Low Level Design Specification

SixTIes SAT Command Support ACI

Department:	WTBU - Cellular Systems		
Creation Date:	2004-08-25		
Last Modified:	2004-09-24 by Nick Holt		
ID:	8462.743.04	Version:	1
Status:	Submitted	ECCN:	Not Applicable

0 Document Control

Texas Instruments Incorporated and / or its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products, software and services at any time and to discontinue any product, software or service without notice. Customers should obtain the latest relevant information during product design and before placing orders and should verify that such information is current and complete.

All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment. TI warrants performance of its products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI products, software and / or services. To minimize the risks associated with customer products and applications, customers should provide adequate design, testing and operating safeguards.

Any access to and / or use of TI software described in this document is subject to Customers entering into formal license agreements and payment of associated license fees. TI software may solely be used and / or copied subject to and strictly in accordance with all the terms of such license agreements.

Customer acknowledges and agrees that TI products and / or software may be based on or implement industry recognized standards and that certain third parties may claim intellectual property rights therein. The supply of products and / or the licensing of software do not convey a license from TI to any third party intellectual property rights and TI expressly disclaims liability for infringement of third party intellectual property rights.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products, software or services are used.

Information published by TI regarding third–party products, software or services does not constitute a license from TI to use such products, software or services or a warranty, endorsement thereof or statement regarding their availability. Use of such information, products, software or services may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the express written permission of TI.

0.1 Export Control Statement

Recipient agrees that it will not knowingly export or re-export, directly or indirectly, any product or technical data (as defined by the U.S, EU and other Export Administration Regulations) including software, or any controlled product restricted by other applicable national regulations, received from Disclosing party under this Agreement, or any direct product of such technology, to any destination to which such export or re-export is restricted or prohibited by U.S or other applicable laws, without obtaining prior authorisation from U.S. Department of Commerce and other competent Government authorities to the extent required by those laws. This provision shall survive termination or expiration of this Agreement.

According to our best knowledge of the state and end-use of this product or technology, and in compliance with the export control regulations of dual-use goods in force in the origin and exporting countries, this

technology is classified as given on the front page.

This product or technology may require export or re-export license for shipping it in compliance with certain countries regulations.

0.2 Document History

Date	Version	Status	Author
2004-08-25 Initial version.	0.1	Draft	Nick Holt
2004-09-17 Pre-Review Version	0.2	Draft	Nick Holt
2004-09-24 Updated from Review by Thomas Luettig	1	Draft	Nick Holt

0.3 References, Abbreviations, Terms

1. SixTIes SAT Command Support, High Level Design (Ref: Unknown)

Table of Contents

1 Introduction

This document provides a low level design of the changes required to

- Implement SixTIes application specific handling of certain proactive SIM commands and SIM events.
- Provide additional AT command interface functionality to support the customization request
- Provide additional AT command interface functionality to support modified Call and Short Message Control By SIM handling
- Provide handling to ensure that only agreed events will be supported in the MMI Terminal Profile

It will not reproduce the information already produced in the High Level Design (ref: 1), but will expand on the design concepts to give detailed description of the actual changes required.

2 AT Interface Modifications

In order to implement each of the required AT commands in the modem part, changes will be required to the ATI and the ACI functional areas.

2.1 Set a Customisation Mode : AT%CUST

The purpose of the AT%CUST command is to provide a mechanism for customer specific modifications to be applied in the ACI and other entities, without impacting on the generic nature of the software.

With this command the modem part switches to another mode with different SAT behaviour than before. This AT command must be received in the modem part earlier than either the AT%SATC command or the AT+CFUN command.

The sAT and qAT functional interface will be supported, although there is no requirement for the rAT interface.

2.1.1 Command and Interface Definition

In the file aci_cmh.h

- The T_ACI_AT_CMD enumeration will be expanded to include the entry AT_CMD_CUST
- The interface functions sAT_PercentCUST and qAT_PercentCUST will need to be prototyped
- The currently available customization modes will be encapsulated in an enumerated type, T_CUST_MOD, which will provide the values
    - CUST_NORMAL_BEHAVIOUR and
    - CUST_MODE_BEHAVIOUR_1

In the file ati_cmd.c

- The interface functions setatPercentCUST and queatPercentCUST will be both prototyped and implemented (defined below)
- The local constant array *cmds* will be expanded with an entry for the new command AT_CMD_CUST using the function setatPercentCUST and queatPercentCUST.

2.1.2 Global Data

2.1.2.1 Description

The ACI currently supports several channels being simultaneously connected. In theory, each channel may indicate that a different Customisation Mode is to apply. However, in practice, the entire system will need to run in a single mode. In order to achieve this, each channel may request its own Custmisation Mode, however the system will use the Customisation Mode of the channel that causes the ME to change into a fully functional state.

In order to enable this functionality a new variable, cust_mode, will be added to the source specific SIM Shared Parameter data, and a variable overall_cust_mode will be added to the system wide SIM Shared Parameter data. These variables will be added in the file psa_sim.h to the T_SIM_SET_PRM and T_SIM_SHRD_PRM structures respectively.

