



Dialogic® Converged Services Platform - SwitchKit® Development Environment

TCAP Interface User's Guide

Copyright and Legal Disclaimer

Copyright © [1998-2008] Dialogic Corporation. All Rights Reserved. You may not reproduce this document in whole or in part without permission in writing from Dialogic Corporation at the address provided below.

All contents of this document are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation or its subsidiaries. Reasonable effort is made to ensure the accuracy of the information contained in the document. However, due to ongoing product improvements and revisions, Dialogic Corporation and its subsidiaries do not warrant the accuracy of this information and cannot accept responsibility for errors or omissions that may be contained in this document.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH DIALOGIC® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS EXPLICITLY SET FORTH BELOW OR AS PROVIDED IN A SIGNED AGREEMENT BETWEEN YOU AND DIALOGIC, DIALOGIC ASSUMES NO LIABILITY WHATSOEVER, AND DIALOGIC DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF DIALOGIC PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHT OF A THIRD PARTY.

Dialogic products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

It is possible that the use or implementation of any one of the concepts, applications, or ideas described in this document, in marketing collateral produced by or on web pages maintained by Dialogic Corporation or its subsidiaries may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic Corporation or its subsidiaries do not provide any intellectual property licenses with the sale of Dialogic products other than a license to use such product in accordance with intellectual property owned or validly licensed by Dialogic Corporation or its subsidiaries. More detailed information about such intellectual property is available from Dialogic Corporation's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. The software referred to in this document is provided under a Software License Agreement. Refer to the Software License Agreement for complete details governing the use of the software.

Dialogic Corporation encourages all users of its products to procure all necessary intellectual property licenses required to implement any concepts or applications and does not condone or encourage any intellectual property infringement and disclaims any responsibility related thereto. These intellectual property licenses may differ from country to country and it is the responsibility of those who develop the concepts or applications to be aware of and comply with different national license requirements.

Dialogic, Dialogic Pro, Brooktrout, Cantata, SnowShore, Eicon, Eicon Networks, Eiconcard, Diva, SIPcontrol, Diva ISDN, TruFax, Realblobs, Realcomm 100, NetAccess, Instant ISDN, TRXStream, Exnet, Exnet Connect, EXS, ExchangePlus VSE, Switchkit, N20, Powering The Service-Ready

Network, Vantage, Connecting People to Information, Connecting to Growth, Making Innovation Thrive, and Shiva, among others as well as related logos, are either registered trademarks or trademarks of Dialogic.

The names of actual companies and products mentioned herein are the trademarks of their respective owners.

Dialogic Product Line Warranty

Unless otherwise stated in an applicable product purchase agreement between the Customer and Dialogic, Dialogic warrants that during the Warranty Period, products will operate in substantial conformance with Dialogic's standard published documentation accompanying the product. If a product does not operate in accordance therewith during the Warranty Period, the Customer must promptly notify Dialogic. Dialogic, at its option, will either repair or replace the product without charge. The Customer has the right, as their exclusive remedy, to return the product for a refund of purchase price or license fee if Dialogic is unable to repair or replace it.

Warranty Period

In the event that you have no signed agreement setting out a warranty period, the Warranty Period shall be the standard warranty period set out on www.dialogic.com on the date of your purchase of the product.

The Warranty Period begins on the date of shipment of any products or software by Dialogic.

The Warranty Period for repaired, replaced or corrected products and software shall be coterminous to the Warranty Provided for the original products or software purchased.

To report warranty claims, Customer may contact Dialogic via email at techsupport@cantata.com or call (781) 433-9600.

Warranty Provisions

A. During the Warranty Period, Dialogic warrants to Customer only that:

- (i) Products manufactured by Dialogic (including those manufactured for Dialogic by an original equipment manufacturer) will be free from defects in material and workmanship and will substantially conform to specifications for such products;
- (ii) software developed by Dialogic will be free from defects which materially affect performance in accordance with the specifications for such software. With respect to products or software or partial assembly of products furnished by Dialogic but not manufactured by Dialogic, Dialogic hereby assigns to Customer, to the extent permitted, the warranties given to Dialogic by its vendors of such items.

B. If, under normal and proper use, a defect or non conformity appears in warranted products or software during the applicable Warranty Period and Customer promptly notifies Dialogic in writing during the applicable warranty period of such defect or non conformance, and follows Dialogic's instructions regarding return of such defective or non conforming Product or Software, then Dialogic will, at no charge to Customer, either:

- (i) repair, replace or correct the same at its manufacturing or repair facility or
- (ii) if Dialogic determines that it is unable or impractical to repair, replace or correct the product or software, provide a refund or credit not to exceed the original purchase price or license fee.

C. No product or software will be accepted for repair or replacement without the written authorization of and in accordance with instructions from Dialogic. Removal and reinstallation expenses as well as transportation expenses associated with returning such product or software to Dialogic shall be borne by Customer. Dialogic shall pay the costs of transportation of the repaired or replaced product or software to the destination designated in the original Order. If Dialogic determines that any returned product or software is not defective, Customer shall pay Dialogic's costs of handling, inspecting, testing and transportation. In repairing or replacing any product, part of product, or software medium under this warranty, Dialogic may use new, remanufactured, reconditioned, refurbished or functionally equivalent products, parts or software media. Replaced products or parts shall become Dialogic's property.

