ETSI TS 131 131 V14.0.0 (2017-04)

Universal Mobile Telecommunications System (UMTS);
LTE;
C-language binding to (U)SIM API
(3GPP TS 31.131 version 14.0.0 Release 14)

Reference
RTS/TSGC-0631131ve00

Keywords
LTE,UMTS

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

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

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

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

The present document may refer to technical specifications or reports using their 3GPP identities, UMTS identities or GSM identities. These should be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between GSM, UMTS, 3GPP and ETSI identities can be found under http://webapp.etsi.org/key/queryform.asp.

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Contents

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.

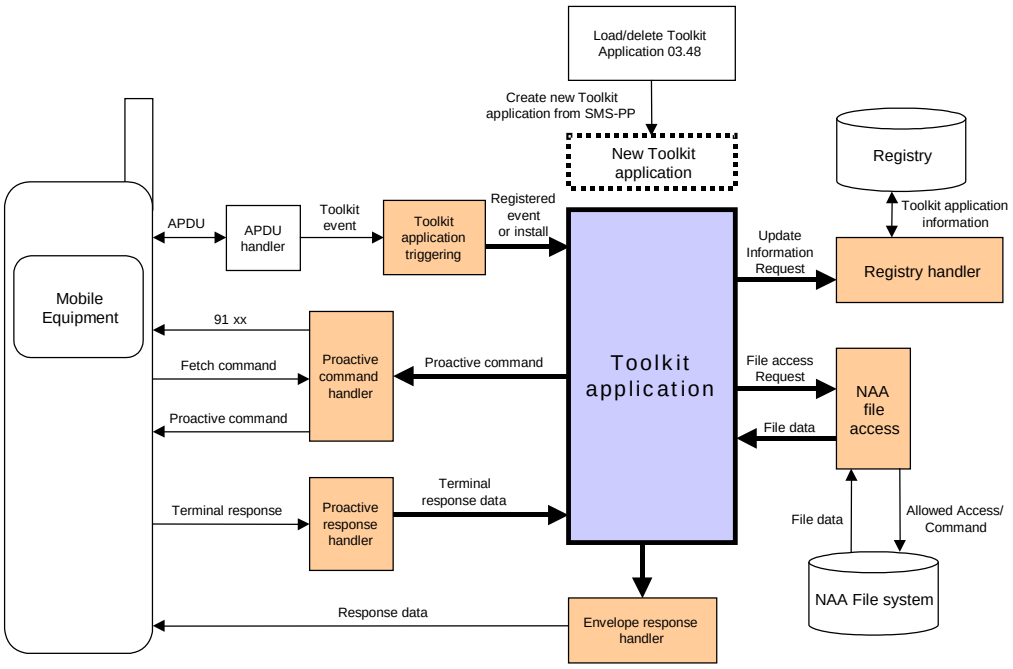

Figure 1

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 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 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 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 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 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 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);
```

TitleLength	[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.

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 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 TextOptions,
                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.
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.
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 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.
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.
IconIdentifier	[in]	Optional icon identifier; see CatIconIdentifier 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.
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.
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.
DefaultReplyDCS	[in]	The data-coding scheme for *DefaultReply*. Acceptable values for this parameter are listed in CatDCSValue.
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 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.
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.
TextLength	[in]	The length in bytes of *Text*.
Text	[in]	String to display while ME is idle.
IconIdentifier	[in]	Optional icon identifier; see CatIconIdentifier 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.
Units	[in]	Unit of time specified for *duration* parameter. Acceptable values for this parameter are listed in CatTimeUnit.
Duration	[in]	Amount of time to play the tone, in units specified in the *Units* parameter
IconIdentifier	[in]	Optional icon identifier; see CatIconIdentifier 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.

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.
TimerValue	[in]	Initial value of the timer. The value is specified in a structure of type CatTimerValue.
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
TimerValue	[out]	The time remaining to run of timer *TimerID*. The value is returned in a structure of type CatTimerValue.
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.
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.
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.
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.
Options	[in]	Selects what type of status information to return. An acceptable value for this parameter is selected from CatGetReaderStatusOptions.
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.
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.
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.
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.
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.
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 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 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.
IconIdentifier	[in]	Optional icon identifier; see CatIconIdentifier 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 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 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.
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.
UserConfirmationIconIdentifier	[in]	Optional icon identifier to use during the user confirmation phase; see CatIconIdentifier 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 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.
TONandNPI	Acceptable values for this parameter are listed in CatTypeOfNumberAndNumberingPlanIdentifier.
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. 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 EF_{CCP}.
CalledPartySubaddressLength	Length in bytes of CalledPartySub*address*.
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.
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.
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.
IconIdentifier	[in]	Optional icon identifier; see CatIconIdentifier 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.
SSStringLength	[in]	Length in bytes of *SSString*.
SSString	[in]	SS string to ME.
IconIdentifier	[in]	Optional icon identifier; see CatIconIdentifier 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.
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 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.
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 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.
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. 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. 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.***
Duration2Defined	***Set to nonzero if Duration2 is defined.***
Duration2	***Duration of timeout; see CatTimeInterval.***
LocalAddress	***The LocalAddress; see CatAddressType.***
UserLogin	***The user login string.***
UserPassword	The user password string.
CAT_MEInterfaceTransportLevel	See CAT_MEInterfaceTransportLevelType.
DataDestinationAddress	The DataDestinationAddress; see CatAddressType.

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.
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 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. 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. 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.**
CAT_ME_InterfaceTransportLevel	See CAT_MEInterfaceTransportLevelType.
DataDestinationAddress	See CatAddressType.

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 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 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.
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 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 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 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 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. 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 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

Name	Value	Meaning
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

Change history								
Date	TSG #	TSG Doc	CR	Rev	Cat	Subject/Comment	Old	New
2003-03	TP-19	TP-030023	001		D	Editorial Corrrections	6.0.0	6.1.0
2007-06	-	-	-	-	-	Update to Rel-7 version (MCC)	6.1.0	7.0.0
2009-03	-	-	-	-	-	Update to Rel-8 version (MCC)	7.1.0	8.0.0
2009-12	CT-46	CP-091011	002	1	F	References update	8.0.0	8.1.0
2009-12	CT-46	-	-	-	-	Upgrade of the specification to Rel-9	8.1.0	9.0.0
2011-03	SP-51	-	-	-	-	Upgrade of the specification to Rel-10	9.0.0	10.0.0
2012-09	SP-57	-	-	-	-	Upgrade of the specification to Rel-11	10.0.0	11.0.0
2014-10	SP-65	-	-	-	-	Upgrade of the specification to Rel-12	11.0.0	12.0.0
2015-12	SP-70	-	-	-	-	Upgrade of the specification to Rel-13	12.0.0	13.0.0
2017-03	SA-75	-	-	-	-	Update to Rel-14 version (MCC)	13.0.0	14.0.0

History

Document history		
V14.0.0	April 2017	Publication