable values for this parameter are listed in [CatDCSValue](#_GsmDCSValue). |
| TextLength | [in] | The length in bytes of Text. |
| *Text* | [in] | String to display on ME. |
| *TextOptions* | [in] | Acceptable values for this parameter are listed in [CatDisplayTextOptions](#_GsmDisplayTextOptions). |
| *TextFormat* | [in] | Format of text; e.g. bold, italic, etc. |
| *ForegroundColour* | [in] | Foreground colour of text; i.e. colour of the letters. |
| *BackgroundColour* | [in] | Background colour of text; i.e. colour filled in behind the letters. |
| *IconIdentifier* | [in] | Optional icon identifier; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If *IconIdentifier* is NULL or if *IconIdentifier*.*UseIcon* is zero, no icon identifier is sent to the ME. |
| *ImmediateResponse* | [in] | True-program continues execution as soon as ME receives instruction.  False-program waits until text is cleared on the ME before continuing, and the Immediate Response TLV is not passed to the ME. |
| *RETURN* |  | The GeneralResult code of the DISPLAY TEXT proactive command. |

### 5.4.5 CatGetInKey

**CatGeneralResult  
CatGetInKey (CatDCSValue** *TitleDCS*, **BYTE** *TitleLength*, **const void \****Title*,  
 **CatGetInKeyOptions** *Options*,   
 **const CatIconIdentifier \****IconIdentifier*,  
 **CatDCSValue \****DCSOut*, **void \****KeyOut***)**;

|  |  |  |
| --- | --- | --- |
| TitleDCS | [in] | The data-coding scheme for Title. Acceptable values for this parameter are listed in [CatDCSValue](#_GsmDCSValue). |
| TitleLength | [in] | The length in bytes of Title. |
| *Title* | [in] | String to display on ME. |
| *Options* | [in] | Acceptable values for this parameter are listed in [CatGetInKeyOptions](#_GsmGetInKeyOptions). |
| *IconIdentifier* | [in] | Optional icon identifier; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If *IconIdentifier* is NULL or if *IconIdentifier*.*UseIcon* is zero, no icon identifier is sent to the ME. |
| *DcsOut* | [out] | The packing type of the returned key. This parameter is set to one of the values listed in [CatDCSValue](#_GsmDCSValue). |
| *KeyOut* | [out] | The key pressed. |
| *RETURN* |  | The GeneralResult code of the GET INKEY proactive command. |

*CatGetInKey* issues the proactive command GET INKEY. The terminal response is parsed and if successful the *DCSOut* and *KeyOut* parameters are updated.

### 5.4.6 CatGetInput

**CatGeneralResult  
CatGetInput(CatDCSValue** *TitleDCS*, **BYTE** *TitleLength*, **const void \****Title*,   
 **CatGetInputOptions** *Options*,   
 **CatDCSValue** *DefaultReplyDCS*,   
 **BYTE** *DefaultReplyLength*, **const void \****DefaultReply*,   
 **BYTE** *MinimumResponseLength*,   
 **BYTE** *MaximumResponseLength*,   
 **const CatIconIdentifier \****IconIdentifier*,  
 **CatDCSValue \****MsgOutDCS*, **BYTE \****MsgOutLength*, **void \****MsgOut*);

|  |  |  |
| --- | --- | --- |
| TitleDCS | [in] | The data-coding scheme for Title. Acceptable values for this parameter are listed in [CatDCSValue](#_GsmDCSValue). |
| TitleLength | [in] | The length in bytes of Title. |
| *Title* | [in] | String to display on ME while waiting for the user to press a key. |
| *Options* | [in] | Acceptable values for this parameter are listed in [CatGetInputOptions](#_GsmGetInputOptions). |
| *DefaultReplyDCS* | [in] | The data-coding scheme for DefaultReply. Acceptable values for this parameter are listed in [CatDCSValue](#_GsmDCSValue). |
| *DefaultReplyLength* | [in] | The length in bytes of DefaultReply. |
| *DefaultReply* | [in] | Default response string; use NULL for "no reply"-no Default Reply tag length value (TLV) is sent to the ME. |
| *MinimumResponseLength* | [in] | Minimum allowed length for the response, in either characters or digits. |
| *MaximumResponseLength* | [in] | Maximum allowed length for the response, in either characters or digits. |
| *IconIdentifier* | [in] | Optional icon identifier; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If *IconIdentifier* is NULL or if *IconIdentifier*.*UseIcon* is zero, no icon identifier is sent to the ME. |
| *MsgOutDCS* | [out] | Packing type of the returned data. This parameter is set to one of the values listed in [CatDCSValue](#_GsmDCSValue). |
| *MsgOutLength* | [out] | Length of the returned message in bytes. |
| *MsgOut* | [out] | A pointer to where the returned string or message is placed. |
| *RETURN* |  | The GeneralResult code of the GET INPUT proactive command. |

*CatGetInput* issues the proactive command GET INPUT. The terminal response is parsed and if successful *MsgOutDCS*, *MsgOutLength*, *MsgOut* parameters are updated.

### 5.4.7 CatSetupIdleModeText

**CatGeneralResult  
CatSetupIdleModeText (CatDCSValue** *TextDCS*, **BYTE** *TextLength*, **const void \****Text*,  
 **const CatIconIdentifier \****IconIdentifier*);

|  |  |  |
| --- | --- | --- |
| TextDCS | [in] | The data-coding scheme for Text. Acceptable values for this parameter are listed in [CatDCSValue](#_GsmDCSValue). |
| TextLength | [in] | The length in bytes of Text. |
| *Text* | [in] | String to display while ME is idle. |
| *IconIdentifier* | [in] | Optional icon identifier; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If *IconIdentifier* is NULL or if *IconIdentifier*.*UseIcon* is zero, no icon identifier is sent to the ME. |
| *RETURN* |  | The GeneralResult code of the SETUP IDLE MODE TEXT proactive command. |

*CatSetupIdleModeText* issues the proactive command SET UP IDLE MODE TEXT that sets the ME's default text string.

### 5.4.8 CatPlayTone

**CatGeneralResult  
CatPlayTone (BYTE** *TextLength*, **const void \****Text*,  
 **CatTone** *Tone*,  
 **CatTimeUnit** *Units*, **BYTE** *Duration,* **const CatIconIdentifier \****IconIdentifier*);

|  |  |  |
| --- | --- | --- |
| *TextLength* | [in] | The length in bytes of the string Text to display on the ME. |
| *Text* | [in] | String to display on ME while sound is being played. |
| *Tone* | [in] | Specifies tone to play. Acceptable values for this parameter are listed in [CatTone](#_GsmTone). |
| *Units* | [in] | Unit of time specified for *duration* parameter. Acceptable values for this parameter are listed in [CatTimeUnit](#_GsmTimeUnit). |
| *Duration* | [in] | Amount of time to play the tone, in units specified in the *Units* parameter |
| *IconIdentifier* | [in] | Optional icon identifier; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If *IconIdentifier* is NULL or if *IconIdentifier*.*UseIcon* is zero, no icon identifier is sent to the ME. |
| *RETURN* |  | The GeneralResult code of the PLAY TONE proactive command. |

*CatPlayTone* issues the proactive command PLAY TONE.

## 5.5 Timers

### 5.5.1 CatGetTimer

**BYTE  
CatGetTimer (void);**

|  |  |  |
| --- | --- | --- |
| *RETURN* |  | The identifier of the timer. |

*CatGetTimer* returns the ID of a timer that is not currently in use. If no timer is available, this function returns zero. Timer identifiers are assigned by the framework.

### 5.5.2 CatFreeTimer

**void  
CatFreeTimer (BYTE** *TimerID***)**;

|  |  |  |
| --- | --- | --- |
| *TimerID* | [in] | ID of timer to free; obtained from [CatGetTimer](#_GsmGetTimer). |

*CatFreeTimer* frees the handle to the specified timer, making it available for the next request. It is not a proactive command. No information is passed to the ME by this function.

### 5.5.3 CatStartTimer

**void  
CatStartTimer (BYTE** *TimerID*, **CatTimerValue \****TimerValue***)**;

|  |  |  |
| --- | --- | --- |
| *TimerID* | [in] | ID of the timer to initialize; obtained from [CatGetTimer](#_GsmGetTimer). |
| *TimerValue* | [in] | Initial value of the timer. The value is specified in a structure of type [CatTimerValue](#_GsmTimerValue). |
| *RETURN* |  | The GeneralResult code of the TIMER MANAGEMENT proactive command. |

*CatStartTimer* issues a proactive TIMER MANAGEMENT command to initialize a timer to the parameter values.

### 5.5.4 CatGetTimerValue

**void  
CatGetTimerValue (BYTE** *TimerID*, **CatTimerValue \****TimerValue***)**;

|  |  |  |
| --- | --- | --- |
| *TimerID* | [in] | ID of the timer from which to obtain values; obtained from [CatGetTimer](#_GsmGetTimer) |
| *TimerValue* | [out] | The time remaining to run of timer *TimerID*. The value is returned in a structure of type [CatTimerValue](#_GsmTimerValue). |
| *RETURN* |  | The GeneralResult code of the TIMER MANAGEMENT proactive command. |

*CatGetTimerValue* issues a proactive TIMER MANAGEMENT command to obtain the timer's current value.

## 5.6 Supplementary Card Reader Management

These functions access the supplementary card-reader on a dual-slot ME.

### 5.6.1 CatPowerOnCard

**CatGeneralResult  
CatPowerOnCard (CatDevice** *DeviceID*, **BYTE \****ATRLength,* **void \****ATR***)**;

|  |  |  |
| --- | --- | --- |
| *DeviceID* | [in] | The device to power on. An acceptable value for this parameter is a card reader device selected from [CatDevice](#_SimDevice). |
| *ATRLength*[in/out] | Size of the ATR buffer on input and the umber of bytes returned by the card as the ATR on output. |  |
| *ATR* | [out] | Pointer to where answer to reset (ATR) will be stored. |
|  |  |  |
| *RETURN* |  | The GeneralResult code of the POWER ON CARD proactive command. |

*CatPowerOnCard* issues the proactive command POWER ON CARD that powers on a supplementary card reader. The terminal response is parsed and if successful the *ATR* and *ATRLength* parameters are.

### 5.6.2 CatPowerOffCard

**CatGeneralResult  
CatPowerOffCard (CatDevice** *DeviceID***)**;

|  |  |  |
| --- | --- | --- |
| *DeviceID* | [in] | The device to power off. An acceptable value for this parameter is a card reader device selected from [CatDevice](#_SimDevice). |
| *RETURN* |  | The GeneralResult code of the POWER OFF CARD proactive command. |

*CatPowerOffCard* issues the proactive command POWER OFF CARD that turns off the supplementary card reader.

### 5.6.3 CatPerformCardAPDU

**CatGeneralResult  
CatPerformCardAPDU (CatDevice** *DeviceID*,  
 **BYTE** *CAPDULength*,**const void \****CAPDU,*  
 **BYTE \****RAPDULength,* **void \****RAPDU***)**;

|  |  |  |
| --- | --- | --- |
| *DeviceID* | [in] | The device to send the command APDU (C-APDU) to. An acceptable value for this parameter is a card reader device selected from [CatDevice](#_SimDevice). |
| *CAPDU* | [in] | *Pointer to the command C-APDU to be sent to the additional card device.* |
| *CAPDULength* | [in] | The number of bytes in the C-APDU. |
| *RAPDU* | [out] | Pointer to the buffer that will contain the response APDU (R-APDU) returned by the card in the additional card reader. You must allocate enough space to hold the R-APDU sent by the card. |
| *RAPDULength* | [out] | The number of bytes returned by the card in the additional card reader. |
| *RETURN* |  | The GeneralResult code of the PERFORM CARD APDU proactive command. |

*CatPerformCardAPDU* issues the proactive command PERFORM CARD APDU that sends application program data units (APDU) to the supplementary card reader. The terminal response is parsed and if successful the *RAPDU* and *RAPDULength* parameters are updated.

### 5.6.4 CatGetReaderStatus

**CatGeneralResult  
CatGetReaderStatus (CatDevice** *DeviceID*, **CatReaderStatusOptions***Options*,  
 **BYTE \****Status***)**;

|  |  |  |
| --- | --- | --- |
| DeviceID | [in] | Device to detect status of. An acceptable value for this parameter is a card reader device selected from [CatDevice](#_SimDevice). |
| *Options* | [in] | Selects what type of status information to return. An acceptable value for this parameter is selected from [CatGetReaderStatusOptions](#_SimReaderStatusOptions). |
| *Status* | [out] | Status of additional card reader. |
| *RETURN* |  | The GeneralResult code of the GET READER STATUS proactive command. |

*CatGetReaderStatus* issues the proactive command GET READER STATUS that retrieves the status of the additional card readers on the ME. The terminal response is parsed and if successful the *Status* parameter is updated.

## 5.7 UICC File Store Access

The abstract type FID is used to denote the file and a set of pre-processor macros are defined that enumerate all of the standard files of a NAA file store. A FID could be implemented as an unsigned 16-bit number as follows:

- typedef unsigned short FID;

- #define FID\_MF 0x3F00

The starting file-context of a Toolkit application is the MF. When a Toolkit application exits, the file-context is lost.

