3GPP TSG-T (Terminals) Meeting #14
Kyoto, Japan, 12 - 14 December 2001

Tdoc TP-010249

Presentation of Specification

Presentation to:	TSG-T Meeting #14
Document for presentation:	TS 31.131, v1.0.0 " 'C' Language Binding to USIM API ", Rel-5
Presented for:	Information

Abstract of document:

This specification aims at source-level compatibility of Toolkit Applications. It also aims at hiding the details of assembling proactive commands and disassembling the responses for the application programmer. It does not cover differences in the names of include-files and different names of some of the functions in the C-runtime library. Ways to solve this are well know to developers who write for different platforms.

The main body of the document specifies 'C' language binding for the (U)SIM API but remains independent of the underlying platform. Assumptions about the platform are those carried over from 02.19 [5] which are:

· There shall be a virtual machine through which the Toolkit applications execute.

· The platform shall provide triggering and context switching between applications.

The present document includes information applicable to SIM Toolkit application developers programming in 'C' and an annex showing how an example STK application can be written in a platform-independent manner. It specifies a stage two description of the (U)SIM API internal to the (U)SIM.

The API for loading and deleting toolkit application is specified in GSM 03.48 [4] and is not part of the (U)SIM API 02.19 [5]. Therefore, C-bindings for loading, life-cycle management and deleting Toolkit application are not included in this document.

Changes since last presentation to TSG-T Meeting:

First Presentation

Outstanding Issues:

Draft 3GPP TS 31.131 V1.0.0 (2001-12)

Technical Specification

3rd Generation Partnership Project;
Technical Specification Group Terminals;
'C'-language binding to (U)SIM API
(Release 5)

Keywords
SIM, USIM, API

3GPP

Postal address

3GPP support office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet
http://www.3gpp.org

1. Contents

2. Scope

A Subscriber Identity Module Application Programming Interface (SIM API) has been defined elsewhere [5] as a technology-independent API specification of how SIM Toolkit applications and (U)SIMs co-operate. That specification aims to be independent of both the underlying platform and the programming language technologies.

This specification aims at source-level compatibility of Toolkit Applications. It also aims at hiding the details of assembling proactive commands and disassembling the responses for the application programmer.

It does not cover differences in the names of include-files and different names of some of the functions in the C-runtime library. Ways to solve this are well know to developers who write for different platforms.

The main body of the document specifies 'C' language binding for the (U)SIM API but remains independent of the underlying platform. Assumptions about the platform are those carried over from 02.19 [5] which are:

- There shall be a virtual machine through which the Toolkit applications execute.
- The platform shall provide triggering and context switching between applications.

The present document includes information applicable to SIM Toolkit application developers programming in 'C' and an annex showing how an example STK application can be written in a platform-independent manner. It specifies a stage two description of the (U)SIM API internal to the (U)SIM.

The API for loading and deleting toolkit application is specified in GSM 03.48 [4] and is not part of the (U)SIM API 02.19 [5]. Therefore, C-bindings for loading, life-cycle management and deleting Toolkit application are not included in this document.

3. References

References may be made to:

a) specific versions of publications (identified by date of publication, edition number, version number, etc.), in which case, subsequent revisions to the referenced document do not apply; or

b) all versions up to and including the identified version (identified by "up to and including" before the version identity); or

c) all versions subsequent to and including the identified version (identified by "onwards" following the version identity); or

d) publications without mention of a specific version, in which case the latest version applies.

A non-specific reference to an ETS shall also be taken to refer to later versions published as an EN with the same number.

3.1. Normative references

[1] GSM 01.04 “Digital cellular telecommunications system (Phase 2+); Abbreviations and acronyms”.

[2] 3GPP TS 11.11 V8.4.0: “3rd Generation Partnership Project; Technical Specification Group Terminals Specification of the Subscriber Identity Module – Mobile Equipment (SIM-ME) interface (Release 1999)”.

[3] ETSI TS 101 223 Card Application Toolkit release 4

[4] 3GPP TS 03.48 V8.4.0: “3rd Generation Partnership Project; Technical Specification Group Terminals; Security Mechanisms for the SIM application toolkit; Release 5”.