D. Dialogic makes no warranty with respect to defective conditions or non conformities resulting from any of the following: Customer's modifications, misuse, neglect, accident or abuse; improper wiring, repairing, splicing, alteration, installation, storage or maintenance performed in a manner not in accordance with Dialogic's or its vendor's specifications, or operating instructions; failure of Customer to apply Dialogic's previously applicable modifications or corrections; or items not manufactured by Dialogic or purchased by Dialogic pursuant to its procurement specifications. Dialogic makes no warranty with respect to products which have had their serial numbers removed or altered; with respect to expendable items, including, without limitation, fuses, light bulbs, motor brushes and the like; or with respect to defects related to Customer's data base errors. Improper packaging of product for repair will not be covered under this warranty agreement. No warranty is made that software will run uninterrupted or error free.

E. Warranty does not include:

- a) Dialogic's assistance in diagnostic efforts;
- b) access to Dialogic's Technical Support web sites, databases or tools;
- c) product integration testing;
- d) on-site assistance; or
- e) product documentation updates.

These services are available either during or after warranty at Dialogic's published prices.

F. THE FOREGOING WARRANTIES ARE EXCLUSIVE & ARE GRANTED IN LIEU OF ALL OTHER EXPRESS & IMPLIED WARRANTIES (WHETHER WRITTEN, ORAL, STATUTORY OR OTHERWISE), INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. CUSTOMER'S SOLE AND EXCLUSIVE REMEDY AND DIALOGIC'S SOLE OBLIGATION HEREUNDER, SHALL BE TO REPAIR, REPLACE, CREDIT OR REFUND AS SET FORTH ABOVE.

G. IN NO EVENT SHALL DIALOGIC, ITS DIRECTORS, OFFICERS, EMPLOYEES, AGENTS OR AFFILIATES, BE LIABLE FOR ANY COSTS OR DAMAGES ARISING DIRECTLY OR INDIRECTLY FROM YOUR USE OF ANY PRODUCT INCLUDING ANY INDIRECT,

INCIDENTAL, SPECIAL, EXEMPLARY, MULTIPLE, PUNITIVE OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, WHETHER BASED ON CONTRACT, TORT (INCLUDING NEGLIGENCE), STRICT LIABILITY OR OTHER LEGAL THEORY, EVEN IF DIALOGIC, OR ANY OF ITS DIRECTORS, OFFICERS, EMPLOYEES, AGENTS OR AFFILIATES HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY EVENT, DIALOGIC'S CUMULATIVE LIABILITY TO YOU FOR ANY AND ALL CLAIMS RELATING TO THE USE OF ANY PRODUCT SHALL NOT EXCEED THE TOTAL AMOUNT OF THE PURCHASE PRICE OR LICENSE FEES PAID TO DIALOGIC FOR SUCH PRODUCT.

H. CUSTOMER AND DIALOGIC HEREBY WAIVE THEIR RIGHT TO TRIAL BY JURY TO THE FULLEST EXTENT PERMITTED BY LAW IN CONNECTION WITH ALL CLAIMS ARISING OUT OF OR RELATED TO THIS WARRANTY, THE PRODUCTS COVERED HEREBY OR THE PERFORMANCE OF ANY PARTY HEREUNDER.

I. THIS WARRANTY SHALL BE CONSTRUED UNDER AND GOVERNED BY THE LAWS OF THE COMMONWEALTH OF MASSACHUSETTS WITHOUT GIVING EFFECT TO ANY CHOICE OR CONFLICT OF LAW PROVISION OR RULE (WHETHER OF THE COMMONWEALTH OF MASSACHUSETTS OR ANY OTHER JURISDICTION) THAT WOULD CAUSE THE APPLICATION OF THE LAWS OF ANY JURISDICTION OTHER THAN THE COMMONWEALTH OF MASSACHUSETTS. CUSTOMER SPECIFICALLY AND IRREVOCABLY CONSENTS TO THE PERSONAL AND SUBJECT MATTER JURISDICTION AND VENUE OF THE FEDERAL AND STATE COURTS OF THE COMMONWEALTH OF MASSACHUSETTS AND SUCH COURTS SHALL HAVE EXCLUSIVE JURISDICTION WITH RESPECT TO ALL MATTERS CONCERNING THIS WARRANTY OR THE ENFORCEMENT OF ANY OF THE FOREGOING.

J. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM JURISDICTION TO JURISDICTION.

About this Publication

Purpose

This documentation provides guidelines for using the Dialogic® CSP.

Safety Labels

The following Safety labels may appear in this information product to alert customers to avoidable hazards. The following are in the order of priority:



DANGER

Danger indicates the presence of a hazard that will cause death or severe personal injury if the hazard is not avoided.



WARNING

Warning indicates the presence of a hazard that can cause death or severe personal injury if the hazard is not avoided.