The Access Control privileges of the application are granted during installation according to the level of trust. When an application requests access to UICC or operator specific files, the Toolkit Framework checks if this access is allowed by examination of the file control information stored on the card. If access is granted the Toolkit Framework will process the access request, if access is not granted, an appropriate status word will be returned.

Contents and coding of the file(s) containing access control information are defined in 3GPP TS 31.101 [16].

All UICC functions return the status bytes according to 3GPP TS 31.101 [16], where 90 00 represents success.

### 5.7.1 CatSelect

**UINT16  
CatSelect (CatFID** *FileIdentifier,* **CatFileStatus \****Status***)**;

|  |  |  |
| --- | --- | --- |
| *FileIdentifier* | [in] | *The file to select.* |
| *Status*[out] | *Useful information about the directory or file if it is successfully selected.* |  |
| *RETURN* | *The returned 16-bit unsigned value is a concatenation of the SW response bytes with SW1 as the high byte and SW2 as the low byte, so a successful execution would return 0x9000.* |  |

*CatSelect* selects the specified file as the current working file.

### 5.7.2 CatStatus

**UINT16  
CatStatus (CatFileStatus \****Status***)**;

|  |  |  |
| --- | --- | --- |
| *NumBytes* | [out] | The number of bytes written. |
| *Status*[out] | The status of the currently selected file. |  |
| *RETURN* | *The returned 16-bit unsigned value is a concatenation of the SW response bytes with SW1 as the high byte and SW2 as the low byte, so a successful execution would return 0x9000.* |  |

*CatStatus* returns the file status of the currently selected file as specified in 3GPP TS 31.101 [16].

### 5.7.3 CatGetCHVStatus

**void  
CatGetCHVStatus (BYTE CHVStatus[4]);**

|  |  |  |
| --- | --- | --- |
| *CHVStatus* | [out] | Updates the CHVStatus array with the statusof CHV1, CHV2, UNBLOCKCHV1, and UNBLOCKCHV2 with CHV1 at array element zero. |

*CatGetCHVStatus* returns the current CHV status values. The format of the returned bytes is specified in 3GPP TS 31.101 [16].

### 5.7.4 CatReadBinary

**UINT16  
CatReadBinary (DWORD** *Offset*,   
 **DWORD \****NumBytes,* **void \****Buffer***)**;

|  |  |  |
| --- | --- | --- |
| *Offset* | [in] | *The offset into the file.* |
| *NumBytes* | [in/out] | The number of bytes to be read on input and the actual number read on output |
| *Buffer*[out] | The buffer into which the data is written. |  |
| *RETURN* | *The returned 16-bit unsigned value is a concatenation of the SW response bytes with SW1 as the high byte and SW2 as the low byte, so a successful execution would return 0x9000.* |  |

*CatReadBinary* reads *NumBytes* from position *Offset* in the currently selected file into *Buffer*.

### 5.7.5 CatUpdateBinary

**UINT16  
CatUpdateBinary (DWORD** *Offset*,   
 **DWORD** *NumBytes,* **const void \****Buffer***)**;

|  |  |  |
| --- | --- | --- |
| *Offset* | [in] | *The offset into the file.* |
| *NumBytes* | [in] | The number of bytes to write. |
| *Buffer*[in] | *The buffer containing the data to write to the file.* |  |
| *RETURN* | *The returned 16-bit unsigned value is a concatenation of the SW response bytes with SW1 as the high byte and SW2 as the low byte, so a successful execution would return 0x9000.* |  |

*CatUpdateBinary* writes *NumBytes* contained in *Buffer* to position *Offset* in the currently selected file.

### 5.7.6 CatReadRecord

**UINT16  
CatReadRecord (DWORD** *RecordNumber*,   
 **CatRecordAccessMode** *Mode*,   
 **DWORD Offset, DWORD \****NumBytes*,   
 **void \****Buffer***)**;

|  |  |  |
| --- | --- | --- |
| *RecordNumber* | [in] | *The record number from which to read when Mode is ABSOLUTE or 0 otherwise.* |
| *Mode*[in] | Indication of which record is to be read; viz. NEXT, PREVIOUS, CURRENT or ABSOLUTE. |  |
| *Offset* | [in] | The offset into the record. |
| *NumBytes* | [in/out] | The number of bytes to be read from the record on input and the number of bytes actually read on output |
| *Buffer*[out] | *The buffer into which the data is read.* |  |
| *RETURN* | *The returned 16-bit unsigned value is a concatenation of the SW response bytes with SW1 as the high byte and SW2 as the low byte, so a successful execution would return 0x9000.* |  |

*CatReadRecord* reads *NumBytes* from the record *RecordNumber* of the currently selected file into *Buffer*.

### 5.7.7 CatUpdateRecord

**UINT16  
CatUpdateRecord (DWORD** *RecordNumber*,   
 **CatRecordAccessMode** *Mode*,   
 **DWORD Offset, DWORD** *NumBytes*,   
 **const void \****Buffer***)**;

|  |  |  |
| --- | --- | --- |
| *RecordNumber* | [in] | *The record number to which to write when Mode is ABSOLUTE or 0 otherwise* |
| *Mode*[in] | Indication of which record is to be read; viz. NEXT, PREVIOUS, CURRENT or ABSOLUTE. |  |
| *Offset* | [in] | The offset into the record. |
| *NumBytes* | [in] | The number of bytes to write into the record. |
| *Buffer*[out] | *The buffer containing the data to write to the record.* |  |
| *RETURN* | *The returned 16-bit unsigned value is a concatenation of the SW response bytes with SW1 as the high byte and SW2 as the low byte, so a successful execution would return 0x9000.* |  |

*CatUpdateRecord* writes *NumBytes* into the record *RecordNumber* of the currently selected file from *Buffer*.

### 5.7.8 CatSearch

**UINT16  
CatSearch (CatSearchModes** *Mode*,   
 **DWORD Offset, DWORD** *PatternLength*, **const void \****Pattern***)**;

|  |  |  |
| --- | --- | --- |
| *Mode* | [in] | *Defines the seek method, One of SEEK\_FROM\_BEGINNING\_FORWARD, SEEK\_FROM\_END\_BACKWARD, SEEK\_FROM\_NEXT\_FORWARD, SEEK\_FROM\_PREVIOUS\_BACKWARD* |
| *Offset* | [in] | *The offset into the record at which to being pattern matching.* |
| *PatternLength* | [in] | The size in bytes of the pattern to search for. |
| *Pattern*[in] | *The buffer containing the pattern to search for.* |  |
| *RETURN* | *The returned 16-bit unsigned value is a concatenation of the SW response bytes with SW1 as the high byte and SW2 as the low byte, so a successful execution would return 0x9000.* |  |

*CatSearch* searches records in the currently selected file starting at Offset for the pattern of length *PatternLength* contained in *Pattern*.If the pattern is found the current record is set appropriately.

### 5.7.9 CatIncrease

**UINT16  
CatIncrease(DWORD** *Increment*,   
 **DWORD *\*****Value***)**;

|  |  |  |
| --- | --- | --- |
| *Increment* | [in] | *The value to increase by.* |
| *Value* | [out] | *The new value.* |
| *RETURN* | *The returned 16-bit unsigned value is a concatenation of the SW response bytes with SW1 as the high byte and SW2 as the low byte, so a successful execution would return 0x9000.* |  |

*CatIncrease* adds *Increment* to the current record of the selected cylic file and returns the new *Value.* The most significant byte of *Increment* is ignored.

### 5.7.10 CatInvalidate

**UINT16  
CatInvalidate (void);**

|  |  |  |
| --- | --- | --- |
| *RETURN* | *The returned 16-bit unsigned value is a concatenation of the SW response bytes with SW1 as the high byte and SW2 as the low byte, so a successful execution would return 0x9000.* |  |

*CatInvalidate* invalidates the selected file.

### 5.7.11 CatRehabilitate

**UINT16  
CatRehabilitate (void) ;**

|  |  |  |
| --- | --- | --- |
| *RETURN* | *The returned 16-bit unsigned value is a concatenation of the SW response bytes with SW1 as the high byte and SW2 as the low byte, so a successful execution would return 0x9000.* |  |

*CatRehabilitate* rehabilitates the selected file.

## 5.8 Miscellaneous

### 5.8.1 CatGetTerminalProfile

**void  
CatGetTerminalProfile (BYTE \****ProfileOutLength,* **BYTE \****Profile***)**;

|  |  |  |
| --- | --- | --- |
| *ProfileOutLength* | [out] | The number of bytes written to Profile. |
| *Profile* | [out] | The address at which the terminal profile is written. |

*CatGetTerminalProfile* returns the stored terminal profile in *Profile*.

### 5.8.2 CatMoreTime

**CatGeneralResult  
CatMoreTime (void);**

|  |  |  |
| --- | --- | --- |
| *RETURN* | *The GeneralResult code of the MORE TIME proactive command.* |  |

*CatMoreTime* issues the proactive command MORE TIME to the ME that it needs more time to process an application.

### 5.8.3 CatPollingOff

**CatGeneralResult  
CatPollingOff (void);**

|  |  |  |
| --- | --- | --- |
| *RETURN* | *The GeneralResult code of the POLLING OFF proactive command.* |  |

*CatPollingOff* issues the proactive command POLLING OFF that disables proactive polling; this essentially turns off *CatPollInterval*.

### 5.8.4 CatPollInterval

**CatGeneralResult  
CatPollInterval (CatTimeUnit** *Unit*, **BYTE** *Interval*,  
 **CatTimeInterval \****ActualIntervalOut***)**;

|  |  |  |
| --- | --- | --- |
| *Unit* | [in] | Desired time interval. Acceptable values for this parameter are listed in [CatTimeUnit](#_GsmTimeUnit). |
| *Interval* | [in] | Interval in *unit*s. |
| *ActualIntervalOut* | [out] | Response from ME negotiating the interval. This may or may not be the same as *Unit* and *Interval*. The value returned is in a structure of type [CatTimeInterval](#_GsmTimeInterval). |
| *RETURN* | *The GeneralResult code of the POLL INTERVAL proactive command.* |  |

*CatPollInterval* issues the proactive command POLL INTERVAL that requests the ME to set a time interval between status application program data units (APDU) that the ME sends to the UICC. The ME responds with a time interval of its own that most closely matches the application programming interface (API) request.

Polling can be disabled by using *CatPollingOff*.

### 5.8.5 CatRefresh

**CatGeneralResult  
CatRefresh (CatRefreshOptions** *Options*);

**CatGeneralResult  
CatRefreshWithFileList (CatRefreshOptions** *Options*,  
 **BYTE** *FileListLength*,  
 **const void \****FileList***)**;

|  |  |  |
| --- | --- | --- |
| *Options* | [in] | Informs the ME of what needs refreshing. Acceptable values for this parameter are listed in [CatRefreshOptions](#_GsmRefreshOptions). |
| *FileListLength* | [in] | The length, *in bytes*, of *FileList*. |
| *FileList* | [in] | The file identifiers of the files that have changed. |
| *RETURN* | *The GeneralResult code of the REFRESH proactive command.* |  |

*CatRefresh* issues the proactive command REFRESH that informs ME that the NAA has changed configuration due to UICC activity (such as an application running).

### 5.8.6 CatLanguageNotification

**void  
CatLanguageNotification (CatLanguageNotificationOptions** *Options*,  
 **const void \****Language***)**;

|  |  |  |
| --- | --- | --- |
| *Options* | [in] | Language options. An acceptable value for this parameter is a card reader device selected from [CatLanguageNotificationOptions](#_GsmLanguageNotificationOptions). |
| *Language* | [in] | *The 2-character language code as defined by ISO 639 [5], encoded using SMS default 7-bit coded alphabet as defined by TS 23.038 [6].* |
| *RETURN* | *The GeneralResult code of the LANGUAGE NOTIFICATION proactive command.* |  |

*CatLanguageNotification* issues the proactive command LANGUAGE NOTIFICATION that notifies the ME about the language currently used for any text string within proactive commands or envelope command responses.

### 5.8.7 CatLaunchBrowser

**CatGeneralResult  
CatLaunchBrowser (CatLaunchBrowserOptions** *Options*,  
 **BYTE** *TitleLength*, **const void \****Title*,  
 **BYTE** *URLLength*, **const void \****URL,*  **const CatIconIdentifier \****IconIdentifier***)**;

|  |  |  |
| --- | --- | --- |
| *Options*[in] | Options used to launch the browser. Acceptable values for this parameter are listed in [CatLaunchBrowserOptions](#_GsmLaunchBrowserOptions). |  |
| *TitleLength* | [in] | The length in bytes of the string Title |
| *Title* | [in] | String to display on the ME during the user confirmation phase. |
| URLLength | [in] | The length in bytes of URL. |
| *URL* | [in] | The URL to open the browser at. |
| *IconIdentifier* | [in] | Optional icon identifier; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If IconIdentifier is NULL or if IconIdentifier.UseIcon is zero, no icon identifier is sent to the ME. |
| *RETURN* | *The GeneralResult code of the LAUNCH BROWSER proactive command.* |  |

*CatLaunchBrowser* and *CatLaunchBrowserEx*  issue the proactive command LAUNCH BROWSER that launches a browser on the ME.