[5] ETSI TS 02.19 V7.1.0: “Digitial cellular telecommunications system (Phase 2+); Subscriber Identity Module Application Programming Interface (SIM API); Service description;

[6] ISO 639 (1988): “Code for the representation of names of languages”.

[7] GSM 03.38: “Digital cellular telecommunications system (Phase 2+); Alphabets and language-specific information”.

4. Definitions and abbreviations

4.1. Definitions

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

Application: A smart card application.

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

GSM application: Functionality conforming to GSM 11.11[2] and GSM 11.14[3]. This may be an application executing through a virtual machine, or it may be implemented in native code if the underlying technology requires.

Toolkit Application: An application which uses the API [5] for which the 'C'-language binding is described within this document and which only runs under the control of the GSM Application.

4.2. Abbreviations

For the purpose of the present document, the following abbreviations apply, in addition to those listed in GSM 01.04[1]:

APDU	Application Protocol Data Unit
API	Application Programming Interface
CAT	Card Application Toolkit
DCS	Digital Cellular System
DF	Dedicated File
DTMF	Dual Tone Multiple Frequency
EF	Elementary File
FFS	For Further Study
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
TBD	To be determined
TLV	Tag, Length, Value
TPDU	Transport Protocol Data Unit
UICC	Universal Integrated Circuit Card
URL	Uniform Resource Locator
USIM	Universal SIM
USSD	Unstructured Supplementary Services Data

5. Description

The ~~GSM~~SIM Application consists of the following:

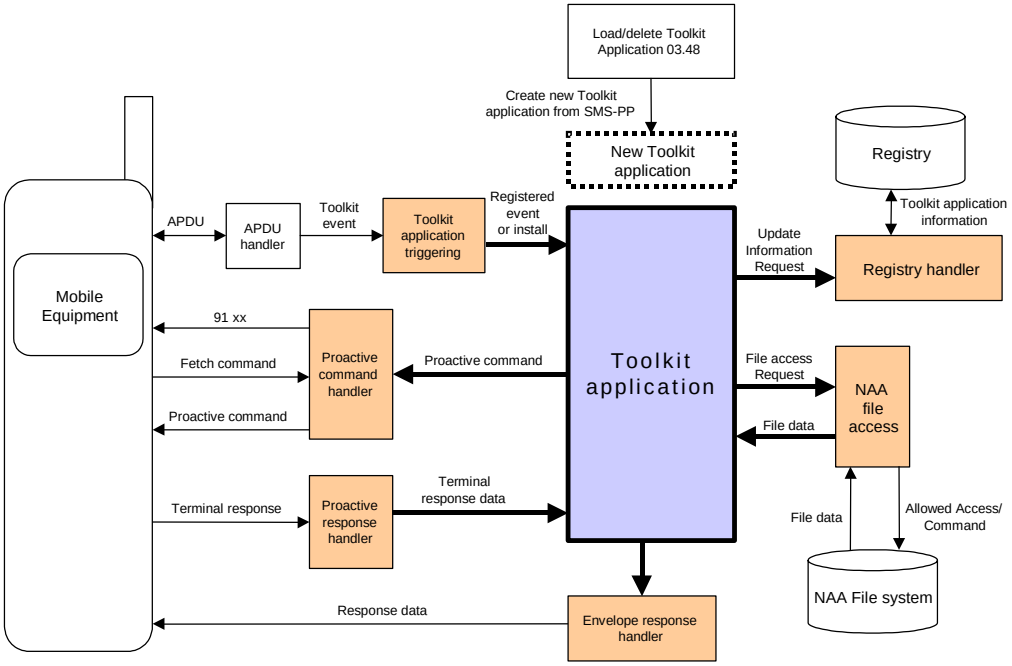
- GSM 11.11[3] APDU handlers for communicating with the mobile equipment,
- GSM 11.11[3] File system and file access control,
- ~~SIM~~ Toolkit Framework which provides services to Toolkit applications.