Customisation may be requested only until the ME is put into its Fully Functional state, after which the overall_cust_mode will be applied and no modifications will be possible. In order to facilitate this a LOCAL variable, *Cust_Mode_Set*, will be initialized to FALSE and set to TRUE only when the overall_cust_mode is

initialized. In order for other functional areas to determine the state of *Cust_Mode_Set*, a function psaSIM_hasCustModeBeenSet() (see Para 2.1.5) will be provided.

2.1.2.2 Initialisation

In the function psaSIM_Init(), in file psa_simf.c, functionality must be added to initalise the cust_mode field for each source channel, along with the overall_cust_mode, to a default value of CUST_NORMAL_BEHAVIOUR.

2.1.3 AT Functions

2.1.3.1 sAT_PercentCUST

2.1.3.1.1 Prototype

T_ACI_RETURN sAT_PercentCUST(T_ACI_CMD_SRC srcId, T_CUST_MOD customisation_mode);

2.1.3.1.2 Implementation

1. Check the incoming *srcId*, if it is invalid then
   - log an error and return AT_FAIL
2. If a call to the function psaSIM_hasCustModeBeenSet() does not return FALSE
   - log an error and return AT_FAIL
3. Switch on the *cutsomisation_mode*
   - for CUST_NORMAL_BEHAVIOUR or CUST_MODE_BEHAVIOUR_1, set the cust_mode for the given *srcId* channel
   - for all other values log an error and return AT_FAIL
4. Return AT_CMPL

2.1.3.2 qAT_PercentCUST

2.1.3.2.1 Prototype

T_ACI_RETURN qAT_PercentCUST(T_ACI_CMD_SRC srcId, T_CUST_MOD *customisation_mode);

2.1.3.2.2 Implementation

1. Check the incoming *srcId*, if it is invalid then
   - log an error and return AT_FAIL
2. Set the contents of the variable pointed to by *customisation_mode* to the value of cust_mode for the given *srcId* channel
3. Return AT_CMPL

2.1.4 ATI Functions

2.1.4.1 setatPercentCUST

2.1.4.1.1 Prototype

T_ATI_RSLT setatPercentCUST (char *cl, UBYTE srcId)

2.1.4.1.2 Implementation

1. Switch on the contents of *cl*
   - For ‘0’, call the function sAT_PercentCUST, passing in *srcId* and customization mode of CUST_NORMAL_BEHAVIOUR. If the result is ATI_FAIL then report the error and return ATI_FAIL
   - For ‘1’, call the function sAT_PercentCUST, passing in *srcId* and customization mode of CUST_MODE_BEHAVIOUR_1. If the result is ATI_FAIL then report the error and return ATI_FAIL
   - For all other values report an error and return ATI_FAIL
2. Return ATI_CMPL

2.1.4.2 quetatPercentCUST

2.1.4.2.1 Prototype

T_ATI_RSLT queatPercentCUST (char *cl, UBYTE srcId)

2.1.4.2.2 Implementation

1. Declare a local automatic variable *mode*
2. Call the function qAT_PercentCUST , passing in *srcId* and the address of *mode*. If the function returns ATI_FAIL then report the error and return ATI_FAIL
3. Compose a string of the form %CUST: <*mode*> and use the function io_sendMessage to write it to the UART.
4. Return ATI_CMPL

2.1.5 PsaSIM_hasCustModeBeenSet

2.1.5.1 Prototype

BOOL psaSIM_hasCustModeBeenSet(void)

2.1.5.2 Implementation

Return the value of LOCAL variable Cust_Mode_Set

2.2 Enable / Disable Call or Short Message Control By SIM : AT%SATCC

The purpose of the AT%SATCC command is to allow the MMI to enable or disable Call Control By SIM, or the Short Message Control By SIM, functionality within the modem part.

The sAT and qAT functional interface will be supported, although there is no requirement for the rAT interface.

2.2.1 Command and Interface Definition

In the file aci_cmh.h

- The T_ACI_AT_CMD enumeration will be expanded to include the entry AT_CMD_SATCC
- The interface functions sAT_PercentSATCC and qAT_PercentSATCC will need to be prototyped
- The possible states will be encapsulated in an enumerated type, T_SAT_CC_MOD, which will provide the values
    - SATCC_CONTROL_BY_SIM_INACTIVE and
    - SATCC_CONTROL_BY_SIM_ACTIVE

In the file ati_cmd.c

- The interface functions setatPercentSATCC and queatPercentSATCC will be both prototyped and implemented (defined below)
- The local constant array *cmds* will be expanded with an entry for the new command AT_CMD_SATCC using the function setatPercentSATCC and queatPercentSATCC.

2.2.2 Global Data

2.2.2.1 Description

The ACI currently supports several channels being simultaneously connected. In theory, each channel may indicate that a different Call Control By SIM Mode is to apply, although this should not be possible.

In order to enable this functionality a new variable, sat_cc_mode, will be added to the source specific SIM Shared Parameter data. The variable will be added in the file psa_sim.h to the T_SIM_SET_PRM structure.