**CatGeneralResult  
CatLaunchBrowserEx (const CatLaunchBrowserExParams \**params*);**

The structure *CatLaunchBrowserExParams* has the following members:

typedef struct  
{  
 // Mandatory fields  
 CatLaunchBrowserOptions Options,  
 BYTE URLLength;  
 const void \*URL;

// Optional fields  
 BYTE BrowserIdentityLength;  
 const void \*BrowserIdentity;  
 BYTE BearerLength;  
 const BYTE \*Bearer;   
 BYTE NumProvisioningFileReferences;  
 BYTE \*ProvisioningFileReferenceLengths;  
 const BYTE \*\*ProvisioningFileReferences;  
 BYTE GatewayProxyIdLength;  
 const void \* GatewayProxyId;  
 CatAlphaString Title;  
 CatIconIdentifier IconIdentifier;   
} CatLaunchBrowerExParams;

with the following members:

|  |  |  |
| --- | --- | --- |
| URLLength[in] | The length in bytes of URL. |  |
| *URL*[in] | The URL to open the browser at. |  |
| *BrowserIdentityLength* | [in] | Length in bytes of *BrowserIdentity*. |
| *BrowserIdentity* | [in] | *The browser identity. If BrowserIdentity is NULL, no BROWSER IDENTITY TLV is sent to the ME.* |
| *BearerLength* | [in] | Length in bytes of *Bearer*. |
| *Bearer* | [in] | The list of bearers in order of priority requested. The type [CatBearer](#_GsmBearer) defines the values acceptable. If *Bearer* is NULL, no BEARER TLV is sent to the ME. |
| *NumProvisioningFileReferences* | [in] | The number of Provisioning File References. |
| *ProvisioningFileReferenceLengths* | [in] | A pointer to the array of Provisioning File References lengths. |
| *ProvisioningFileReferences* | [in] | A pointer to the array of Provisioning File References. |
| *GatewayProxyIdLength* | [in] | Length in bytes of *GatewayProxyId*. |
| *GatewayProxyId* | [in] | The gateway or proxy identity. If *GatewayProxyId* is NULL, no TEXT STRING TLV describing the gateway/proxy is sent to the ME. |
| *Title* | [in] | String to display on the ME; see [CatAlphaString](#_SimAlphaString). |
| *IconIdentifier* | [in] | Optional icon identifier; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If *IconIdentifier*.*UseIcon* is zero, no icon identifier is sent to the ME. |

## 5.9 Low-level Interface

This subclause presents a low-level programming interface that allows you to:

- Construct proactive commands and send them to the ME.

- Access the terminal response from the ME.

- Search the terminal response and contents of envelopes for specified TLVs.

- Unpack the contents of envelopes from the ME and send responses.

These functions are provided so that functionality that is not provided in the high level API is still accessible. All of these functions work on a single data buffer that has a single data pointer and can only be accessed sequentially. The high-level proactive functions may make use of the data buffer so consequently the high-level proactive functions should not be used whilst using the low-level functions.

### 5.9.1 CatResetBuffer

**void  
CatResetBuffer(void);**

This function resets the data pointer to the beginning of the buffer.

### 5.9.2 CatStartProactiveCommand

**void  
CatStartProactiveCommand(BYTE** *Command*,  
 **BYTE** *Options*,  
 **BYTE** *To***)**;

|  |  |  |
| --- | --- | --- |
| *Command* | [in] | Command byte of proactive command. |
| *Options* | [in] | Command options of proactive command. |
| *To* | [in] | The destination device identity. |

*CatStartProactiveCommand* resets the data pointer and starts the construction of a proactive command by writing the command tag, command details and device identities to the data buffer. The data pointer is left pointing after the device identities so that proactive command specific data can be written.

### 5.9.3 CatSendProactiveCommand

**CatGeneralResult  
CatSendProactiveCommand (BYTE \****Length***)**;

|  |  |  |
| --- | --- | --- |
| *Length* | [out] | Pointer that is updated with the length of the terminal response |
| *RETURN* |  | The general result byte of the terminal response |

*CatSendProactiveCommand* sends the contents of the data buffer as a proactive command and updates the data buffer with the terminal response. The general result byte of the terminal response is returned by this function. The length of the terminal response is written to \*Length. The data pointer is set to point to the additional information of the terminal response.

### 5.9.4 CatOpenEnvelope

**CatEnvelopeTagType  
CatOpenEnvelope(BYTE \****Length***)**;

|  |  |  |
| --- | --- | --- |
| *Length* | [out] | Pointer that is updated with the length of the envelope |
| *RETURN* |  | The envelope tag |

*CatOpenEnvelope* returns the envelope tag of the data buffer and the length of the envelope data. The data pointer is set to point to the envelope data.

### 5.9.5 CatSendEnvelopeResponse

**void  
CatSendEnvelopeResponse (void);**

*CatSendEnvelopeResponse* sends the contents of the data buffer as a successful envelope response.

### 5.9.6 CatSendEnvelopeErrorResponse

**void  
CatSendEnvelopeErrorResponse (void);**

This function sends the contents of the data buffer as an unsuccessful envelope response.

### 5.9.7 CatPutData

**void  
CatPutData(BYTE** *Length*,   
 **const void \****Data***)**;

|  |  |  |
| --- | --- | --- |
| *Length* | [in] | Length of Data |
| *Data* | [in] | Pointer to Data |

*CatPutData* appends Length bytes of data to the data buffer.

### 5.9.8 CatPutByte

**void  
CatPutByte (BYTE** *Data***)**;

|  |  |  |
| --- | --- | --- |
| *Data* | *[in]* | Data byte. |

*CatPutByte* appends the supplied data byte to the data buffer.

### 5.9.9 CatPutTLV

**void  
CatPutTLV (BYTE** *Tag*,   
 **BYTE** *Length*,   
 **const void \****Value***)**;

|  |  |  |
| --- | --- | --- |
| *Tag* | [in] | *Tag byte.* |
| *Length* | [in] | Length of value. |
| *Value* | [in] | A pointer to the value. |

*CatPutTLV*  appends a general TLV to the data buffer.

### 5.9.10 CatPutBytePrefixedTLV

**void  
CatPutBytePrefixedTLV (BYTE** *Tag*,   
 **BYTE** *Prefix*,  
 **BYTE** *Length*,   
 **const void \****Value***)**;

|  |  |  |
| --- | --- | --- |
| *Tag* | [in] | Tag byte. |
| *Prefix*[in] | Prefix byte. |  |
| *Length* | [in] | Length of value. |
| *Value* | [in] | A pointer to the value. |

*CatPutBytePrefixedTLV*  appends a TLV to the data buffer with a single byte placed before the Value.

### 5.9.11 CatPutOneByteTLV

**void  
CatPutOneByteTLV (BYTE** *Tag*,   
 **BYTE** *Value***)**;

|  |  |  |
| --- | --- | --- |
| *Tag* | [in] | *Tag byte.* |
| *Value* | [in] | Value byte. |

*CatPutOneByteTLV* appends a single byte valued TLV to the data buffer.

### 5.9.12 CatPutTwoByteTLV

**void  
CatPutTwoByteTLV (BYTE** *Tag*,   
 **BYTE** *Value1*,  
 **BYTE** *Value2***)**;

|  |  |  |
| --- | --- | --- |
| *Tag* | [in] | Tag byte. |
| *Value1* | [in] | First Value byte. |
| *Value2* | [in] | Second Value byte. |

*CatPutTwoByteTLV* appends a two byte valued TLV to the data buffer.

### 5.9.13 CatGetByte

**BYTE  
CatGetByte (void);**

|  |  |  |
| --- | --- | --- |
| *RETURN* |  | Data byte. |

*CatGetByte*  returns the byte at the current data pointer and increments the data pointer by one.

### 5.9.14 CatGetData

**const void \*  
CatGetData (BYTE** *Length***)**;

|  |  |  |
| --- | --- | --- |
| *Length* | [in] | Length of Data |
| *RETURN* |  | Pointer to Data. |

*CatGetData*  returns the current data pointer and increments the data pointer by *Length* bytes.

### 5.9.15 CatFindNthTLV

**const void \*  
CatFindNthTLV (BYTE** *Tag*,   
 **BYTE** *Occurrence*,   
 **BYTE \****Length***)**;

|  |  |  |
| --- | --- | --- |
| *Tag* | [in] | Tag to find. |
| *Occurrence* | [in] | Occurrence of Tag to find with "1" being the first. |
| *Length* | [out] | Length of found TLV. |
| *RETURN* |  | Pointer to data of found TLV |

*CatFindNthTLV*  finds the nth TLV that matches Tag in the data buffer, where nth is specified by the *Occurrence* parameter. If a match is found the data pointer is updated to the found TLV, the function returns a pointer to the found value and updates *Length* with the data length. If no match was found the function returns the null pointer and the data pointer is left unchanged.

### 5.9.16 CatFindNthTLVInUserBuffer

**const void \*  
CatFindNthTLVInUserBuffer (BYTE** BufferLen**,   
 const void \***Buffer**,   
 BYTE** *Tag***,   
 BYTE** *Occurrence***,   
 BYTE \****Length***);**

|  |  |  |
| --- | --- | --- |
| *BufferLen* | [in] | Length of buffer |
| *Buffer* | [in] | Buffer to search |
| *Tag* | [in] | Tag to find. |
| *Occurrence* | [in] | Occurrence of Tag to find with "1" being the first. |
| *Length* | [out] | Length of found TLV. |
| *RETURN* |  | Pointer to data of found TLV |

*CatFindNthTLVInUserBuffer*  finds the nth TLV that matches *Tag* is the supplied buffer. The function returns a pointer to the found value and updates *Length* with the data length. If no match was found the function returns the null pointer.

## 5.10 Network Services

### 5.10.1 CatGetLocationInformation

**CatGeneralResult  
CatGetLocationInformation (CatLocationInformation \****LocationInformation***);**

|  |  |  |
| --- | --- | --- |
| *LocationInformation* | [out] | A pointer to where the location information from the ME is placed. Refer to the [CatLocalInformation](#_SimLocationInformation) section for member details. |
| *RETURN* |  | The GeneralResult code of the PROVIDE LOCAL INFORMATION proactive command. |
|  |  | The GeneralResult code of the DISPLAY TEXT proactive command. |

*CatProvideLocationInformation* requests the ME to send location information to the (U)SIM using the PROVIDE LOCAL INFORMATION proactive command.

### 5.10.2 CatGetTimingAdvance

**CatGeneralResult  
CatGetTimingAdvance (CatTimingAdvance \****TimingAdvance***);**

|  |  |  |
| --- | --- | --- |
| *TimingAdvance* | [out] | A pointer to where the timing advance information from the ME is placed. Refer to the [CatTimingAdvance](#_SimTimingAdvance) section for member details. |
| *RETURN* |  | The GeneralResult code of the PROVIDE LOCAL INFORMATION proactive command. |

*CatProvideTimingAdvance* requests the ME to send timing advance information to the (U)SIM using the PROVIDE LOCAL INFORMATION proactive command.

### 5.10.3 CatGetIMEI

**CatGeneralResult  
CatGetIMEI (BYTE** *IMEI***[8]);**

|  |  |  |
| --- | --- | --- |
| *IMEI* | [out] | A pointer to where the IMEI of the ME is placed. |
| *RETURN* |  | The GeneralResult code of the PROVIDE LOCAL INFORMATION proactive command. |

*CatGetIMEI* requests the ME to send the IMEI to the (U)SIM using the PROVIDE LOCAL INFORMATION proactive command.

### 5.10.4 CatGetNetworkMeasurementResults

**CatGeneralResult  
CatGetNetworkMeasurementResults (BYTE** *MeasurementResults***[10]);**

|  |  |  |
| --- | --- | --- |
| *MeasurementResults* | [out] | A pointer to where the network measurement results from the ME is placed. |
| *RETURN* |  | The GeneralResult code of the PROVIDE LOCAL INFORMATION proactive command. |

*CatGetNetworkMeasurementResults*  requests the ME to send the network measurement results to the (U)SIM using the PROVIDE LOCAL INFORMATION proactive command.

### 5.10.5 CatGetDateTimeAndTimeZone

**CatGeneralResult  
CatGetDateTimeAndTimeZone (BYTE** *DateTimeAndTimeZone***[7]);**

|  |  |  |
| --- | --- | --- |
| *DateTimeAndTimeZone* | [out] | A pointer to where the date, time, and time zone from the ME is placed. |
| *RETURN* |  | The GeneralResult code of the PROVIDE LOCAL INFORMATION proactive command. |

*CatGetDateTimeAndTimeZones* requests the ME to send the date, time, and time zone information to the (U)SIM using the PROVIDE LOCAL INFORMATION proactive command.

### 5.10.6 CatGetLanguage

**CatGeneralResult  
CatGetLanguage (BYTE** *Language***[2]);**

|  |  |  |
| --- | --- | --- |
| *DateTimeAndTimeZone* | [out] | A pointer to where the language from the ME is placed. |
| *RETURN* |  | The GeneralResult code of the PROVIDE LOCAL INFORMATION proactive command. |

*CatGetLanguage* requests the ME to send the language information to the (U)SIM using the PROVIDE LOCAL INFORMATION proactive command.