This document describes the 'C' language bindings for the API [5] between the ~~GSM~~(U)SIM Application and Toolkit Applications. This API allows application 'C' programmers to access functions and data described in GSM 11.11[2] and GSM 11.14[3], such that the ~~GSM~~(U)SIM based services can be developed and loaded onto ~~SIM~~UICCs. If required and supported by the underlying smart card technology, Toolkit Applications can be loaded or deleted remotely, after the card has been issued.

From the STK application programmer's point of view, this API [5] is an extension to any existing platform API available.

5.1. Overview

The *'C'-binding for (U)SIM API* shall provide function calls for GSM 11.14 [3] (pro-active functions) and GSM 11.11 [2] (transport functions). The figure below shows the interactions between a typical Toolkit application (shown in blue) and the various functional blocks (shown in orange) of the SIM [3]. The C-bindings for these APIs are presented in section 5.2.



5.2. Design Rationale and Upward Compatibility

This C SIM API is intended to be general enough for many purposes. Some functions that implement proactive commands take parameters that correspond to optional TLVs in GSM 11.14. If the actual parameter value passed to the function is NULL, the corresponding TLV is not passed to the mobile equipment; 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 GSM 11.14) 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 this standard. 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 are threefold:

- Rather than extend the parameter list of a function to take a large number of optional parameters for each call, it is sometimes preferable to set up the parameters using named structure members before issuing the call to the function.
- If a future version of GSM 11.14 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*. No claim is made as to binary compatibility between implementations or different releases of this standard.

5.3. 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 the latter does not 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 SIM, the Toolkit Framework stores the ME
profile and then triggers the registered toolkit application which 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 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 03.48[4] 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 03.48[4].
The Toolkit Framework shall:
- verify the TS 03.48[4] 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 03.48[4].
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 03.48[4] 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 which 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 succesful CatGetTimer(). The toolkit application can then

manage this 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 11.14 [3] and if no corresponding constant is defined in the list of the ToolkitConstants interface. The Unrecognized Envelope Event will allow a toolkit application to handle the evolution of the TS 11.14 specification.

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 proprietary usage (from –128 to –1). The use of these events will make the toolkit application incompatible.

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.

5.4. Proactive command handling

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

The GSM 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.

5.5. Application Loading

The application loading mechanism, protocol and application life cycle are defined in TS 03.48 [4]

6. 'C'-language binding for (U)SIM API

6.1. Overview

This section 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
- ~~GSM~~ 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.

The function names begin with "Cat" in order to avoid clashing with other function names perhaps being used within STK application.

6.2. Toolkit Version

The version of the API that is implemented by a (U)SIM is defined by the preprocessor symbol "SIM_TOOLKIT_VERSION". This preprocessor symbol can be used by toolkit applications to conditionally compile applications that add or remove functionality depending upon the toolkit version they are compiled for.

6.3. 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 *CatGetFrameworkEvent* 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
- Framework fabricated events

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 precise details of the application lifecycle event are not defined in this document.

Framework fabricated events enable the framework to invoke an application when some state in the SIM has changed. An example of this is to invoke an application when the EFsms file has been updated. The set of framework fabricated events are not defined in this document.

6.3.1. 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(~~SIM_EXIT_SUCCESS~~);
              }
          }
          break;
        default:
          CatExit(~~SIM_EXIT_FAILURE~~);
          break;
      }
    CatExit(~~SIM_EXIT_FAILURE~~);
  }
```

6.3.2. CatGetFrameworkEvent

CatFrameworkEventType
CatGetFrameworkEvent(void);

RETURN	Framework event type that caused the application to run; see CatFrameworkEventType for details.

CatGetFrameworkEvent returns the framework event that caused the application to run.

6.3.3. CatExit

void
CatExit (~~UINT16 code~~ void);

~~*Code*~~	~~[in]~~	~~The implementation defined macros SIM_EXIT_SUCCESS or SIM_EXIT_FAILURE should be used to indicate success or failure.~~

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

6.4. 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 section.

6.4.1. 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 be 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 this document.

6.4.2. 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.

6.4.3. 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.

6.4.4. CatNotifyOnEvent

```
void
CatNotifyOnEnvelopent(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.