2.2.2.2 Initialisation

In the function psaSIM_Init(), in file psa_simf.c, functionality must be added to initalise the sat_cc_mode field for each source channel to a default value of SATCC_CONTROL_BY_SIM_ACTIVE.

2.2.3 AT Functions

2.2.3.1 sAT_PercentSATCC

2.2.3.1.1 Prototype

T_ACI_RETURN sAT_PercentSATCC(T_ACI_CMD_SRC srcId, T_SAT_CC_MOD sat_cc_mode)

2.2.3.1.2 Implementation

1. Check the incoming *srcId*, if it is invalid then
   - log an error and return AT_FAIL
2. Switch on the *sat_cc_mode*:
   - for SATCC_CONTROL_BY_SIM_INACTIVE or SATCC_CONTROL_BY_SIM_ACTIVE, set the sat_cc_mode for the given *srcId* channel
   - for all other values log an error and return AT_FAIL
3. Return AT_CMPL

2.2.3.2 qAT_PercentSATCC

2.2.3.2.1 Prototype

T_ACI_RETURN qAT_PercentSATCC(T_ACI_CMD_SRC srcId, T_SAT_CC_MOD *sat_cc_mode)

2.2.3.2.2 Implementation

1. Check the incoming *srcId*, if it is invalid then
   - log an error and return AT_FAIL
2. Set the contents of the variable pointed to by *sat_cc_mode* to the value of sat_cc_mode for the given *srcId* channel
3. Return AT_CMPL

2.2.4 ATI Functions

2.2.4.1 setatPercentSATCC

2.2.4.1.1 Prototype

T_ATI_RSLT setatPercentSATCC (char *cl, UBYTE srcId)

2.2.4.1.2 Implementation

1. Switch on the contents of *cl*
   - For ‘0’, call the function sAT_PercentSATCC, passing in *srcId* and sat_cc_mode of SATCC_CONTROL_BY_SIM_INACTIVE. If the result is AT_FAIL then report the error and return AT_FAIL
   - For ‘1’, call the function sAT_PercentSATCC, passing in *srcId* and sat_cc_mode of SATCC_CONTROL_BY_SIM_ACTIVE. If the result is AT_FAIL then report the error and return AT_FAIL
   - For all other values report an error and return AT_FAIL
2. Return AT_CMPL

2.2.4.2 quetatPercentSATCC

2.2.4.2.1 Prototype

T_ATI_RSLT queatPercentSATCC (char *cl, UBYTE srcId)

2.2.4.2.2 Implementation

1. Declare a local automatic variable *mode*
2. Call the function qAT_PercentSATCC , passing in *srcId* and the address of *mode*. If the function returns AT_FAIL then report the error and return AT_FAIL
3. Compose a string of the form %SATCC: <*mode*> and use the function io_sendMessage to write it to the UART.
4. Return AT_CMPL

2.2.5 Reset to Default

In order to ensure that Call or Short Message Control By SIM is not inadvertently turned off permanently whenever the sat_cc_mode for a given *srcId* is checked, and Call or Short Message Control is skipped the *sat_cc_mode* parameter will be reset to the default value of SATCC_CONTROL_BY_SIM_ACTIVE.

2.2.6 Call and Short Message Control modifications

2.2.6.1 Call Control

In the file cmh_satf.c modify the function cmhSAT_CalCntrlBySIM, which checks whether Call Control By SIM is required, so that the first check is whether the *sat_cc_mode* (See Para: 2.2.2) of the ACI channel related to this call request is SATCC_CONTROL_BY_SIM_INACTIVE, if so return AT_CMPL.

2.2.6.2 SS Control

In the file cmh_satf.c modify the function cmhSAT_SSCntrlBySIM, which checks whether Call Control By SIM is required, so that the first check is whether the *sat_cc_mode* (See Para: 2.2.2) of the ACI channel related to this SS request is SATCC_CONTROL_BY_SIM_INACTIVE, if so return AT_CMPL.

2.2.6.3 USSD Control

In the file cmh_satf.c modify the function cmhSAT_USSDCntrlBySIM, which checks whether Call Control By SIM is required, so that the first check is whether the *sat_cc_mode* (See Para: 2.2.2) of the ACI channel related to this USSD request is SATCC_CONTROL_BY_SIM_INACTIVE, if so return AT_CMPL.

2.2.6.4 Short Message

2.2.6.4.1 cmhSMS_SMReadCMSS

In the file cms_smsr.c, in function cmhSMS_SMReadCMSS, modify the check before the call to cmhSAT_MoSmCntr from:

- "if (psaSIM_ChkSIMSrvSup(SRV_MOSMCtrlSIM))"

to:

- "if (psaSIM_ChkSIMSrvSup(SRV_MOSMCtrlSIM) AND
  (simShrdPrm.setPrm[srcId].sat_cc_mode EQ SATCC_CONTROL_BY_SIM_ACTIVE)) "

2.2.6.4.2 sAT_PlusCMGS_Gl

In the file cms_smss.c, in function sAT_PlusCMGS_Gl, modify the check before the call to cmhSAT_MoSmCntr from:

- "if (psaSIM_ChkSIMSrvSup(SRV_MOSMCtrlSIM))"

to:

- "if (psaSIM_ChkSIMSrvSup(SRV_MOSMCtrlSIM) AND
  (simShrdPrm.setPrm[srcId].sat_cc_mode EQ SATCC_CONTROL_BY_SIM_ACTIVE)) "

2.2.6.4.3 sAT_PlusCMGSPdu

In the file cms_smss.c, in function sAT_PlusCMGSPdu, modify the check before the call to cmhSAT_MoSmCntr from:

- "if (psaSIM_ChkSIMSrvSup(SRV_MOSMCtrlSIM))"

to:

- "if (psaSIM_ChkSIMSrvSup(SRV_MOSMCtrlSIM) AND
  (simShrdPrm.setPrm[srcId].sat_cc_mode EQ SATCC_CONTROL_BY_SIM_ACTIVE)) "

2.2.6.4.4 sAT_PlusCMSS_Gl

In the file cms_smss.c, in function sAT_PlusCMSS_Gl, modify the check before the call to cmhSAT_MoSmCntr from:

- "if (psaSIM_ChkSIMSrvSup(SRV_MOSMCtrlSIM))"

to:

- "if (psaSIM_ChkSIMSrvSup(SRV_MOSMCtrlSIM) AND
  (simShrdPrm.setPrm[srcId].sat_cc_mode EQ SATCC_CONTROL_BY_SIM_ACTIVE)) "

3 SIM SAP Modifications

The SIM SAP interface is defined in the file sim.sap, and is edited using the SAPE editor

3.1 New Basic Elements

Two new basic elements are required:

- U8 cust_mode - Customisation flag with valid values of 0 or 1
- U8 user_accepts - Treated as a Boolean with the value TRUE or FALSE

3.2 SIM_ACTIVATE_REQ

This is an existing signal that needs to be updated to contain a new element that will be linked to the *cust_mode* Basic Element.

3.3 SIM_REFRESH_USER_RES

This is a new DOWNLINK signal, which will contain 2 parameters. The first will be linked to the new Basic Element *user_accepts*, while the second will be linked to the existing Structure Element *stk_cmd*

4 Startup Procedure

Where an MMI is to request SixTIes Customisation, it must do so before reporting the Terminal Profile to the ACI, and before performing an AT+CFUN to start the SIM functionality.

4.1 MMI Terminal Profile Handling

The MMI sends its Terminal Profile to the ACI in an AT%SATC command, the profile contains at maximum 20*8 entrys. See at GSM 11.14 for more details. The fields supported in a SixTIes Profile are given in detail in the High Level Design document (ref :1), and are not reproduced here.

The detailed changes required to process the SixTIes Terminal profile are as follows.

4.1.1 Definition of a Mask

In the file psa_sat.h, a constant UBYTE array, satMaskCust1Prf, will be defined which will hold a valid Maximal Terminal Profile mask for the SixTIes MMI. It will be defined as:

```
const GLOBAL UBYTE satMaskCust1Prfl[MAX_STK_PRF] =
            {
             SAT_TP1_PRF_DNL |   /* Profile Download */
             SAT_TP1_MENU_SEL,   /* Class 2: Menu Selection */
             SAT_TP2_CMD_RES |   /* Command Result */
             SAT_TP2_CC      |   /* Class 2: Call Control by SIM */
             SAT_TP2_MOSMC   |   /* Class 3: MO Short Message Control */
             SAT_TP2_ALPHA_ID  |/* Class 2: Alpha Id Handling 9.1.3  */
             SAT_TP2_UCS2_ENTRY |/* Class 2: UCS2 Entry supported     */
             SAT_TP2_UCS2_DSPL  |/* Class 2: UCS2 Display supported   */
             SAT_TP2_DSPL_EXT,   /* Class 3: Display of extended text */
             SAT_TP3_DSPL_TXT   |/* Class 2: DISPLAY TEXT */
             SAT_TP3_GET_INKEY  |/* Class 2: GET INKEY */
             SAT_TP3_GET_INPUT  |/* Class 2: GET INPUT */
             SAT_TP3_PLAY_TONE  |/* Class 2: PLAY TONE */
             SAT_TP3_REFRESH,    /* Class 2: REFRESH */
             SAT_TP4_SEL_ITEM   |/* Class 2: SELECT ITEM */
             SAT_TP4_SEND_SS    |/* Class 2: SEND SS */
             SAT_TP4_SEND_USSD  |/* Class 3: SEND USSD */
             SAT_TP4_SETUP_CALL |/* Class 2: SETUP CALL */
             SAT_TP4_SETUP_MENU, /* Class 2: SETUP MENU */
             SAT_TP5_EVENT_LIST |/* Class 3: SETUP EVENT LIST */
             SAT_TP5_USER_ACT   |/* Class 3: User activity */
             SAT_TP5_SCR_AVAIL,  /* Class 3: Idle Screen available */
             SAT_TP6_LANG_SEL,
             0x00,
             SAT_TP8_BIN_GET_INKEY |/* Class 3: Binary Choice in GET INKEY */
             SAT_TP8_IDLE_TXT      |/* Class 3: SETUP IDLE MODE TEXT */
             SAT_TP8_AI2_SETUP_CALL, /* Class 3: 2nd alpha identifier in SETUP CALL */
#if defined (WAP)
             SAT_TP9_SUST_DSPL_TXT |  /* Class 3: Sustained DISPLAY TEXT */
             SAT_TP9_LAUNCH_BROWSER,  /* Class 3: LAUNCH BROWSER */
#else
             SAT_TP9_SUST_DSPL_TXT,   /* Class 3: Sustained DISPLAY TEXT */
#endif
             SAT_TP10_SFTKEY_SEL_ITEM |  /* soft key support SELECT ITEM */
             SAT_TP10_SFTKEY_SETUP_MENU, /* soft key support SET UP MENU */
             SAT_TP11_MAX_NR_SFTKEY,     /* number of soft keys available */
```

```
            0x00,
            0x00,
            SAT_TP14_NR_OF_CHAR_DSPL_DWN| /* number of characters supported down ME display */
            SAT_TP14_SCRN_SIZE_PARAM,     /* screen sizing parameter supported */
            SAT_TP15_NR_OF_CHAR_DSPL_ACRS|/* number of characters supported across ME display */
            SAT_TP15_VAR_SIZE_FONT,       /* variable size fonts supported */
            SAT_TP16_DSPL_RESIZE |        /* display can be resized */
            SAT_TP16_TEXT_WRAP |          /* text wrapping supported */
            SAT_TP16_TEXT_SCROLL |        /* text scrolling supported */
            SAT_TP16_WIDTH_RDCT_MENU,     /* width reduction in a menu */
            0x00,
            0x00,
            0x00,
            0x00
}
```