### 5.10.7 CatSetupCall

**CatGeneralResult  
CatSetupCall (BYTE** *CallSetupMessageLength***, const void \****CallSetupMessage***,   
 CatTypeOfNumberAndNumberingPlanIdentifier** *TONandNPI***,   
 BYTE** *DiallingNumberLength***, const void \****DiallingNumber***,   
 CatSetupCallOptions** *Options,* **const CatIconIdentifier \****UserConfirmationIconIdentifier***,  
 BYTE** *CallSetupMessageLength***, const void \****CallSetupMessage***,  
 const CatIconIdentifier \****CallSeupIconIdentifier***);**

|  |  |  |
| --- | --- | --- |
| *UserConfirmationMessageLength* | [in] | Length in bytes of UserConfirmationMessage. |
| *UserConfirmationMessage* | [in] | Message to display for user confirmation or NULL. |
| *TONandNPI* | [in] | Acceptable values for this parameter are listed in [CatTypeOfNumberAndNumberingPlanIdentifier](#_GsmTypeOfNumberAndNumberingPlanIdentifi). |
| *DiallingNumberLength* | [in] | Length in bytes of DiallingNumber. |
| *DialingNumber* | [in] | Number to call is coded as binary-coded decimal. |
| *Options* | [in] | Acceptable values for this parameter are listed in [CatSetupCallOptions](#_GsmSetupCallOptions). |
| *UserConfirmationIconIdentifier* | [in] | Optional icon identifier to use during the user confirmation phase; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If UserConfirmationIconIdentifier is NULL or if UserConfirmationIconIdentifier.UseIcon is zero, no user confirmation phase icon identifier is sent to the ME. |
| *CallSetupMessageLength* | [in] | Length in bytes of CallSetupMessage. |
| *CallSetupMessage* | [in] | Message to display for call set up or NULL. |
| *CallSetupIconIdentifier* | [in] | Optional icon identifier to use during the call setup phase; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If CallSetupIconIdentifier is NULL or if CallSetupIconIdentifier.UseIcon is zero, no call setup phase icon identifier is sent to the ME. |
| *RETURN* |  | The GeneralResult code of the SET UP CALL proactive command. |

*CatSetupCall* and *CatSetupCallEx* issue the SET UP CALL proactive command to the ME.

**CatGeneralResult  
CatSetupCallEx (const CatSetupCallExParams \****Params***);**

The type *CatSetupCallExParams* is defined as follows:

typedef struct  
{  
 // Mandatory fields  
 CatSetupCallOptions Options;  
 CatTypeOfNumberAndNumberingPlanIdentifier TONandNPI;  
 BYTE DiallingNumberLength;  
 const void \*DialingNumber;

// Optional fields  
 CatAlphaString UserConfirmationMessage;  
 BYTE CapabilityConfigParamsLength;  
 const void \*CapabilityConfigParams;  
 BYTE CalledPartySubaddressLength;  
 const void \*CalledPartySubaddress;  
 CatTimeInterval RedialMaximumDuration;  
 CatIconOption UserConfirmationIcon;  
 CatAlphaString CallSetupMessage;  
 CatIconOptions CallSetupIcon;   
} CatSetupCallExParams;

With the following members:

|  |  |
| --- | --- |
| *Options* | Acceptable values for this parameter are listed in [CatSetupCallOptions](#_GsmSetupCallOptions). |
| *TONandNPI* | Acceptable values for this parameter are listed in [CatTypeOfNumberAndNumberingPlanIdentifier](#_GsmTypeOfNumberAndNumberingPlanIdentifi). |
| *DiallingNumberLength* | Length in bytes of DiallingNumber. |
| *DialingNumber* | Number to call is coded as binary-coded decimal. |
| *UserConfirmationMessage* | String to display during the user confirmation phase; see [CatAlphaString](#_SimAlphaString). If this parameter is null, no user confirmation message TLV is passed to the ME. |
| *CapabilityConfigParamsLength* | Length in bytes of CapabilityConfigParams. |
| *CapabilityConfigParams* | A pointer to the capability configuration parameters as coded for EFCCP. |
| *CalledPartySubaddressLength* | Length in bytes of CalledPartySubaddress. |
| *CalledPartySubaddress* | The called party subaddress. |
| *RedialMaximumDuration* | An optional maximum duration for the redial mechanism. If the timeInterval member of this structure is zero, no duration TLV is sent to the ME. |
| *UserConfirmationIcon* | *The icon to display during the user confirmation phase. If the UseIcon member of this structure is zero, no user confirmation icon TLV is sent to the ME.* |
| *CallSetupMessage* | String to display during the call set up phase; see [CatAlphaString](#_SimAlphaString). |
| *CallSetupIcon* | The icon to display during the call setup phase. |

Optional parameters are specifically chosen to use an all-zero binary representation. This means that it is simple to set up only the required members of the *SetupCallExParams* structure by zeroing the whole structure using *memset*, filling in the required members, and sending the result to *CatSetupCallEx*. As all optional parameters use a zero binary representation, the *memset* serves to *initialise* them all to the "not present" status.

### 5.10.8 CatSendShortMessage

**CatGeneralResult  
CatSendShortMessage (BYTE** *TitleLength***, const void \****Title***,   
 CatTypeOfNumberAndNumberingPlanIdentifier** *TONandNPI***,   
 BYTE** *AddressLength***, const void \****Address***,   
 BYTE** *SmsTPDULength***, const void \****SmsTPDU***,  
 CatSendShortMessageOptions** *Options***,  
 const CatIconIdentifier \****IconIdentifier***);**

|  |  |  |
| --- | --- | --- |
| *TitleLength* | [in] | Length in bytes of Title. |
| *Title* | [in] | String to display while ME is sending a message. |
| *TONandNPI* | [in] | Acceptable values for this parameter are listed in [CatTypeOfNumberAndNumberingPlanIdentifier](#_GsmTypeOfNumberAndNumberingPlanIdentifi). |
| *AddressLength* | [in] | Length in bytes of Address. |
| *Address* | [in] | Address of the service center where message is being sent. |
| *SmsTPDULength* | [in] | Length in bytes of SmsTPDU. |
| *SmTPDU* | [in] | Formatted short message service (SMS) message to send. |
| *Options* | [in] | Specifies who packs the message. Acceptable values for this parameter are listed in [CatSendShortMessageOptions](#_GsmSendShortMessageOptions). |
| *IconIdentifier* | [in] | Optional icon identifier; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If IconIdentifier is NULL or if IconIdentifier.UseIcon is zero, no icon identifier is sent to the ME. |
| *RETURN* |  | The GeneralResult code of the SEND SHORT MESSAGE proactive command. |

*CatSendShortMessage* issues the SEND SHORT MESSAGE proactive command.

### 5.10.9 CatSendSS

**CatGeneralResult  
CatSendSS (BYTE** *TitleLength***, const void \****Title***,   
 CatTypeOfNumberAndNumberingPlanIdentifier** *TONandNPI***,   
 BYTE** *SSStringLength***, const void \****SSString***,  
 const CatIconIdentifier \****IconIdentifier***);**

|  |  |  |
| --- | --- | --- |
| *TitleLength* | [in] | Length in bytes of Title. |
| *Title* | [in] | String to display while ME is sending a message. |
| *TONandNPI* | [in] | Acceptable values for this parameter are listed [CatTypeOfNumberAndNumberingPlanIdentifier](#_GsmTypeOfNumberAndNumberingPlanIdentifi). |
| *SSStringLength* | [in] | Length in bytes of SSString. |
| *SSString* | [in] | SS string to ME. |
| *IconIdentifier* | [in] | Optional icon identifier; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If IconIdentifier is NULL or if IconIdentifier.UseIcon is zero, no icon identifier is sent to the ME. |
| *RETURN* |  | The GeneralResult code of the SEND SS proactive command. |

*CatSendSS* issues the SEND SS proactive command to the ME.

### 5.10.10 CatSendUSSD

**CatGeneralResult  
CatSendUSSD (BYTE** *TitleLength***, const void \****Title***,   
 CatDCSValue** *MessageDCS***, BYTE** *MessageLength***, const void \****Message***,  
 CatDCSValue \****MsgOutDCS***, BYTE \****MsgOutLength***, void \****MsgOut***,  
 const CatIconIdentifier \****IconIdentifier***);**

|  |  |  |
| --- | --- | --- |
| TitleLength | [in] | The length in bytes of Title. |
| Title | [in] | String to display while ME is sending a message. |
| MessageDCS | [in] | The data-coding scheme for Message. Acceptable values for this parameter are listed in [CatDCSValue](#_GsmDCSValue). |
| MessageLength | [in] | The length in bytes of Message. |
| *Message* | [in] | Message to send. |
| *MsgOutDCS* | [out] | *Identifies type of DCS for the returned message.* |
| *MsgOutLength* | [out] | Length of the returned message in bytes. |
| *MsgOut* | [out] | Returned string or message. |
| *IconIdentifier* | [in] | Optional icon identifier; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If IconIdentifier is NULL or if IconIdentifier.UseIcon is zero, no icon identifier is sent to the ME. |
| *RETURN* |  | The GeneralResult code of the SEND USSD proactive command. |

*CatSendUSSD* issues the SEND USSD proactive command. The terminal response is parsed and if successful the MsgOutDCS, MsgOutLength and MsgOut parameters are updated.

### 5.10.11 CatOpenCSChannel

**CatGeneralResult  
CatOpenCSChannel(CatOpenChannelOptions** *Options***,  
 BYTE** *UserConfirmationLength***, const void \****UserConfirmation***,  
 const** CatIconIdentifier **\****UserConfimationIconIdentifier***,  
 CatTypeOfNumberAndNumberingPlanIdentifier** *TONandNPI***,   
 BYTE** *DiallingNumberLength***, const void \****DiallingNumber***,  
 BYTE** *BearerDescription[3]***,  
 UINT16 \****BufferSize***,  
 CatDevice \****ChannelIdentifier***);**

|  |  |  |
| --- | --- | --- |
| *Options* | [in] | Acceptable values for this parameter are listed in [CatOpenChannelOptions](#_GsmSetupCallOptions). |
| *UserConfirmationLength* | [in] | Length in bytes of UserConfirmation. |
| *UserConfirmation* | [in] | String to display when ME alerts user that channel is to be opened. |
| *UserConfirmationIconIdentifier* | [in] | Optional icon identifier to use during the user confirmation phase; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If UserConfirmationIconIdentifier is NULL or if UserConfirmationIconIdentifier.UseIcon is zero, no user confirmation phase icon identifier is sent to the ME. |
| *TONandNPI* | [in] | Acceptable values for this parameter are listed in [CatTypeOfNumberAndNumberingPlanIdentifier](#_GsmTypeOfNumberAndNumberingPlanIdentifi). |
| *DiallingNumberLength* | [in] | Length in bytes of DiallingNumber. |
| *DialingNumber* | [in] | Number to call is coded as binary-coded decimal. |
| *BearerDescription* | [in/out] | Initially contains the bearer description parameters (data rate, bearer service and connection element) and is modified to the actual bearer description as allocated by the ME. |
| *BufferSize* | [in/out] | Initially contains the desired buffer size and is modified to the actual buffer size as allocated by the ME. |
| *ChannelIdentifier* | [out] | The channel identifier that has been allocated by the ME. |
| *RETURN* |  | The GeneralResult code of the OPEN CHANNEL proactive command. |

**CatGeneralResult  
CatOpenCSChannelEx(const CatOpenCSChannelExParams \****Params*,  
 **CatDevice \****ChannelIdentifier*,  
 **BYTE** *BearerDescription[3]*,  
 **UINT16 \****BufferSize***)**;

|  |  |  |
| --- | --- | --- |
| *Params* | [in] | Constant parameter set as defined below. |
| *ChannelIdentifier* | [out] | The channel identifier that has been allocated by the ME. |
| *BearerDescription* | [out] | An array to which the actual bearer description allocated by the ME will be written. |
| *BufferSize* | [out] | The actual buffer size allocated by the ME. |
| *RETURN* |  | The GeneralResult code of the OPEN CHANNEL proactive command. |

*CatOpenCSChannel* and *CatOpenCSChannelEx* issue the proactive command OPEN CHANNEL related to a CS bearer. The terminal response is parsed and if the command was successful the *BearerDescription*, *BufferSize* and *ChannelIdentifier* parameters are updated.