6.5. Man-Machine Interface

6.5.1. CatAddItem

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

ItemTextLength	[in]	The length in bytes of *ItemText*.
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. It is not a proactive command.

To display a list of items for the user to choose from, at least three calls that must be issued with no intervening global services for mobile commmunications (GSM) proactive commands in between them. This application programming interface (API) call is the second call. *CatAdditem* must be called after *CatSelectItem* and before *CatEndSelectItem*. *CatAddItem* may be called multiple times consecutively add items to a selection list.

6.5.2. 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 displays a list of items on the mobile equipment for the user to choose from. Even though this function, by name, maps to a GSM proactive command, this API does not itself issue a proactive command. *CatEndSelectItem* must be called for an actual proactive command to be issued.

To display a list of items for the user to choose from, at least three calls must be issued with no intervening GSM proactive commands between them. This API call is the first. The other two APIs required are *CatAddItem* and *CatEndSelectItem*.

6.5.3. 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 mobile equipment a list of items for the user to choose from. The terminal response is parsed and if successful the *SelectedItem* parameter is updated.

To display a list of items for the user to choose from, at least three calls must be issued with no intervening global services for mobile communications (GSM) proactive commands in between them. This function call is the last. The other two APIs required are *CatSelectItem* and *CatAddItem*.

6.5.4. CatDisplayText

```
CatGeneralResult
CatDisplayText (CatDCSValue TextDCS, BYTE TextLength, const void *Text,
                CatDisplayTextOptions Options,
                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.
Options	[in]	Acceptable values for this parameter are listed in CatDisplayTextOptions.
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 mobile equipment before continuing, and the Immediate Response TLV is not passed to the mobile equipment.
RETURN		The GeneralResult code of the DISPLAY TEXT proactive command.

CatDisplayText issues a proactive command that displays text on the display of the mobile equipment.

6.5.5. 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

6.5.6. 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.

6.5.7. 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 mobile equipment 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 mobile equipment's default text string.

6.5.8. 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.

6.6. Timers

6.6.1. CatGetTimer

```
BYTE
CatGetTimer (void);
```

RETURN	The identifier of the timer.

CatGetTimer returns the ID of an available timer. If no timer is available, this function returns zero. Timer identifiers are assigned by the framework.

6.6.2. 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 mobile equipment by this function.

The value returned is zero if the TimerID is valid and is freed, otherwise a non-zero value is received.

6.6.3. 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.

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

6.6.4. 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.

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

6.7. Supplementary Card Reader Management

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

6.7.1. CatPowerOnCard

```
CatGeneralResult
CatPowerOnCard (CatDevice DeviceID, ~~void *ATR,~~ 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.
ATR	[out]	Pointer to where answer to reset (ATR) will be stored.
ATRLength	[out]	Number of bytes returned by the card as the ATR.
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.

6.7.2. 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.

6.7.3. CatPerformCardAPDU

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

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.

6.7.4. 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 mobile equipment. The terminal response is parsed and if successful the *Status* parameter is updated.

6.8. ~~GSM~~UICC File Store Access

These functions are required by 02.19, chapter 4.3.

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 ~~GSM 11.11~~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 Tooolkit 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 Card Application Toolkit Framework checks if this access is allowed by examination of the file control information stored on the card. If access is granted the CAT Framework will process the access request, if access is not granted, an appropriate statusword will be returned .

[Contents and coding of the file(s) containing access control information will be defined in ETSI TS 101.221]

All ~~GSM~~UICC functions return the status bytes according to ETSI TS 101 221 ~~GSM11.11~~., where 90 00 represents "success."

6.8.1. CatSelect

```
UINT16
CatSelect (FID FileIdentifier, CatFileStatus *status);
```

FileIdentifier	[in]	The file to select.
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.

6.8.2. CatStatus

```
UINT16
CatStatus (CatFileStatus *status);
```

NumBytes	[out]	The number of bytes written.
Buffer	[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 ETSI TS 101 221 ~~GSM11.11~~.

6.8.3. CatGetCHVStatus

```
void
CatGetCHVStatus (BYTE CHV[4]);
```

CHVType	[out]	Updates the CHV array with CHV1, CHV2, UNBLOCKCHV1, and UNBLOCKCHV2 with CHV1 at array element zero.