4.1.2 Masking the Incoming Profile

In function sAT_PercentSATC the incoming profile is already masked with a default MMI Maximal Terminal Profile. The function has to be modified such that, depending on the customisation mode specified, either the existing satMask MMIPrfl or the new satMaskCust1Prfl will be used.

4.2 SIM Activate

With the system wide field *overall_cust_mode*, and the ACI channel specific variable *cust_mode*, already defined (see Para: 2.1.2), the customisation mode must be set and transmitted to the SIM Entity as part of the Startup Procedure.

The function psaSIM_activateSIM, defined in file psa_sims.c, must be modified such that

- the value of *overall_cust_mode* is set prior to sending the SIM_ACTIVATE_REQ primitive, and
- the value of *overall_cust_mode* should be copied to the cust_mode field in the primitive (see Para: 3.2)

This is achieved as follows:

- If the required action is SIM_INITIALISATION then
    - If a call to the function psaSIM_hasCustModeBeenSet() returns FALSE then set the value of *overall_cust_mode* to the value of *cust_mode* as defined for the activating ACI channel.
    - Set the value of the cust_mode field in the primitive to the value of *overall_cust_mode*
- Otherwise
    - If a call to the function psaSIM_hasCustModeBeenSet() returns FALSE then set the value of cust_mode field in the primitive to CUST_NORMAL_BEHAVIOUR, otherwise set it to the value of *overall_cust_mode*

5 STK Cmd Processing

5.1 Store the Command

This is needed for the SixTIes handling of the SetupCall, SendSS, SendUSSD and SendSMS STK Commands, because the Modem Part must perform the Call or Short Message Control By SIM functionality first and send the result to the MMI before sending the original command PDU. To this end a two fields must be added to the satShrdParm global data, and a function, cmhSAT_Cust1StkCmdInd, must be created to forward the Cmd to the MMI (See Para: 5.2.3.2.2)

The actual storing of the original data is done as part of the **Modified Cmd Processing for SixTIes** (See Para: 5.2)

5.1.1 SatShrdParm Modifications

In the file psa_sat.h, the structure T_SAT_SHRD_PRM must be updated to add the fields:

- UBYTE *cust1StkCmd
- USHORT cust1StkCmdLen

5.2 Modified Cmd Processing for SixTIes

5.2.1 Modifications to function psa_sim_toolkit_ind

If the *overall_cust_mode* is CUST_MODE_BEHAVIOUR_1, then the priority for processing the command is placed with the MMI first and then the ACI, rather than the other way around. This has given rise to the following changes.

In the file psa_satp.c, the function psa_sim_toolkit_ind must be modified such that:

- if *overall_cust_mode* is CUST_MODE_BEHAVIOUR_1 then the function will switch on the cmd_type
    - ***for SetupCall, SendSS, SendUSSD, and SendSMS***, then the incoming command must be stored before continuing. In order to do this, the pointer cust1StkCmd (See Para: 5.1.1) must be checked to determine whether it is NULL.
    - If the pointer is not NULL, then the allocated memory must be freed.
    - Once the cust1StkCmd pointer is NULL, the function will attempt to allocate enough dynamic memory to store the received sim_toolkit_ind primitive and will memcpy the data into it, setting the value of cust1StkCmdLen as well. If the function is unable to allocate enough memory for the operation, then a Terminal Response of "ME Unable to Process Cmd" will be returned to the SIM and the processing will end.
    - If the allocation and copy were successful the processing will continue without a 'break"
    - ***For SetupEventList*** (and the previous commands) the incoming command will be passed to the function psaSAT_dasmMECmd for processing in the ACI. Before being passed in a SATI indication to the MMI (See Paras: to for further details)
    - ***For Refresh*** a flag will be set to indicate that the ACI is expecting a User Response in the form of a SATR indication, and the command will be passed to the MMI for processing.