The type *CatOpenCSChannelExParams* is defined as follows:

typedef struct   
{   
 // Mandatory fields  
 CatOpenChannelOptions Options;  
 BYTE AddressLength;  
 const BYTE \*Address;  
 BYTE BearerDescription[3];  
 UINT16 BufferSize;

// Optional fields  
 CatAlphaString UserConfirmationMessage; CatIconIdentifier UserConfirmationIconIdentifier;   
 BYTE SubAddressLength; const BYTE \*SubAddress;  
 BYTE Duration1Defined;  
 CatTimeInterval Duration1;  
 BYTE Duration2Defined;  
 CatTimeInterval Duration2;  
 CatAddressType LocalAddress;  
 CatTextString UserLogin;   
 CatTextString UserPassword;   
 CAT\_MEInterfaceTransportLevelType CAT\_MEInterfaceTransportLevel;  
 CatAddressType DataDestinationAddress;  
} CatOpenCSChannelExParams;

With the following members:

|  |  |
| --- | --- |
| *Options* | Acceptable values for this parameter are listed in [CatOpenChannelOptions](#_GsmSetupCallOptions). This field is mandatory. |
| *AddressLength* | Length in bytes of Address. This field is mandatory. |
| *Address* | *The address to call. This field is mandatory.* |
| *BearerDescription* | *The desired bearer parameters (data rate, bearer service and connection element). This field is mandatory.* |
| *BufferSize* | The desired buffer size. This field is mandatory. |
| *UserConfirmationMessage* | String to display during the user confirmation phase; see [CatAlphaString](#_SimAlphaString). If this parameter is null, no user confirmation message TLV is passed to the ME. If UserConfirmationMessage is not null but UserConfirmationMessageLength is zero, a user confirmation message TLV is passed to the ME with the length component set to zero. |
| *UserConfirmationIconIdentifier* | *The icon to display during the user confirmation phase. If the UseIcon member of this structure is zero, no user confirmation icon TLV is sent to the ME.* |
| *SubAddressLength* | Length in bytes of SubAddress. |
| *SubAddress* | The subaddress to call. |
| *Duration1Defined* | Set to nonzero if Duration1 is defined. |
| *Duration1* | *Duration of reconnect tries; see* [*CatTimeInterval*](#_SimTimeInterval)*.* |
| *Duration2Defined* | *Set to nonzero if Duration2 is defined.* |
| *Duration2* | *Duration of timeout; see* [*CatTimeInterval*](#_GsmTimeInterval)*.* |
| *LocalAddress* | *The LocalAddress; see* [*CatAddressType*](#_SimAddressType)*.* |
| *UserLogin* | *The user login string.* |
| *UserPassword* | The user password string. |
| *CAT\_MEInterfaceTransportLevel* | See [CAT\_MEInterfaceTransportLevelType](#_SimSIM_MEInterfaceTransportLevelType). |
| *DataDestinationAddress* | The DataDestinationAddress; see [CatAddressType](#_SimAddressType). |

### 5.10.12 CatOpenGPRSChannel

**CatGeneralResult  
CatOpenGPRSChannel(CatOpenChannelOptions** *Options***,  
 BYTE** *UserConfirmationLength***, const void \****UserConfirmation***,  
 const CatIconIdentifier \****UserConfirmationIconIdentifier***,   
 BYTE** *BearerDescription[8]***,   
 UINT16 \****BufferSize***,   
 CatDevice \****ChannelIdentifier***);**

|  |  |  |
| --- | --- | --- |
| *Options* | [in] | Acceptable values for this parameter are listed in [CatOpenChannelOptions](#_GsmSetupCallOptions). |
| *UserConfirmationLength* | [in] | Length in bytes of UserConfirmation. |
| *UserConfirmation* | [in] | String to display when ME alerts user that channel is to be opened. |
| *UserConfirmationIconIdentifier* | [in] | Optional icon identifier to use during the user confirmation phase; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If *UserConfirmationIconIdentifier* is NULL or if *UserConfirmationIconIdentifier*.UseIcon is zero, no user confirmation phase icon identifier is sent to the ME. |
| *BearerDescription* | [in/out] | Initially contains the bearer description and is modified to the actual bearer description as allocated by the ME. |
| *BufferSize* | [in/out] | Initially contains the desired buffer size and is modified to the actual buffer size as allocated by the ME. |
| *ChannelIdentifier* | [out] | The channel identifier that has been allocated by the ME. |
| *RETURN* |  | The GeneralResult code of the OPEN CHANNEL proactive command. |

**CatGeneralResult  
CatOpenGPRSChannelEx(const CatOpenGPRSChannelExParams \****Params***,  
 CatDevice \****ChannelIdentifier***,  
 BYTE** *ActualBearerDescription[8]*, **UINT16 \****ActualBufferSize***);**

|  |  |  |
| --- | --- | --- |
| *Params* | [in] | Constant parameter set as defined below. |
| *ChannelIdentifier* | [out] | The channel identifier that has been allocated by the ME. |
| *ActualBearerDescription* | [out] | An array to which the actual bearer description allocated by the ME will be written. |
| *ActualBufferSize* | [out] | The actual buffer size allocated by the ME. |
| *RETURN* |  | The GeneralResult code of the OPEN CHANNEL proactive command. |

*CatOpenGPRSChannel* and *CatOpenGPRSChannelEx* issue the proactive command OPEN CHANNEL related to a GPRS bearer. The terminal response is parsed and if the command was successful the *BearerDescription*, *BufferSize* and *ChannelIdentifier* parameters are updated.

The type *CatOpenGPRSChannelExParams* is defined as follows:

typedef struct   
{  
 // Mandatory fields  
 GsmOpenChannelOptions Options;  
 BYTE AddressLength;  
 const BYTE \*Address;  
 BYTE BearerDescription[8];  
 UINT16 BufferSize;

// Optional fields  
 CatAlphaString UserConfirmationMessage;  
 CatIconIdentifier UserConfirmationIconIdentifier;   
 BYTE AccessPointNameLength;  
 const BYTE \*AccessPointName;  
 CatAddressType LocalAddress;  
 CAT\_ME\_InterfaceTransportLevelType CAT\_ME\_InterfaceTransportLevel;  
 CatAddressType DataDestinationAddress;  
} GsmOpenGPRSChannelExParams;

With the following members:

|  |  |
| --- | --- |
| *Options* | Acceptable values for this parameter are listed in [CatOpenChannelOptions](#_GsmSetupCallOptions). This field is mandatory. |
| *AddressLength* | Length in bytes of Address. This field is mandatory. |
| *Address* | *The address to call. This field is mandatory.* |
| *BearerDescription* | *The desired bearer. This field is mandatory.* |
| *BufferSize* | The desired buffer size. This field is mandatory. |
| *UserConfirmationMessage* | String to display during the user confirmation phase; see [CatAlphaString](#_SimAlphaString). If this parameter is null, no user confirmation message TLV is passed to the ME. If UserConfirmationMessage is not null but UserConfirmationMessageLength is zero, a user confirmation message TLV is passed to the ME with the length component set to zero. |
| *UserConfirmationIconIdentifier* | The icon to display during the user confirmation phase. If the UseIcon member of this structure is zero, no user confirmation icon TLV is sent to the ME. |
| *AccessPointNameLength* | The length in bytes of AccessPoint. |
| *AccessPointName* | The Access Point Name. |
| *LocalAddress* | See [CatAddressType](#_SimAddressType). |
| *CAT\_ME\_InterfaceTransportLevel* | See [CAT\_MEInterfaceTransportLevelType](#_SimSIM_MEInterfaceTransportLevelType). |
| *DataDestinationAddress* | See [CatAddressType](#_SimAddressType). |

### 5.10.13 CatCloseChannel

**CatGeneralResult  
CatCloseChannel (CatDevice** *ChannelIdentifier*,  
 **BYTE** *TitleLength***, const void \****Title****,*  
 const CatIconIdentifier \****IconIdentifier***);**

|  |  |  |
| --- | --- | --- |
| ChannelIdentifier[in] | The channel identifier as returned from one of the open commands |  |
| TitleLength | [in] | The length in bytes of Title. |
| *Title* | [in] | String to display while ME is closing the channel. |
| *IconIdentifier* | [in] | Optional icon identifier; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If *IconIdentifier* is NULL or if *IconIdentifier*.UseIcon is zero, no icon identifier is sent to the ME. |
| *RETURN* |  | The GeneralResult code of the CLOSE CHANNEL proactive command. |

*CatCloseChannel* issues a CLOSE CHANNEL proactive command that closes an open channel.

### 5.10.14 CatReceiveData

**CatGeneralResult  
CatReceiveData (CatDevice** *ChannelIdentifier*,  
 **BYTE** *TitleLength***, const void \****Title***,****BYTE** *RequestedChannelDataLength***,  
 const CatIconIdentifier \****IconIdentifier***,****BYTE \****ChannelData***,****BYTE \****NumChannelBytesRead,* **BYTE \****NumChannelBytesLeft***);**

|  |  |  |
| --- | --- | --- |
| ChannelIdentifier[in] | The channel identifier as returned from one of the open commands |  |
| TitleLength | [in] | The length in bytes of Title. |
| *Title* | [in] | String to display while ME is receiving data. |
| *RequestedChannelDataLength* | [in] | The number of bytes requested to be read. |
| *IconIdentifier* | [in] | Optional icon identifier; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If *IconIdentifier* is NULL or if *IconIdentifier*.UseIcon is zero, no icon identifier is sent to the ME. |
| *ChannelData* | [out] | Received channel data. |
| *NumChannelBytesRead* | [out] | *The number of bytes received as channel data.* |
| *NumChannelBytesLeft* | [out] | The number of bytes remaining to be read from the channel buffer, or 255 if there are more than 255 bytes left to be read. |
| *RETURN* |  | The GeneralResult code of the RECEIVE DATA proactive command. |

*CatReceiveData*  issues a RECEIVE DATA proactive command that receives data from an open channel. The terminal response is parsed and if the command is successful the received data is copied into the ChannelData array and the NumChannelBytesRead and NumChannelBytesLeft parameters are updated.

### 5.10.15 CatSendData

**CatGeneralResult  
CatSendData (CatDevice** *ChannelIdentifier*,  
 **CatSendDataOptions** *Options***,** **BYTE** *TitleLength***, const void \****Title***,  
 BYTE** *ChannelDataLength,* **const void \****ChannelData***,  
 const CatIconIdentifier \****IconIdentifier***,  
 BYTE \****ActualBytesSent***);**

|  |  |  |
| --- | --- | --- |
| ChannelIdentifier[in] | The channel identifier as returned from one of the open commands |  |
| TitleLength | [in] | The length in bytes of Title. |
| *Title* | [in] | String to display while ME is receiving data. |
| *Options* | [in] | Specifies who packs the message. Acceptable values for this parameter are listed in [CatSendDataOptions](#_GsmSendDataOptions). |
| *ChannelDataLength* | [in] | The number of bytes to be sent from *ChannelData*. |
| *ChannelData* | [in] | The data to be sent. |
| *IconIdentifier* | [in] | Optional icon identifier; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If *IconIdentifier* is NULL or if *IconIdentifier*.*UseIcon* is zero, no icon identifier is sent to the ME. |
| *ActualBytesSent* | [out] | The number of bytes sent (derived from the CHANNEL DATA LENGTH TLV in the TERMINAL RESPONSE). |
| *RETURN* |  | The GeneralResult code of the SEND DATA proactive command. |

*CatSendData* issues the proactive command SEND DATA that sends data to an open channel.

### 5.10.16 CatGetChannelStatus

**CatGeneralResult  
CatGetChannelStatus (CatDevice** *ChannelIdentifier*, **void \****ChannelStatus***);**

|  |  |  |
| --- | --- | --- |
| ChannelIdentifier[in] | The channel identifier. |  |
| *ChannelStatus* | [out] | Returned channel status bytes. |
| *RETURN* |  | The GeneralResult code of the GET CHANNEL STATUS proactive command. |

*CatGetChannelStatus* issues a proactive command GET CHANNEL STATUS. The terminal response is parsed if the command is successful to find the status of the supplied channel.

### 5.10.17 CatServiceSearch

**CatGeneralResult  
CatServiceSearch (CatBearer** *BearerId,* **BYTE** *AttributeLength***, void \****Attributes,***void \****ServiceAvailability***);**

|  |  |  |
| --- | --- | --- |
| *BearerId* | [in] | The identifier of the bearer whose services will be searched. |
| *AttributeLength* | [in] | The length of the following attribute array. |
| *Attributes* | [in] | Attributes that describe bearer services, typically in a bearer specific format. |
| *ServiceAvailability* | [in] | List of services offered by the bearer that satisfy the attributes, typically in a bearer specific format. |

*CatServiceSearch* searches for a particular service on a bearer.

### 5.10.18 CatGetServiceInformation

**CatGeneralResult  
CatGetServiceInformation (BYTE** *TitleLength***, const BYTE \****Title***,** **const catIconIdentifier \****IconIdentifier***,****CatBearer** *BearerId***,****BYTE \****AttributeLength, void \*Attributes,* **void \****ServiceInformation***);**

|  |  |  |
| --- | --- | --- |
| TitleLength | [in] | The length in bytes of Title. |
| *Title* | [in] | String to display acquiring service information. |
| *IconIdentifier* | [in] | Optional icon identifier; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If *IconIdentifier* is NULL or if *IconIdentifier*.*UseIcon* is zero, no icon identifier is sent to the ME. |
| *BearerId* | [in] | The identifier of the bearer whose service information is requested. |
| *AttributeLength* | [in] | *The number of bytes in the following attribute array.* |
| *Attributes* | [in] | Attributes describing the service information requested. |
| *ServiceInformation* | [out] | The requested information. |

*CatGetServiceInformation* retrieves information about a particular service on a bearer.

### 5.10.19 CatDeclareService

**CatGeneralResult  
CatDeclareService (CatBearer** *BearerId***, BYTE** *ServiceId***,****CatTransportProtocol** *TransportProtocol***,****WORD \****PortNumber***,****BYTE** *ServiceRecordLength,* **void \****ServiceRecord***);**

|  |  |  |
| --- | --- | --- |
| BearerId | [in] | The identifier of the bearer for which this service is being offered. |
| *TransportProtocol* | [in] | The transport protocol on which the service is provided. |
| *PortNumber* | [in] | The port on which the service is provided. |
| *ServiceRecordLength* | [in] | The number of bytes in the following service record. |
| *ServiceRecord* | [in] | The service record describing the service. |

*CatDeclareService* describes a new service.