CatGetCHVStatus returns the current CHV values. The format of the returned bytes is specified in ETSI TS 101 221 ~~GSM11.11~~..

6.8.4. CatReadBinary

```
UINT16
CatReadBinary (unsigned short Offset,
               void *NumBytes,
               void *Buffer);
```

Offset	[in]	The offset into the file.
NumBytes	[in]	The number of bytes to read.
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*.

6.8.5. CatUpdateBinary

```
UINT16
CatUpdateBinary (unsigned short Offset,
                 BYTE *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.

6.8.6. CatReadRecord

```
UINT16
CatReadRecord (BYTE RecordNumber,
               CatRecordAccessModes Mode,
               BYTE NumBytes,
               void *Buffer);
```

RecordNumber	[in]	The record number to read from.
Mode	[in]	How to interpret the *RecordNumber*. One of REC_ACC_MODE_NEXT, REC_ACC_MODE_PREVIOUS, REC_ACC_MODE_ABSOLUTE_CURRENT.
NumBytes	[in]	The number of bytes to read from the record.
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.

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

6.8.7. CatUpdateRecord

```
UINT16
CatUpdateRecord (BYTE RecordNumber,
                 CatRecordAccessModes Mode,
                 BYTE NumBytes,
                 const void *Buffer);
```

RecordNumber	[in]	The record number to write into.
Mode	[in]	How to interpret the *RecordNumber*. One of REC_ACC_MODE_NEXT, REC_ACC_MODE_PREVIOUS,

		REC_ACC_MODE_ABSOLUTE_CURRENT.
NumBytes	[in]	The number of bytes to write into the record.
Buffer	[out]	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.

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

6.8.8. CatSeek

```
UINT16
CatSeek (CatSeekModes Mode,
         BYTE 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
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.

CatSeek searches the currently selected file for a pattern of length *patternLength* contained in *Pattern.* If the pattern is found the current record is set appropriately.

6.8.9. CatIncrease

```
UINT16
CatIncrease(unsigned long Increment,
            unsigned long *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.

6.8.10. 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.

6.8.11. 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.

6.9. Miscellaneous

6.9.1. CatGetTerminalProfile

```
void
CatGetTerminalProfile (~~BYTE *Profile,~~
                       BYTE *ProfileOutLength, BYTE *Profile);
```

Profile	[out]	Where the terminal profile is written.
ProfileOutLength	[out]	The number of bytes written to *Profile*.

CatGetTerminalProfile returns the stored terminal profile in *Profile*.

6.9.2. CatMoreTime

```
CatGeneralResult
CatMoreTime (void);
```

RETURN	The GeneralResult code of the MORE TIME proactive command.

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

6.9.3. 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*.

6.9.4. 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 mobile equipment 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 mobile equipment to set a time interval between status application program data units (APDU) that the mobile equipment sends to the ~~subscriber identity module (SIM)~~UICC. The mobile equipment 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*.

6.9.5. 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 SIM REFRESH proactive command.

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

6.9.6. 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 [6], encoded using SMS default 7-bit coded alphabet as defined by GSM 03.38 [7].
RETURN		The GeneralResult code of the LANGUAGE NOTIFICATION proactive command.

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

6.9.7. 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.

6.10. Low-level Interface

This section presents a low-level programming interface which allows you to

- Construct proactive commands and send them to the mobile equipment.
- Access the terminal response from the mobile equipment.
- Search the terminal response and contents of envelopes for specified TLVs.
- Unpack the contents of envelopes from the ME and send responses.

It is required by 02.19 chapter 10.2 .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.

6.10.1. CatResetBuffer

```
void
CatResetBuffer(void);
```

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

6.10.2. 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.

6.10.3. CatSendProactiveCommand

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

Length	[out]	Pointer that is updated with the length of the terminal response
RETURN	[out]	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.

6.10.4. CatOpenEnvelope

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

Length	[out]	Pointer that is updated with the length of the envelope
RETURN	[out]	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.

6.10.5. CatSendEnvelopeResponse