- ***For all other Stk Commands*** a new function, cmhSAT_IsStkCmdForMmi, will be called to determine whether the command should be passed to the MMI for Processing. Finally, if the command remains unhandled, the function will determine whether it may be handled within the ACI. If not, a Terminal Response of "Beyond ME Capabilities" will be returned to the SIM

5.2.2 Free Memory Allocated to Store Stk cmd

In the file psa_satf.c, modify the function psaSAT_SendTrmResp, so that when a terminal response is sent to the SIM Entity, the pointer *cust1StkCmd* (See Para: 5.1.1) will be checked to determine whether it is NULL. If the pointer has memory allocated (not NULL) then the allocated memory will be freed.

This is a "catch-all" scenario and ensures that the memory cannot be left allocated and lost.

5.2.3 New Functions

5.2.3.1 cmhSAT_GetCmdPrfLocation

5.2.3.1.1 Description

This is a new local function which will determine the location of the BIT in the Terminal Profile that will indicate whether the passed command is supported or not. It will return TRUE if the command has been found, and FALSE if the command (or qualifier) is unknown. If the function has returned TRUE then the Byte Location and Byte Mask are returned in the passed parameters. A return value of TRUE, with the Byte Location and Mask both set to 0xFF, will indicate that the command is an END_OF_SESSION which is not in the profile but should also be sent to the MMI.

5.2.3.1.2 Prototype

LOCAL BOOL cmhSAT_GetCmdPrfLocation (U8 cmd_typ, U8 cmd_qual,

U8 *prf_byte_no, U8 *byte_mask)

5.2.3.1.3 Implementation

- Initialise the contents of both *prf_byte_no* and *byte_mask* to 0xFF
- Switch of the value of cmd_typ and set the contents of *prf_byte_no* and *byte_mask* accordingly (for example : for SAT_CMD_REFRESH, *prf_byte_no=2 and *byte_mask=SAT_TP3_REFRESH) repeat for all known values of SAT_CMD_?. For SAT_CMD_PROV_LOC_INFO, the function will also have to check the value of the command qualifier before it will be possible to determine which Byte and Bits have to be checked.
- The ***default*** for the Switch statement will return FALSE, otherwise the return value will be TRUE.

5.2.3.2 cmhSAT_IsStkCmdForMmi

5.2.3.2.1 Description

This function will check whether the command should be set to the MMI and if so will trigger a SATI indication to be sent. It will return TRUE if the command is sent to the MMI, otherwise it will return FALSE.

5.2.3.2.2 Prototype

BOOL cmhSAT_IsStkCmdForMmi (U8 cmd_typ, U8 cmd_qual)

5.2.3.2.3 Implementation

- Define an automatic BOOL variable *is_handled* and initialise it to FALSE
- Define automatic U8 variables *prf_byte_no* and *byte_mask*
- If cmd_typ is SAT_CMD_END_SESSION then
    - set *is_handled* to TRUE
- Otherwise
    - Call function cmhSAT_GetCmdPrfLocation, passing in cmd_typ, cmd_qual and the addresses of *prf_bye_no* and *byte_mask*.
    - If the function returns TRUE then for each valid channel do the following
        - Mask the appropriate BYTE of the StkProf for the channel, and if the BIT is set, then change *is_handled* to the value TRUE.
- If *is_handled* is set to TRUE then call the function cmhSAT_STKCmdInd
- Return the value of *is_handled*

5.3 Forward the Original STK Cmd to the MMI

This is a requirement for processing a SetupCall, SendSS, SendUSSD or SendSMS command. The original command will have been preserved in the *cust1StkCmd* (See Para: 5.1.1) memory and may be sent on receipt of the Call or Short Message Control By SIM Response.

A new function, cmhSAT_Cust1StkCmdInd, has been written to provide the functionality.