### 5.10.20 CatRunATCommand

**CatGeneralResult  
CatRunATCommand (BYTE** *TitleLength***, const void \****Title***,   
 BYTE** *CommandLength***, const void \****Command***,  
 const CatIconIdentifier \****IconIdentifier***,  
 void \****Response***, BYTE \****ResponseLength***);**

|  |  |  |
| --- | --- | --- |
| *TitleLength* | [in] | Length in bytes of Title. |
| *Title* | [in] | String to display on ME while command is executing. |
| *CommandLength* | [in] | Length in bytes of Command. |
| *Command* | [in] | AT command string |
| *IconIdentifier* | [in] | Optional icon identifier; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If *IconIdentifier* is NULL or if *IconIdentifier*.*UseIcon* is zero, no icon identifier is sent to the ME. |
| *Response* | [out] | ME response string. |
| *ResponseLength* | [out] | *Length in bytes of ME response string.* |
| *RETURN* | *The GeneralResult code of the RUN AT COMMAND proactive command.* |  |

*CatRunATCommand* issues the proactive command RUN AT COMMAND that sends an AT command to the ME. The terminal response is parsed and if successful the parameters *Response* and *ResponseLength* are updated.

### 5.10.21 CatSendDTMFCommand

**CatGeneralResult  
CatSendDTMFCommand (BYTE** *TitleLength***, const void \****Title***,  
 BYTE** *DTMFCodeLength***, const void \****DTMFCode****,* const CatIconIdentifier \****IconIdentifier***);**

|  |  |  |
| --- | --- | --- |
| TitleLength | [in] | The length in bytes of Title. |
| *Title* | [in] | Title displayed while the DTMF string is sent to the network. |
| DTMFCodeLength | [in] | The length in bytes of DTMFCode. |
| *DTMFCode* | [in] | DTMF string sent to the network. |
| *IconIdentifier* | [in] | Optional icon identifier; see [CatIconIdentifier](#_GsmIconIdentifier) for member details. If IconIdentifier is NULL or if IconIdentifier.UseIcon is zero, no icon identifier is sent to the ME. |
| *RETURN* | *The GeneralResult code of the SEND DTMF COMMAND proactive command.* |  |

*CatSendDTMF* issues the proactive command SEND DTMF COMMAND that sends a dual tone multiple frequency (DTMF) string to the network.

## 5.11 Supporting Data Types

- typedef unsigned char BYTE.

- typedef unsigned short WORD.

- typedef unsigned long int DWORD.

### 5.11.1 CatRecordAccessMode

typedef enum {

NEXT = 0x02,

PREVIOUS = 0x03,

CURRENT = 0x04,

ABSOLUTE = 0x04

} CatRecordAccessMode;

### 5.11.2 CatSearchMode

typedef enum {

BEGINNING\_FORWARD,  
END\_BACKWARD,  
NEXT\_FORWARD,  
PREVIOUS\_BACKWARD

} CatSearchMode;

### 5.11.3 CatFrameworkEventType

typedef enum  
{  
 // Command monitoring events  
 EVENT\_TERMINAL\_PROFILE\_COMMAND,  
 EVENT\_STATUS\_COMMAND  
 EVENT\_ENVELOPE\_COMMAND,  
 // Application lifecycle events start here  
 EVENT\_APPLICATION\_LIFECYCLE\_INSTALL = 0x20  
 // Framework fabricated events start here  
 EVENT\_UPDATE\_EF\_SMS = 0x40

EVENT\_PROFILE\_DOWNLOAD,

EVENT\_FORMATTED\_SMS\_PP\_UPD,

EVENT\_STATUS\_COMMAND,

EVENT\_UNFORMATTED\_SMS\_PP\_UPD,

EVENT\_MENU\_SELECTION,

EVENT\_FORMATTED\_SMS\_PP\_ENV,

EVENT\_UNFORMATTED\_SMS\_PP\_ENV,

EVENT\_FORMATTED\_SMS\_PP\_CB,

EVENT\_MENU\_SELECTION\_HELP\_REQUEST,

EVENT\_CALL\_CONTROL\_BY\_SIM,

EVENT\_MO\_SHORT\_MESSAGE\_CONTROL\_BY\_SIM,

EVENT\_TIMER\_EXPIRATION,

EVENT\_DOWNLOAD\_MT\_CALL\_EVENT,

EVENT\_DOWNLOAD\_CALL\_CONNECTED\_EVENT,

EVENT\_DOWNLOAD\_CALL\_DISCONNECTED\_EVENT,

EVENT\_DOWNLOAD\_LOCATION\_STATUS\_EVENT,

EVENT\_DOWNLOAD\_USER\_ACTIVITY\_EVENT,

EVENT\_DOWNLOAD\_IDLE\_SCREEN\_AVAILABLE\_EVENT,

EVENT\_DOWNLOAD\_CARD\_READER\_STATUS\_EVENT,

EVENT\_DOWNLOAD\_LANGUAGE\_SELECTION\_EVENT,

EVENT\_DOWNLOAD\_BROWSER\_TERMINATION\_EVENT,

EVENT\_DOWNLOAD\_DATA\_AVAILABLE\_EVENT,

EVENT\_DOWNLOAD\_CHANNEL\_STATUS\_EVENT,

EVENT\_UNRECOGNIZED\_ENVELOPE,

EVENT\_TERMINAL\_RESPONSE,

EVENT\_APPLICATION\_INSTALL  
} CatFrameworkEventType;

### 5.11.4 CatEnvelopeTagType

typedef enum {  
 SMS\_PP\_DOWNLOAD\_TAG = 0xD1,  
 CELL\_BROADCAST\_TAG = 0xD2,  
 MENU\_SELECTION\_TAG = 0xD3,  
 CALL\_CONTROL\_TAG = 0xD4,  
 MO\_SHORT\_MESSAGE\_CONTROL\_TAG = 0xD5,  
 EVENT\_DOWNLOAD\_TAG = 0xD6,  
 TIMER\_EXPIRATION = 0xD7  
} CatEnvelopeTagType;

### 5.11.5 CatEventType

typedef enum {  
 MT\_CALL\_EVENT = 0x00,  
 CALL\_CONNECTED\_EVENT = 0x01,  
 CALL\_DISCONNECTED\_EVENT = 0x02,  
 LOCATION\_STATUS\_EVENT = 0x03,  
 USER\_ACTIVITY\_EVENT = 0x04,  
 IDLE\_SCREEN\_AVAILABLE = 0x05,  
 CARD\_READER\_STATUS = 0x06,  
 LANGUAGE\_SELECTION = 0x07,  
 BROWSER\_TERMINATION = 0x08,  
 DATA\_AVAILABLE = 0x09,  
 CHANNEL\_STATUS = 0x0A  
} CatEventType;

### 5.11.6 CatTextString

typedef struct

{

CatDCSValue DCSValue;  
 BYTE TextStringLength;  
 const void \*TextString;  
} CatTextString;

### 5.11.7 CatAlphaString

typedef struct

{  
 BYTE AlphaStringLength;  
 const void \*AlphaString;  
} CatTextString;

### 5.11.8 CatIconIdentifier

typedef struct

{

BYTE UseIcon;

BYTE IconIdentifier;   
 BYTE IconOptions;

} CatIconIdentifier;

The *CatIconIdentifier* structure is defined as follows:

|  |  |
| --- | --- |
| *UseIcon* | If zero, the icon identifier is not used in the proactive command. If non-zero, the IconIdentifier and IconOption members are used in the proactive command. |
| *IconIdentifier* | Index of the icon to display. |
| *IconOptions* | Options with which to display the icon selected from [CatIconOption](#_GsmIconOptions). This is specified as a BYTE rather than CatIconOptios as, in C, an enumeration uses the same storage as an int which is at least 16 bits, whereas the proactive commands that use these identifiers use 8-bit quantities. |

### 5.11.9 CatIconOption

typedef enum

{

SHOW\_WITHOUT\_TEXT = 0x00,

SHOW\_WITH\_TEXT = 0x01

} CatIconOption;

### 5.11.10 CatDCSValue

typedef enum

{

DCS\_SMS\_PACKED = 0x00,

DCS\_SMS\_UNPACKED = 0x04,

DCS\_SMS\_UNICODE = 0x08

} CatDCSValue;

### 5.11.11 CatDisplayTextOptions

typedef enum

{

NORMAL\_PRIORITY\_AUTO\_CLEAR = 0x00,

NORMAL\_PRIORITY\_USER\_CLEAR = 0x80,

HIGH\_PRIORITY\_AUTO\_CLEAR = 0x01,

HIGH\_PRIORITY\_USER\_CLEAR = 0x81

} CatDisplayTextOptions;

### 5.11.12 CatGetInKeyOptions

typedef enum

{

YES\_NO\_OPTION\_NO\_HELP = 0x04,

YES\_NO\_OPTION\_WITH\_HELP = 0x84,

DIGITS\_ONLY\_NO\_HELP = 0x00,

DIGITS\_ONLY\_WITH\_HELP = 0x80,

SMS\_CHARACTER\_NO\_HELP = 0x01,

SMS\_CHARACTER\_WITH\_HELP = 0x81,

UCS2\_CHARACTER\_NO\_HELP = 0x03,

UCS2\_CHARACTER\_WITH\_HELP = 0x83

} CatGetInKeyOptions;

### 5.11.13 CatGetInputOptions

typedef enum

{

PACKED\_DIGITS\_ONLY\_NO\_HELP = 0x08,

PACKED\_DIGITS\_ONLY\_WITH\_HELP = 0x88,

PACKED\_DIGITS\_ONLY\_NO\_ECHO\_NO\_HELP = 0x0C,

PACKED\_DIGITS\_ONLY\_NO\_ECHO\_WITH\_HELP = 0x8C,

UNPACKED\_DIGITS\_ONLY\_NO\_HELP = 0x00,

UNPACKED\_DIGITS\_ONLY\_WITH\_HELP = 0x80,

UNPACKED\_DIGITS\_ONLY\_NO\_ECHO\_NO\_HELP = 0x04,

UNPACKED\_DIGITS\_ONLY\_NO\_ECHO\_WITH\_HELP = 0x84,

PACKED\_SMS\_ALPHABET\_NO\_HELP = 0x09,

PACKED\_SMS\_ALPHABET\_WITH\_HELP = 0x89,

PACKED\_SMS\_ALPHABET\_NO\_ECHO\_NO\_HELP = 0x0D,

PACKED\_SMS\_ALPHABET\_NO\_ECHO\_HELP = 0x8D,

UNPACKED\_SMS\_ALPHABET\_NO\_HELP = 0x01,

UNPACKED\_SMS\_ALPHABET\_WITH\_HELP = 0x81,

UNPACKED\_SMS\_ALPHABET\_NO\_ECHO\_NO\_HELP = 0x05,

UNPACKED\_SMS\_ALPHABET\_NO\_ECHO\_WITH\_HELP = 0x85,

UCS2\_ALPHABET\_NO\_HELP = 0x03,

UCS2\_ALPHABET\_WITH\_HELP = 0x83,

UCS2\_ALPHABET\_NO\_ECHO\_NO\_HELP = 0x07,

UCS2\_ALPHABET\_NO\_ECHO\_WITH\_HELP = 0x87

} CatGetInputOptions;

### 5.11.14 CatSelectItemOptions

typedef enum

{

PRESENT\_AS\_DATA\_VALUES\_NO\_HELP = 0x01,

PRESENT\_AS\_DATA\_VALUES\_WITH\_HELP = 0x81,

PRESENT\_AS\_NAVIGATION\_OPTIONS\_NO\_HELP = 0x03,

PRESENT\_AS\_NAVIGATION\_OPTIONS\_WITH\_HELP = 0x83,

DEFAULT\_STYLE\_NO\_HELP = 0x00,

DEFAULT\_STYLE\_WITH\_HELP = 0x80

} CatSelectItemOptions;

### 5.11.15 CatTimeUnit

typedef enum

{

GSM\_MINUTES = 0x00,

GSM\_SECONDS = 0x01,

GSM\_TENTHS\_OF\_SECONDS = 0x02

} CatTimeUnit;

### 5.11.16 CatTone

typedef enum

{

DIAL\_TONE = 0x01,

CALLER\_BUSY = 0x02,

CONGESTION = 0x03,

RADIO\_PATH\_ACKNOWLEDGE = 0x04,

CALL\_DROPPED = 0x05,

SPECIAL\_INFORMATION\_OR\_ERROR = 0x06,

CALL\_WAITING\_TONE = 0x07,

RINGING\_TONE = 0x08,

GENERAL\_BEEP = 0x10,

POSITIVE\_ACKNOWLEDGE\_TONE = 0x11,

NEGATIVE\_ACKNOWLEDGE\_TONE = 0x12

} CatTone;

### 5.11.17 CatRefreshOptions

typedef enum

{

REFRESH\_SIM\_INIT\_AND\_FULL\_FILE\_CHANGE\_NOTIFICATION = 0x00,  
 REFRESH\_FILE\_CHANGE\_NOTIFICATION = 0x01,

REFRESH\_SIM\_INIT\_AND\_FILE\_CHANGE\_NOTIFICATION = 0x02,

REFRESH\_SIM\_INIT = 0x03,  
 REFRESH\_SIM\_RESET = 0x04

} CatRefreshOptions;