```
void
CatSendEnvelopeResponse (void);
```

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

6.10.6. CatSendEnvelopeErrorResponse

```
void
CatSendEnvelopeErrorResponse (void);
```

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

6.10.7. 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

6.10.8. CatPutByte

```
void
CatPutByte (BYTE Data)
```

Data	[in]	Data byte.

CatPutByte appends the supplied data byte to the data buffer.

6.10.9. 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.

6.10.10. 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.

6.10.11. 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.

6.10.12. 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.

6.10.13. CatGetByte

```
BYTE
CatGetByte (void)
```

RETURN	[out]	Data byte.

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

6.10.14. CatGetData

```
const void *
CatGetData (BYTE Length)
```

Length	[in]	Length of Data
RETURN	[out]	Pointer to Data.

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

6.10.15. 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	[out]	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.

6.10.16. 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	[out]	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.

6.11. Network Services

6.11.1. CatGetLocationInformation

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

LocationInformation	[out]	A pointer to where the location information from the mobile equipment is placed. Refer to the CatLocalInformation section for member details.
RETURN		The GeneralResult code of the PROVIDE LOCAL INFORMATION proactive command.

CatProvideLocationInformation requests the mobile equipment to send location information to the SIM using the PROVIDE LOCAL INFORMATION proactive command.

6.11.2. CatGetTimingAdvance

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

TimingAdvance	[out]	A pointer to where the timing advance information from the mobile equipment is placed. Refer to the CatTimingAdvance section for member details.
RETURN		The GeneralResult code of the PROVIDE LOCAL INFORMATION proactive command.

CatProvideTimingAdvance requests the mobile equipment to send timing advance information to the SIM using the PROVIDE LOCAL INFORMATION proactive command.

6.11.3. CatGetIMEI

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

IMEI	[out]	A pointer to where the IMEI of the mobile equipment is placed.
RETURN		The GeneralResult code of the PROVIDE LOCAL INFORMATION proactive command.

CatGetIMEI requests the mobile equipment to send the IMEI to the SIM using the PROVIDE LOCAL INFORMATION proactive command.

6.11.4. CatGetNetworkMeasurementResults

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

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

CatGetNetworkMeasurementResults requests the mobile equipment to send the network measurement results to the SIM using the PROVIDE LOCAL INFORMATION proactive command.

6.11.5. CatGetDateTimeAndTimeZone

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

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

CatGetDateTimeAndTimeZones requests the mobile equipment to send the date, time, and time zone information to the SIM using the PROVIDE LOCAL INFORMATION proactive command.

6.11.6. CatGetLanguage

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

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

CatGetLanguage requests the mobile equipment to send the language information to the SIM using the PROVIDE

LOCAL INFORMATION proactive command.

6.11.7. 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 *CalledPartySubaddress.*
CalledPartySubaddress	The called party subaddress.
RedialMaximumDuration	An optional maximum duration for the redial mechanism. If the *timeInterval* member of this structure is zero, no duration TLV is sent to the ME.
UserConfirmationIcon	The icon to display during the user confirmation phase. If the *UseIcon* member of this structure is zero, no user confirmation icon TLV is sent to the ME.
CallSetupMessage	String to display during the call set up phase; see CatAlphaString.
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.

6.11.8. 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 mobile equipment is sending a message.
TONandNPI	[in]	Acceptable values for this parameter are listed in CatTypeOfNumberAndNumberingPlanIdentifier.

		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.

6.11.9. 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 mobile equipment 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 mobile equipment.
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 mobile equipment.

6.11.10. 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 mobile equipment 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.

6.11.11. 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 PROVIDE LOCAL INFORMATION 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.

6.11.12. 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* issues 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.

6.11.13. 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 mobile equipment 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.

6.11.14. 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 mobile equipment 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.

6.11.15. 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 mobile equipment 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.

6.11.16. 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.

6.11.17. catServiceSearch

```
catGeneralResult
catReceiveData (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 which 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.

Search for a particular service on a bearer.