5.3.1 cmhSAT_Cust1StkCmdInd

5.3.1.1 Prototype

SHORT cmhSAT_Cust1StkCmdInd (void)

5.3.1.2 Implementation

- If the pointer *cust1StkCmd* (See Para: 5.1.1) is not NULL then
    - Send the contents of *cust1StkCmd* in a SATI to each valid command source Id
    - Free the memory allocated to pointer *cust1StkCmd*
- Return 0

5.4 Sending of a SATN to the MMI

Under normal behaviour, when the Modem Part receives a SetupCall, SendSS, SendUSSD or SendSMS, command it is processed in function psaSAT_dasmMECmd. One of the first actions by the function will be to send a SATN notification to each connected ACI channel. However, for the SixTIes MMI, a SATI indication will be sent after the appropriate Call or Short Message Control By SIM action has been taken. As such, the function psaSAT_dasmMECmd will be changed so that the check, that the cmd_type is NOT EventList, to determine that the SATN should be sent will be changed to be "(cmd_typ is NOT EventList) AND (*overall_cust_mode* is CUST_NORMAL_BEHAVIOUR)"

6 Setup Call Stk Cmd

6.1 Introduction

For a SixTIes MMI the ACI captures the SetUpCall Command and, provided Call Control By SIM is active, immediately sends a Call Control by SIM request to the SIM. The resultant response is then forwarded through to the MMI in a %SATN notification. The ACI with then forward the original Setup Call command to the MMI in a %SATI indication. For an Emergency Call Request, Call Control By SIM is not required, and the %SATI indication will be sent immediately.

6.2 Handle an Emergency Call

The function cmhSAT_setupCall, in file cmh_satf.c, will be modified such that if the requested Call Setup is for an Emergency Call, and the MMI is a SixTIes MMI, then the call table entry, created at the start of the process, will be cleared and the original Toolkit Command will be sent to the MMI in a SATI inidication immediately.

6.3 Trigger the Call Control By SIM request

Handled automatically in function cmhSAT_setupCall, no changes required.

6.4 Don't notify the user of the call setup request

If Call Control by SIM is not Allocated and Active in the SIM Service Table then the function cmhSAT_setupCall currently checks to determine whether the Command contains details for the User to be notified. If so, a SATA notification is sent to the MMI. However, If the MMI is a SixTIes MMI, then the SATA notification should not be generated but, instead, the original command should be forwarded to the MMI in a SATI notification.

6.5 Notify the MMI of the Call Control Result

The function psa_sim_toolkit_cnf currently sends a SATN notification to the MMI, there are no changes required.

6.6 ACI response to Setup Call, Call Control Result

The function cmhSAT_ResCalCntrlBySim, processes the Call Control Result in the ACI. This function should be modified so that if *overall_cust_mode* is CUST_MODE_BEHAVIOUR_1 and the command was a SIM Originated Command then

- If the CC Result is "Not Allowed"
    - send a Terminal Response of "ME Unable to Process Cmd"
- Otherwise
    - Call the new function cmhSAT_Cust1StkCmdInd (See Para: 5.3) to forward the original Stk Cmd to the MMI
- Clear the Call Table Entry which was allocated at the start of the process
- return from the function

7 Send SS Stk Cmd

7.1 Introduction

For a SixTIes MMI the ACI captures the Send SS Command and, provided Call Control By SIM is active, immediately sends a Call Control by SIM request to the SIM. The resultant response is then forwarded through to the MMI in a %SATN notification. The ACI with then forward the original Send SS command to the MMI in a %SATI indication.

7.2 Trigger the Call Control By SIM request

Handled automatically in function cmhSAT_SendSS, no changes required.

7.3 Call Control By SIM Not Active

If Call Control by SIM is not Allocated and Active in the SIM Service Table then the function cmhSAT_SendSS currently triggers the SS Command to be sent to the network. If the MMI is a SixTIes MMI, then the SS command should not be generated by the ACI, but the original command should be forwarded to the MMI in a SATI notification.

7.4 Notify the MMI of the Call Control Result

See Para 6.5

7.5 ACI response to Send SS, Call Control Result

The function cmhSAT_ResSSCntrlBySim, processes the Send SS Call Control Result in the ACI. This function should be modified so that if *overall_cust_mode* is CUST_MODE_BEHAVIOUR_1 and the command was a SIM Originated Command then

- If the CC Result is “Not Allowed”
    - send a Terminal Response of “ME Unable to Process Cmd”
- Otherwise
    - Call the new function cmhSAT_Cust1StkCmdInd (See Para: 5.3) to forward the original Stk Cmd to the MMI
- Clear the Call Table Entry which was allocated at the start of the process
- return from the function

8 Send USSD Stk Cmd

8.1 Introduction

For a SixTIes MMI the ACI captures the Send USSD Command and, provided Call Control By SIM is active, immediately sends a Call Control by SIM request to the SIM. The resultant response is then forwarded through to the MMI in a %SATN notification. The ACI with then forward the original Send SS command to the MMI in a %SATI indication.

If Call Control by SIM is not Allocated and Active in the SIM Service Table then the function cmhSAT_SendUSSD currently triggers the USSD Command to be sent to the network. If the MMI is a SixTIes MMI, then the USSD command should not be generated by the ACI, but the original command should be forwarded to the MMI in a SATI notification.

8.2 Trigger the Call Control By SIM request

Handled automatically in function cmhSAT_SendUSSD, no changes required.