### 5.11.18 CatGetReaderStatusOptions

typedef enum

{

CARD\_READER\_STATUS = 0x00,   
 CARD\_READER\_IDENTIFIER = 0x01  
} CatGetReaderStatusOptions;

### 5.11.19 CatDevice

typedef enum

{

DEVICE\_KEPYAD = 0x01,

DEVICE\_DISPLAY = 0x02,

DEVICE\_EARPIECE = 0x03,

DEVICE\_CARD\_READER\_0 = 0x10,

DEVICE\_CARD\_READER\_1 = 0x11,

DEVICE\_CARD\_READER\_2 = 0x12,

DEVICE\_CARD\_READER\_3 = 0x13,

DEVICE\_CARD\_READER\_4 = 0x14,

DEVICE\_CARD\_READER\_5 = 0x15,

DEVICE\_CARD\_READER\_6 = 0x16,

DEVICE\_CARD\_READER\_7 = 0x17,

DEVICE\_CHANNEL\_1 = 0x21,

DEVICE\_CHANNEL\_2 = 0x22,

DEVICE\_CHANNEL\_3 = 0x23,

DEVICE\_CHANNEL\_4 = 0x24,

DEVICE\_CHANNEL\_5 = 0x25,

DEVICE\_CHANNEL\_6 = 0x26,

DEVICE\_CHANNEL\_7 = 0x27,

DEVICE\_SIM = 0x81,

DEVICE\_ME = 0x82,

DEVICE\_NETWORK = 0x83

} CatDevice;

### 5.11.20 CatGeneralResult

typedef enum

{

CAT\_COMMAND\_SUCCESSFUL = 0x00,

CAT\_COMMAND\_SUCCESSFUL\_WITH\_PARTIAL\_COMPREHENSION = 0x01,

CAT\_COMMAND\_SUCCESSFUL\_WITH\_MISSING\_INFORMATION = 0x02,

CAT\_REFRESH\_SUCCESSFUL\_WITH\_ADDITIONAL\_EFS\_READ = 0x03,

CAT\_COMMAND\_SUCCESSFUL\_BUT\_ICON\_NOT\_FOUND = 0x04,

CAT\_COMMAND\_SUCCESSFUL\_BUT\_MODIFIED\_BY\_CALL\_CONTROL = 0x05,

CAT\_COMMAND\_SUCCESSFUL\_BUT\_LIMITED\_SERVICE = 0x06,

CAT\_COMMAND\_SUCCESSFUL\_WITH\_MODIFICATION = 0x07,

CAT\_ABORTED\_BY\_USER = 0x10,

CAT\_BACKWARD = 0x11,

CAT\_NO\_RESPONSE = 0x12,

CAT\_HELP\_REQUIRED = 0x13,

CAT\_USSD\_ABORTED\_BY\_USER = 0x14,

CAT\_ME\_UNABLE\_TO\_PROCESS\_COMMAND = 0x20,

CAT\_NETWORK\_UNABLE\_TO\_PROCESS\_COMMAND = 0x21,

CAT\_USER\_REJECTED\_SETUP\_CALL = 0x22,

CAT\_USER\_CLEARED\_BEFORE\_RELEASE = 0x23,

CAT\_ACTION\_CONTRADICT\_TIMER\_STATE = 0x24,

CAT\_TEMP\_PROBLEM\_IN\_CALL\_CONTROL = 0x25,

CAT\_LAUNCH\_BROWSER\_ERROR = 0x26,

CAT\_COMMAND\_BEYOND\_ME\_CAPABILITIES = 0x30,

CAT\_COMMAND\_TYPE\_NOT\_UNDERSTOOD = 0x31,

CAT\_COMMAND\_DATA\_NOT\_UNDERSTOOD = 0x32,

CAT\_COMMAND\_NUMBER\_NOT\_KNOWN = 0x33,

CAT\_SS\_RETURN\_ERROR = 0x34,

CAT\_SMS\_RP\_ERROR = 0x35,

CAT\_REQUIRED\_VALUES\_MISSING = 0x36,

CAT\_USSD\_RETURN\_ERROR = 0x37,

CAT\_MULTIPLE\_CARD\_COMMAND\_ERROR = 0x38,

CAT\_PERMANENT\_PROBLEM\_IN\_SMS\_OR\_CALL\_CONTROL = 0x39,

CAT\_BEARER\_INDEPENDENT\_PROTOCOL\_ERROR = 0x3A

} CatGeneralResult;

### 5.11.21 CatTimerValue

typedef struct

{

BYTE hour;

BYTE minute;

BYTE second;

} CatTimerValue;

The *CatTimerValue* data type has three one-byte values:

|  |  |
| --- | --- |
| *hour* | Hours part of timer. |
| *Minute* | Minutes part of timer. |
| *Second* | Seconds part of timer. |

### 5.11.22 CatTimeInterval

typedef struct

{

BYTE timeUnit;

BYTE timeInterval;

} CatTimeInterval;

The *CatTimInterval* data type has two one-byte values:

|  |  |
| --- | --- |
| *timeUnit* | One of the [CatTimeUnit](#_GsmTimeUnit) enumeration values. This is specified as a BYTE rather than CatTimeUnit as, in C, an enumeration uses the same storage as an int which is at least 16 bits, whereas the proactive commands that use these identifiers use 8-bit quantities. |
| *TimeInterval* | The number of timeUnits. |

### 5.11.23 CatFileStatus

typedef struct

{

;

WORD recordLength;

WORD numberOfRecords;

BYTE lengthOfTrailer;  
 BYTE trailer[];

} CatEFStatus;

typedef struct

{

BYTE numberOfDFs;  
 BYTE numberofEFs;

BYTE CHV1Status;

BYTE unblockCHV1Status;

BYTE CHV2Status;

BYTE unblockCHV2Status;

BYTE lengthOfTrailer;

BYTE trailer[];

} CatDFStatus;

typedef struct

{

DWORD totalFileSize;

UINT16 fileID;

BYTE fileDescriptorByte;

BYTE fileType; // 00=RFU, 01=MF, 02=DF, 04=EF

BYTE fileLifeCycleStatus;

union  
 {

CatEFStatus ef;  
 CatDFStatus df;  
 } u;

} CatFileStatus;

### 5.11.24 CatLanguageNotificationOptions

typedef enum

{

LANGUAGE\_NON\_SPECIFIC\_NOTIFICATION = 0x00,  
 LANGUAGE\_SPECIFIC\_NOTIFICATION = 0x01  
} CatLanguageNotificationOptions;

### 5.11.25 CatLocationInformation

typedef struct

{

BYTE mobileCountryNetworkCodes[3];  
 BYTE LAC[2];  
 BYTE cellID[2];  
} CatLocationInformation;

### 5.11.26 CatTimingAdvance

typedef struct

{

BYTE MEStatus;  
 BYTE timingAdvance;  
} CatTimingAdvance;

### 5.11.27 CatLaunchBrowserOptions

typedef enum

{

LAUNCH\_BROWSER\_IF\_NOT\_ALREADY\_LAUNCHED = 0x00,

USE\_EXISTING\_BROWSER = 0x02,

CLOSE\_EXISTING\_BROWSER\_AND\_LAUNCH\_NEW\_BROWSER = 0x03

} CatLaunchBrowserOptions;

### 5.11.28 CatSetupCallOptions

typedef enum

{

CALL\_ONLY\_IF\_NOT\_BUSY = 0x00,

CALL\_ONLY\_IF\_NOT\_BUSY\_WITH\_REDIAL = 0x01,

CALL\_AND\_PUT\_ALL\_OTHER\_CALLS\_ON\_HOLD = 0x02,

CALL\_AND\_PUT\_ALL\_OTHER\_CALLS\_ON\_HOLD\_WITH\_REDIAL = 0x03,

CALL\_AND\_DISCONNECT\_ALL\_OTHER\_CALLS = 0x04,

CALL\_AND\_DISCONNECT\_ALL\_OTHER\_CALLS\_WITH\_REDIAL = 0x05

} CatSetupCallOptions;

### 5.11.29 CatTypeOfNumberAndNumberingPlanIdentifier

typedef enum

{

TON\_UNKNOWN\_AND\_NPI\_UNKNOWN = 0x80,

TON\_INTERNATIONAL\_AND\_NPI\_UNKNOWN = 0x90,

TON\_NATIONAL\_AND\_NPI\_UNKNOWN = 0xA0,

TON\_NETWORK\_AND\_NPI\_UNKNOWN = 0xB0,

TON\_SUBSCRIBER\_AND\_NPI\_UNKNOWN = 0xC0,

TON\_UNKNOWN\_AND\_NPI\_TELEPHONE = 0x81,

TON\_INTERNATIONAL\_AND\_NPI\_TELEPHONE = 0x91,

TON\_NATIONAL\_AND\_NPI\_TELEPHONE = 0xA1,

TON\_NETWORK\_AND\_NPI\_TELEPHONE = 0xB1,

TON\_SUBSCRIBER\_AND\_NPI\_TELEPHONE = 0xC1,

TON\_UNKNOWN\_AND\_NPI\_DATA = 0x83,

TON\_INTERNATIONAL\_AND\_NPI\_DATA = 0x93,

TON\_NATIONAL\_AND\_NPI\_DATA = 0xA3,

TON\_NETWORK\_AND\_NPI\_DATA = 0xB3,

TON\_SUBSCRIBER\_AND\_NPI\_DATA = 0xC3,

TON\_UNKNOWN\_AND\_NPI\_TELEX = 0x84,

TON\_INTERNATIONAL\_AND\_NPI\_TELEX = 0x94,

TON\_NATIONAL\_AND\_NPI\_TELEX = 0xA4,

TON\_NETWORK\_AND\_NPI\_TELEX = 0xB4,

TON\_SUBSCRIBER\_AND\_NPI\_TELEX = 0xC4,

TON\_UNKNOWN\_AND\_NPI\_NATIONAL = 0x88,

TON\_INTERNATIONAL\_AND\_NPI\_NATIONAL = 0x98,

TON\_NATIONAL\_AND\_NPI\_NATIONAL = 0xA8,

TON\_NETWORK\_AND\_NPI\_NATIONAL = 0xB8,

TON\_SUBSCRIBER\_AND\_NPI\_NATIONAL = 0xC8,

TON\_UNKNOWN\_AND\_NPI\_PRIVATE = 0x89,

TON\_INTERNATIONAL\_AND\_NPI\_PRIVATE = 0x99,

TON\_NATIONAL\_AND\_NPI\_PRIVATE = 0xA9,

TON\_NETWORK\_AND\_NPI\_PRIVATE = 0xB9,

TON\_SUBSCRIBER\_AND\_NPI\_PRIVATE = 0xC9,

TON\_UNKNOWN\_AND\_NPI\_ERMES = 0x8A,

TON\_INTERNATIONAL\_AND\_NPI\_ERMES = 0x9A,

TON\_NATIONAL\_AND\_NPI\_ERMES = 0xAA,

TON\_NETWORK\_AND\_NPI\_ERMES = 0xBA,

TON\_SUBSCRIBER\_AND\_NPI\_ERMES = 0xCA

} CatTypeOfNumberAndNumberingPlanIdentifier;

### 5.11.30 CatSendShortMessageOptions

typedef enum

{

PACKING\_NOT\_REQUIRED = 0x00,

PACKING\_BY\_THE\_ME\_REQUIRED = 0x01

} CatSendShortMessageOptions;

### 5.11.31 CatSendDataOptions

typedef enum

{

STORE\_DATA\_IN\_TX\_BUFFER = 0x00,  
 SEND\_DATA\_IMMEDIATELY = 0x01

} CatSendDataOptions;

### 5.11.32 CatMEInterfaceTransportLevelType

typedef struct  
{  
 enum   
 {  
 UDP = 0x01,  
 TCP = 0x02  
 } TransportProtocolType;  
 UINT16 CAT\_ME\_PortNumber;  
} CAT\_MEInterfaceTransportLevelType;

### 5.11.33 CatBearer

typedef enum

{

BEARER\_SMS = 0x00,  
 BEARER\_CSD = 0x01,  
 BEARER\_USSD = 0x02,  
 BEARER\_GPRS = 0x03

} CatBearer;

### 5.11.34 CatOpenChannelOptions

typedef enum

{

ON\_DEMAND\_LINK\_ESTABLISHMENT = 0x00,  
 IMMEDIATE\_LINK\_ESTABLISHMENT = 0x01  
} CatOpenChannelOptions;

### 5.11.35 CatAddressType

typedef struct  
{

enum

{

IPV4 = 0x21,

IPV6 = 0x97

} AddressType;  
 BYTE AddressLength;  
 const void \*Address;  
} CatAddressType;