6.11.18. catGetServiceInformation

```
catGeneralResult
catReceiveData (BYTE TitleLength, const BYTE *Title,
                const catIconIdentifier *IconIdentifier,
                BYTE 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.

Retrieve information about a particular service on a bearer.

6.11.19. catDeclareService

```
catGeneralResult
```

```
catReceiveData (BYTE 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.
PortNumberh	[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.

Describe a new service.

6.11.20. 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 mobile equipment 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]	Mobile equipment response string.
ResponseLength	[out]	Length in bytes of mobile equipment 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 mobile equipment. The terminal response is parsed and if successful the parameters *Response* and *ResponseLength* are updated.

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

6.12. Supporting Data Types

6.12.1. 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
} GsmFrameworkEventType;
```

6.12.2. 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;
```

6.12.3. 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;
```

6.12.4. CatTextString

```
typedef struct
{
  CatDCSValue DCSValue;
  BYTE TextStringLength;
  const void *TextString;
} CatTextString;
```

6.12.5. CatAlphaString

```
typedef struct
{
  BYTE AlphaStringLength;
  const void *AlphaString;
} CatTextString;
```

6.12.6. 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.

6.12.7. CatIconOption

```
typedef enum
{
  SHOW_WITHOUT_TEXT = 0x00,
  SHOW_WITH_TEXT    = 0x01
} CatIconOption;
```

6.12.8. CatDCSValue

```
typedef enum
{
  DCS_SMS_PACKED      = 0x00,
  DCS_SMS_UNPACKED    = 0x04,
  DCS_SMS_UNICODE     = 0x08
} CatDCSValue;
```

6.12.9. CatDisplayTextOptions

```
typedef enum
{
  NORMAL_PRIORITY_AUTO_CLEAR      = 0x00,
  NORMAL_PRIORITY_USER_CLEAR      = 0x80,
  HIGH_PRIORITY_AUTO_CLEAR        = 0x01,
  HIGH_PRIORITY_USER_CLEAR        = 0x81
} CatDisplayTextOptions;
```

6.12.10. 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;
```

6.12.11. 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;
```

6.12.12. 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;
```

6.12.13. CatTimeUnit

```
typedef enum
{
  GSM_MINUTES           = 0x00,
  GSM_SECONDS           = 0x01,
  GSM_TENTHS_OF_SECONDS = 0x02
} CatTimeUnit;
```

6.12.14. 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;
```

6.12.15. 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;
```

6.12.16. CatGetReaderStatusOptions

```
typedef enum
{
  CARD_READER_STATUS     = 0x00,
  CARD_READER_IDENTIFIER = 0x01
} CatGetReaderStatusOptions;
```

6.12.17. 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;
```

6.12.18. 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;
```

6.12.19. 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.

6.12.20. 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*.

6.12.21. CatFileStatus

```
typedef struct
{
  BYTE increaseAllowed;
  BYTE accessConditions[3];
  BYTE fileStatus;   // 00=transparent, 01=linear, 03=cyclic
  BYTE lengthOfTrailer;
  BYTE structureOfEF;
  BYTE recordLength;
  BYTE trailer[36];  // Not 36, need to figure out how big this actually is
} CatEFStatus;

typedef struct
{
  BYTE rfu1[4];
  BYTE lengthOfTrailer;
  BYTE fileCharacteristics;
  BYTE numberOfDFs;
  BYTE numberofCHVs;
  BYTE rfu2;
  BYTE CHV1Status;
  BYTE unblockCHV1Status;
  BYTE CHV2Status;
  BYTE unblockCHV2Status;
  BYTE rfu3;
  BYTE adminReserved[10];
} CatDFStatus;

typedef struct
{
  BYTE rfu[2];
  UINT16 fileSize;
  UINT16 fileID;
  BYTE fileType;  // 00=RFU, 01=MF, 02=DF, 04=EF
  union
  {
    CatEFStatus ef;
```

```
    CatDFStatus df;
  } u;
} CatFileStatus;
```

6.12.22. CatLanguageNotificationOptions

```
typedef enum
{
  LANGUAGE_NON_SPECIFIC_NOTIFICATION = 0x00,
  LANGUAGE_SPECIFIC_NOTIFICATION     = 0x01
} CatLanguageNotificationOptions;
```