8.3 Notify the MMI of the Call Control Result

See Para 6.5

8.4 ACI response to Send USSD, Call Control Result

The function cmhSAT_ResUSSDCntrlBySim, processes the Send USSD Call Control Result in the ACI. This function should be modified so that if *overall_cust_mode* is CUST_MODE_BEHAVIOUR_1 and the command was a SIM Originated Command then

- If the CC Result is “Not Allowed”
    - send a Terminal Response of “ME Unable to Process Cmd”
- Otherwise
    - Call the new function cmhSAT_Cust1StkCmdInd (See Para: 5.3) to forward the original Stk Cmd to the MMI
- return from the function

9 Send SMS Stk cmd

9.1 Introduction

For a SixTIes MMI the ACI captures the Send SM Command and, provided MO Short Message Control By SIM is active, sends an MO SM Control by SIM request to the SIM. The resultant response is then forwarded through to the MMI in a %SATN notification. The ACI with then forward the original Send SM command to the MMI in a %SATI indication.

9.2 Trigger the MO Short Message Control By SIM request

Handled automatically in function sAT_PlusCMGSPdu (called from cmhSAT_SendSM), no changes required.

9.3 Call Control By SIM Not Active

If MO Short Message Control by SIM is not Allocated and Active in the SIM Service Table then the function sAT_PlusCMGSPdu currently sends the SMS_SUBMIT to the network. If the MMI is a SixTIes MMI and the srcId indicates that the request is a SIM Toolkit Requuest, then the SMS_SUBMIT should not be generated by the ACI, but the original command should be forwarded to the MMI in a SATI notification, and the global parameters reset to release the ACI SMS processing.

9.4 Notify the MMI of the MO Short Message Control Result

See Para 6.5

9.5 ACI response to the MO Short Message Control Result

The function cmhSAT_ResSMCntrlBySim, processes the MO Short Message Control Result in the ACI. This function should be modified so that if *overall_cust_mode* is CUST_MODE_BEHAVIOUR_1 and the command was a SIM Originated Command then

- If the CC Result is “Not Allowed”
  - send a Terminal Response of “ME Unable to Process Cmd”
- Otherwise
  - Call the new function cmhSAT_Cust1StkCmdInd (See Para: 5.3) to forward the original Stk Cmd to the MMI
- Ensure that the SMS global parameters are reset to release the ACI SMS processing
- return from the function

10 Send DTMF Stk Cmd

10.1 Introduction

This command will be captured by the new function cmhSAT_IsStkCmdForMmi and will, if supported, be sent in a SATI indication to the MMI for processing.

No further modifications will be needed (See Para: 5)

11 Refresh Stk Cmd

11.1 Introduction

This command is captured in function psa_sim_toolkit_ind and processed as described in para.: 5.2.1. Further modifications to the Modem Part are required, to handle the Response from the MMI that needs to be forwarded to the SIM Entity.

11.2 Global Data

11.2.1 Description

Because the MMI will respond to a SIM Toolkit Refresh command using a SATR to indicate the user response, and the fact that the SATR is normally used to indicate a Terminal Response, the ACI will require a flag to indicate that the received SATR is to be treated in a different manner. As a result a UBYTE flag, *cust1SimRefreshRespRqd*, will be added to the T_SAT_SHRD_PARM structure, which will add it, as an additional field, to the *satShrdPrm* global variable.

11.2.2 Initialisation

In the function psaSAT_Init(), in file psa_satf.c, functionality must be added to initalise the field *cust1SimRefreshRespRqd* to a default value of FALSE.

11.3 CheckRefreshUserResp

11.3.1 Description

This is a local function that will parse the Terminal Response data from the MMI to determine whether the User has accepted or rejected the SIM Refresh request. The function will return TRUE or FALSE accordingly.

11.3.2 Prototype

BOOL checkRefreshUserResp (UBYTE *p)

11.3.3 Implementation

- Define a local constant UBYTE array, *TermRespRefreshOk*, to provide a skeleton for comparison.
- Compare the first 2 bytes of *p* and *TermRespRefreshOk*. If they are the same then
    - If the command number is 0x01 (Refresh) AND the device details match AND the result is 0x00 (Ok) then
        - Return TRUE
    - Othewise return FALSE
- Otherwise return FALSE

11.4 psaSAT_SendRefreshUserRes

11.4.1 Description

This is to be the interface function responsible for sending a SIM_REFRESH_USER_RES, the initial response to a SIM Toolkit Refresh command.

11.4.2 Prototype

void psaSAT_SendRefreshUserRes (void)

11.4.3 Implementation

- Use the PALLOC macro to allocate a SIM_REFRESH_USER_RES signal
- Call the local function *checkRefreshUserResp* to set the *user_accepts* parameter of the signal to the return value. The SIM Toolkit Response will be stored in *satShrdParm*
- Fill the *stk_cmd* parameter of the signal with the response data from *satShrdParm*
- Send the SIM_REFRESH_USER_RES primitive to the SIM Entity.

11.5 Receipt of the SATR indication

The function sAT_PercentSATR, in file cmh_sats.c, will need to be modified such that after the preliminary checks have been completed to ensure a valid response, if the *overall_cust_mode* is CUST_MODE_BEHAVIOUR_1 and the *cust1SimRefreshRespRqd* flag is TRUE then the function psaSAT_SendRefreshUserRes is called and the sAT_PercentSATR function will return AT_CMPL