### 5.11.36 CatFID

#define FID\_DF\_GRAPHICS 0x5F50

#define FID\_DF\_TELECOM 0x7F10

#define FID\_EF\_ADN 0x6F3A

#define FID\_EF\_ARR 0x2F06

#define FID\_EF\_BDN 0x6F4D

#define FID\_EF\_CCP 0x6F3D

#define FID\_EF\_DIR 0x2F00

#define FID\_EF\_EXT1 0x6F4A

#define FID\_EF\_EXT2 0x6F4B

#define FID\_EF\_EXT3 0x6F4C

#define FID\_EF\_EXT4 0x6F4E

#define FID\_EF\_FDN 0x6F3B

#define FID\_EF\_ICCID 0x2FE2

#define FID\_EF\_IMG 0x4F20

#define FID\_EF\_LND 0x6F44

#define FID\_EF\_MSISDN 0x6F40

#define FID\_EF\_PL 0x2F05

#define FID\_EF\_SDN 0x6F49

#define FID\_EF\_SMS 0x6F3C

#define FID\_EF\_SMSP 0x6F42

#define FID\_EF\_SMSR 0x6F47

#define FID\_EF\_SMSS 0x6F43

### 5.11.37 CatTextFormat

#define TEXT\_FORMAT\_LEFT 0x00

#define TEXT\_FORMAT\_CENTER 0x01

#define TEXT\_FORMAT\_RIGHT 0x02

#define TEXT\_FORMAT\_LANGUAGE\_DEPENDENT 0x03

#define TEXT\_FORMAT\_NORMAL\_SIZE 0x00

#define TEXT\_FORMAT\_LARGE\_SIZE 0x04

#define TEXT\_FORMAT\_SMALL\_SIZE 0x08

#define TEXT\_FORMAT\_BOLD 0x10

#define TEXT\_FORMAT\_ITALIC 0x20

#define TEXT\_FORMAT\_UNDERLINED 0x40

#define TEXT\_FORMAT\_STRIKETHROUGH 0x80

### 5.11.38 CatTextForegroundColour

typedef enum {

BLACK = 0x00,

DARK\_GREY = 0x01,

DARK\_RED = 0x02,

DARK\_YELLOW = 0x03,

DARK\_GREEN = 0x04,

DARK\_CYAN = 0x05,

DARK\_BLUE = 0x06,

DARK\_MAGENTA = 0x07,

GREY = 0x08,

WHITE = 0x09,

BRIGHT\_RED = 0x0A,

BRIGHT\_YELLOW = 0x0B,

BRIGHT\_GREEN = 0x0C,

BRIGHT\_CYAN = 0x0D,

BRIGHT\_BLUE = 0x0E,

BRIGHT\_MAGENTA = 0x0F

} CatTextForegroundColour;

### 5.11.39 CatTextBackgroundColour

typedef enum {

BLACK = 0x00,

DARK\_GREY = 0x10,

DARK\_RED = 0x20,

DARK\_YELLOW = 0x30,

DARK\_GREEN = 0x40,

DARK\_CYAN = 0x50,

DARK\_BLUE = 0x60,

DARK\_MAGENTA = 0x70,

GREY = 0x80,

WHITE = 0x90,

BRIGHT\_RED = 0xA0,

BRIGHT\_YELLOW = 0xB0,

BRIGHT\_GREEN = 0xC0,

BRIGHT\_CYAN = 0xD0,

BRIGHT\_BLUE = 0xE0,

BRIGHT\_MAGENTA = 0xF0

} CatTextBackgroundColour;

Annex A (normative):  
Application executable architecture

| Name | Value | Meaning |
| --- | --- | --- |
| EM\_NONE | 0 | No machine |
| EM\_M32 | 1 | AT&T WE 32100 |
| EM\_SPARC | 2 | SPARC |
| EM\_386 | 3 | Intel 80386 |
| EM\_68K | 4 | Motorola 68000 |
| EM\_88K | 5 | Motorola 88000 |
| RESERVED | 6 | Reserved for future use |
| EM\_860 | 7 | Intel 80860 |
| EM\_MIPS | 8 | MIPS I Architecture |
| EM\_S370 | 9 | IBM System/370 Processor |
| EM\_MIPS\_RS3\_LE | 10 | MIPS RS3000 Little-endian |
| RESERVED | 11-14 | Reserved for future use |
| EM\_PARISC | 15 | Hewlett-Packard PA-RISC |
| RESERVED | 16 | Reserved for future use |
| EM\_VPP500 | 17 | Fujitsu VPP500 |
| EM\_SPARC32PLUS | 18 | Enhanced instruction set SPARC |
| EM\_960 | 19 | Intel 80960 |
| EM\_PPC | 20 | PowerPC |
| EM\_PPC64 | 21 | 64-bit PowerPC |
| RESERVED | 22-35 | Reserved for future use |
| EM\_V800 | 36 | NEC V800 |
| EM\_FR20 | 37 | Fujitsu FR20 |
| EM\_RH32 | 38 | TRW RH-32 |
| EM\_RCE | 39 | Motorola RCE |
| EM\_ARM | 40 | Advanced RISC Machines ARM |
| EM\_ALPHA | 41 | Digital Alpha |
| EM\_SH | 42 | Hitachi SH |
| EM\_SPARCV9 | 43 | SPARC Version 9 |
| EM\_TRICORE | 44 | Infineon Tricore embedded processor |
| EM\_ARC | 45 | Argonaut RISC Core |
| EM\_H8\_300 | 46 | Hitachi H8/300 |
| EM\_H8\_300H | 47 | Hitachi H8/300H |
| EM\_H8S | 48 | Hitachi H8S |
| EM\_H8\_500 | 49 | Hitachi H8/500 |
| EM\_IA\_64 | 50 | Intel IA-64 processor architecture |
| EM\_MIPS\_X | 51 | Stanford MIPS-X |
| EM\_COLDFIRE | 52 | Motorola ColdFire |
| EM\_68HC12 | 53 | Motorola M68HC12 |
| EM\_MMA | 54 | Fujitsu MMA Multimedia Accelerator |
| EM\_PCP | 55 | Siemens PCP |
| EM\_NCPU | 56 | Sony nCPU embedded RISC processor |
| EM\_NDR1 | 57 | Denso NDR1 microprocessor |
| EM\_STARCORE | 58 | Motorola Star\*Core processor |
| EM\_ME16 | 59 | Toyota ME16 processor |
| EM\_ST100 | 60 | STMicroelectronics ST100 processor |
| EM\_TINYJ | 61 | Advanced Logic Corp. TinyJ embedded processor family |
| Reserved | 62-65 | Reserved for future use |
| EM\_FX66 | 66 | Infineon FX66 microcontroller |
| EM\_ST9PLUS | 67 | STMicroelectronics ST9+ 8/16 bit microcontroller |
| EM\_ST7 | 68 | STMicroelectronics ST7 8-bit microcontroller |
| EM\_68HC16 | 69 | Motorola MC68HC16 Microcontroller |
| EM\_68HC11 | 70 | Motorola MC68HC11 Microcontroller |
| EM\_68HC08 | 71 | Motorola MC68HC08 Microcontroller |
| EM\_68HC05 | 72 | Motorola MC68HC05 Microcontroller |
| EM\_SVX | 73 | Silicon Graphics SVx |
| EM\_ST19 | 74 | STMicroelectronics ST19 8-bit microcontroller |
| EM\_VAX | 75 | Digital VAX |
| EM\_CRIS | 76 | Axis Communications 32-bit embedded processor |
| EM\_JAVELIN | 77 | Infineon Technologies 32-bit embedded processor |
| EM\_FIREPATH | 78 | Element 14 64-bit DSP Processor |
| EM\_ZSP | 79 | LSI Logic 16-bit DSP Processor |
| EM\_MMIX | 80 | Donald Knuth's educational 64-bit processor |
| EM\_HUANY | 81 | Harvard University machine-independent object files |
| EM\_PRISM | 82 | SiTera Prism |
| EM\_MEL | 83 | Multos Executable Language (MEL) byte codes |
| EM\_RTE | 84 | Microsoft Smart Card for Windows Runtime Environment byte codes |

Annex B (informative):  
Example

/\*\*

\*\* Example of Toolkit Application written for the C SIM API

\*/

#pragma AID A0000000090001

#include <stdlib.h>

#include "application.h"

#include "cat.h"

#include "catlow.h"

#define DF\_GSM 0x7F20

#define EF\_PUCT 0x6F41

const BYTE SERVER\_OPERATION = 0x0F;

const BYTE EXIT\_REQUESTED\_BY\_USER = 0x10;

static const char menuEntry[] = "Service1";

static const char menuTitle[]= "MyMenu";

static char item1[] = "ITEM1";

static char item2[] = "ITEM2";

static char item3[] = "ITEM3";

static char item4[] = "ITEM4";

static char textDText[] = "Hello, world";

static char textGInput[] = "Your name?";

BYTE ItemIdentifier;

static BYTE \* byteptr;

static void \* bufptr;

static BYTE buffer[10];

static BYTE itemId;

static BYTE result;

static BYTE repeat;

void main(void)

{

switch (CatGetFrameworkEvent())

{

case EVENT\_APPLICATION\_LIFECYCLE\_INSTALL:

// Define the application Menu Entry and register to the EVENT\_MENU\_SELECTION

CatSetMenuString (1,sizeof(menuEntry),(const void \*)MenuEntry,NULL,0,0);

// register to the EVENT\_UNFORMATTED\_SMS\_PP\_ENV

CatNotifyOnEnvelope(SMS\_PP\_DOWNLOAD\_TAG,1);

break;

case EVENT\_ENVELOPE\_COMMAND:

{

BYTE length;

switch (CatOpenEnvelope(&length))

{

case MENU\_SELECTION\_TAG:

// Prepare the Select Item proactive command

// Append the Menu Title

CatSelectItem (sizeof(MenuTitle),

(const void \*)MenuTitle,

DEFAULT\_STYLE\_NO\_HELP);

// add all the Item

CatSelectAddItem(sizeof(item1),(const void \*)item1,1);

CatSelectAddItem(sizeof(item2),(const void \*)item2,2);

CatSelectAddItem(sizeof(item3),(const void \*)item3,3);

CatSelectAddItem(sizeof(item4),(const void \*)item4,4);

// ask the CAT Toolkit Framework to send

//the proactive command and check the result

if (!CatEndSelectItem(&ItemId,NULL))

{

switch(ItemId)

{

case 1:

case 2:

case 3: // DisplayText

CatDisplayText (DCS\_SMS\_UNPACKED,

sizeof(textDText),

(const void \*) textDText,

NORMAL\_PRIORITY\_USER\_CLEAR, NULL, 0);

break;

case 4: // Ask the user to enter data and display it

repeat=0;

do

{

if (CatGetInput(DCS\_SMS\_UNPACKED,

sizeof(textGInput),

(const void \*) textGInput,

UNPACKED\_SMS\_ALPHABET\_NO\_HELP,

DCS\_SMS\_UNPACKED, 0, NULL,

0, sizeof(buffer), NULL,

(CatDCSValue \*)&result,

&repeat,

(void \*)buffer)==EXIT\_REQUESTED\_BY\_USER)

break;

// display the entered text

CatDisplayText ((CatDCSValue )result,

repeat, (const void \*) buffer,

NORMAL\_PRIORITY\_USER\_CLEAR, NULL, 0);

} while (repeat);

}

}

break;

case EVENT\_UNFORMATTED\_SMS\_PP\_ENV:

CatOpenEnvelope(&result);

byteptr=(BYTE \*)catGetData(1); /\* go to numberlength \*/

result=(\*byteptr)>>1;

/\* calculate numberlength, rounded up \*/

if ((\*byteptr)&1)result++;

catGetData(result+12); /\* move to the beginning of the data \*/

// get the offset of the instruction in the TP-UD field

CatGetData(SERVER\_OPERATION);

result=CatGetBYTE();

switch(result)

{

case 0x41 : // Update of a gsm file

bufptr=CatGetData(3);

// write these data in the Efpuct

CatSelect(FID\_DF\_GSM);

CatSelect(FID\_EF\_PUCT);

CatUpdateBinary(0,3,bufptr);

break;

case 0x36 : // change the MenuTitle for the SelectItem

bufptr=CatGetData(sizeof(menuTitle));

memcpy(bufptr,memuTitle,sizeof(menuTitle));

}

}

}

break;

default:

CatExit();

break;

}

CatExit();

}

Annex C (informative):  
Change history

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Date | TSG # | TSG Doc | CR | Rev | Cat | Subject/Comment | New |
| 2003-03 | TP-19 | TP-030023 | 001 |  | D | Editorial Corrrections | 6.1.0 |
| 2007-06 | - | - | - | - | - | Update to Rel-7 version (MCC) | 7.0.0 |
| 2009-03 | - | - | - | - | - | Update to Rel-8 version (MCC) | 8.0.0 |
| 2009-12 | CT-46 | CP-091011 | 002 | 1 | F | References update | 8.1.0 |
| 2009-12 | CT-46 | - | - | - | - | Upgrade of the specification to Rel-9 | 9.0.0 |
| 2011-03 | SP-51 | - | - | - | - | Upgrade of the specification to Rel-10 | 10.0.0 |
| 2012-09 | SP-57 | - | - | - | - | Upgrade of the specification to Rel-11 | 11.0.0 |
| 2014-10 | SP-65 | - | - | - | - | Upgrade of the specification to Rel-12 | 12.0.0 |
| 2015-12 | SP-70 | - | - | - | - | Upgrade of the specification to Rel-13 | 13.0.0 |
| 2017-03 | SA-75 | - | - | - | - | Update to Rel-14 version (MCC) | 14.0.0 |
| 2019-01 | CT-81 |  |  |  |  | Update to Rel-15 version (MCC) | 15.0.0 |
| 2020-07 | CT-88e | - | - | - | - | Update to Rel-16 version (MCC) | 16.0.0 |