6.12.23. CatLocationInformation

```
typedef struct
{
  BYTE mobileCountryNetworkCodes[3];
  BYTE LAC[2];
  BYTE cellID[2];
} CatLocationInformation;
```

6.12.24. CatTimingAdvance

```
typedef struct
{
  BYTE MEStatus;
  BYTE timingAdvance;
} CatTimingAdvance;
```

6.12.25. CatLaunchBrowserOptions

```
typedef enum
{
  LAUNCH_BROWSER_IF_NOT_ALREADY_LAUNCHED        = 0x00,
  USE_EXISTING_BROWSER                          = 0x02,
  CLOSE_EXISTING_BROWSER_AND_LAUNCH_NEW_BROWSER = 0x03
} CatLaunchBrowserOptions;
```

6.12.26. 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;
```

6.12.27. 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;
```

6.12.28. CatSendShortMessageOptions

```
typedef enum
{
  PACKING_NOT_REQUIRED       = 0x00,
  PACKING_BY_THE_ME_REQUIRED = 0x01
} CatSendShortMessageOptions;
```

6.12.29. CatSendDataOptions

```
typedef enum
{
  STORE_DATA_IN_TX_BUFFER = 0x00,
  SEND_DATA_IMMEDIATELY   = 0x01
} CatSendDataOptions;
```

6.12.30. CAT_MEInterfaceTransportLevelType

```
typedef struct
{
  enum
  {
    UDP = 0x01,
    TCP = 0x02
  } TransportProtocolType;
  UINT16 CAT_ME_PortNumber;
} CAT_MEInterfaceTransportLevelType;
```

6.12.31. CatBearer

```
typedef enum
{
  BEARER_SMS  = 0x00,
  BEARER_CSD  = 0x01,
  BEARER_USSD = 0x02,
  BEARER_GPRS = 0x03
} CatBearer;
```

6.12.32. CatOpenChannelOptions

```
typedef enum
{
  ON_DEMAND_LINK_ESTABLISHMENT = 0x00,
  IMMEDIATE_LINK_ESTABLISHMENT = 0x01
} CatOpenChannelOptions;
```

6.12.33. CatAddressType

```
typedef struct
{
  enum
  {
    IPV4 = 0x21,
    IPV6 = 0x97
  } AddressType;
  BYTE AddressLength;
  const void *Address;
} CatAddressType;
```

Annex A (Informative), example

```
/**
* Example of Toolkit Application
*/
#pragma AID  A0000000090001
#include <stdlib.h>
#include "application.h"
#include "gsm.h"
#include "gsmlow.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 world2";
static char textGInput[] = "Your name?";
BYTE ItemIdentifier;
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);
                          // 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
                            // get the data from the received SMS
                            CatOpenEnvelope(&result);
                            // get the offset of the data in the TP-UD field
                            CatGetData(1);
                            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
                            // get the data from the received SMS
                            CatOpenEnvelope(&result);
```

```
                // get the offset of the data in the TP-UD field
                CatGetData(1);
                bufptr=CatGetData(sizeof(menuTitle));
                memcpy(bufptr,memuTitle,sizeof(menuTitle));
            }
          }
        }
        break;
      default:
        CatExit();
        break;
    }
  CatExit();
}
```

History

Document history		
V0.0.0	August 2000	First draft for comment
V0.0.1	November 2000	Revised first draft, containing typographical and grammatical amendments and alterations.
V0.0.2	January 2001	Revised to present a 'C'-language bindings as the main document. MULTOS implementation detail moved to Annex.
V0.1.0	March 2001	Significant restructuring and changes to make the C binding completely platform independent.
V0.1.1	May 2001	Reworked after meeting T3 ad hoc #34, Edinburgh, according to the meeting report.
V0.2.0	October 2001	Reorganisation, reworked towards 3G , changed Sim to Cat, changed order of some function-parameters so all are in the order 'length, value'
V0.3.0	October 2001	Added lots of explanation to explain the scope of this API, added a C version of the Java example in 03.19
V1.0.0	December 2001	Added specification number. For presentation to TSG-T #14 for information.