CAUTION

Caution indicates the presence of a hazard that will or can cause minor personal injury or property damage if the hazard is not avoided. Caution can also indicate the possibility of data loss, loss of service, or that an application will fail.

Conventions used

This information product uses the text conventions explained below. In addition, hexadecimal numbers are preceded by a zero and small “x.” For example, the decimal number 15 is represented in hexadecimal as 0x0F.

Convention	Description
. . .	A horizontal ellipsis in an API message indicates fields of variable length.
:	A vertical ellipsis in an API message indicates that a block of information is repeated or is variable.
<i>n</i>	The letter <i>n</i> is a generic placeholder for a number.
Sans serif mono space	Indicates a command name, option, input, output, non-GUI error, and system messages.
<i>Sans serif monospace italic</i>	Indicates a parameter name in an input message. Example: move *.dot a: c: -s The -s is the parameter.
<i>Serif italic</i>	Indicates the name of a book, chapter, path, file, or API message. Example: <i>UserDirectory/Config.exe</i>
Boldface	Indicates keyboard keys, key combinations, and command buttons Example: Ctrl+Alt+Del
Sans serif boldface	Identifies text that is part of a graphical user interface (GUI). Example: Go to the Configuration menu and select Card->Span Configuration

Contents

Copyright and Legal Disclaimer	2
Dialogic Product Line Warranty	4

1 Introduction to SwitchKit® Development Environment TCAP Interface API

SwitchKit TCAP API Architecture	1-2
SKIM API 1-2	
SKTAL API 1-2	
SwitchKit API Library 1-2	
SK_DIALOGUE_MONITOR 1-2	
Key Features of SwitchKit TCAP APIs 1-3	
Diagram 1-3	
SKIM and SKTAL API	1-5
SKIM APIs 1-5	
SKTAL APIs 1-6	
Diagram 1-7	
SKIM and SKTAL API Call Flow	1-8
Diagram 1-8	
Description of Initialization 1-10	
Description of Sending Begin + Invoke 1-10	
Description of Receiving Begin + Invoke 1-10	
Description of Termination 1-10	
TCAP Concepts	1-12
Overview 1-12	
Component Sub-layer Services 1-12	
Transaction Sub-Layer Services 1-14	

2 SwitchKit Interface Module

SKIM API Messaging	2-2
Memory Management 2-2	
Allocation / Deallocation of Call Reference ID 2-2	

Handler Functions 2-3	
Handling Multiple	
Stack/SSN 2-3	
SKIM Error Handling 2-3	
Resource Limitation Error 2-4	
Code Syntax 2-4	
Initialize()	2-6
Description 2-6	
Syntax 2-6	
Parameters 2-6	
Return Values 2-7	
Terminate()	2-8
Description 2-8	
Syntax 2-8	
Parameters 2-8	
SSNInService()	2-9
Description 2-9	
Syntax 2-9	
Parameters 2-9	
SSNOutOfService()	2-10
Description 2-10	
Syntax 2-10	
Parameters 2-10	
Return Values 2-10	
Send()	2-11
Description 2-11	
Syntax 2-11	
Parameters 2-11	
Parameters 2-11	
Return Values 2-11	
SendReject()	2-12
Description 2-12	
Syntax 2-12	
Parameters 2-12	
Return Values 2-12	
CancelOperation()	2-13
Description 2-13	
Syntax 2-13	
Parameters 2-13	
Return Values 2-13	
SendPreArrangedEnd()	2-14
Description 2-14	
Syntax 2-14	

Parameters 2-14	
Return Values 2-14	
SetTransHandler()	2-15
Description 2-15	
Syntax 2-15	
Parameters 2-15	
Return Values 2-15	
ClearTransHandler()	2-16
Description 2-16	
Syntax 2-16	
Parameters 2-16	
Return Values 2-16	
EnableTracing()	2-17
Description 2-17	
Syntax 2-17	
Parameters 2-17	
Return Values 2-17	
DisableTracing()	2-18
Description 2-18	
Syntax 2-18	
Parameters 2-18	
Return Values 2-18	

3 SKIM Parameter Classes

SKIM_Trans Class	3-2
Summary of Methods 3-2	
SKIM_Trans Class Methods	3-4
SKIM_Trans() 3-4	
SetTransType() 3-4	
GetTransType() 3-5	
GetCallReferenceId() 3-6	
SetCallReferenceId() 3-6	
AddOperation() 3-6	
RemoveOperation() 3-7	
GetNumOfOperation() 3-7	
SetSourceAddress() 3-7	
GetSourceAddress() 3-8	
SetDestAddress() 3-8	
GetDestAddress() 3-8	
GetAbortReason() (For ITU) 3-9	
SetAbortReason (For ITU) 3-9	
GetAbortInfo() (For ANSI) 3-10	
SetAbortInfo (For ANSI) 3-10	

SetQualityOfService()	3-11
GetQualityOfService()	3-11
SetPreArrangedEnd()	3-12
GetReportCause()	3-12
GetApplicationContext() (ITU only)	3-13
SetApplicationContext() (ITU only)	3-14
SetUserInfo() (ITU only)	3-14
GetUserInfo() (ITU only)	3-14
SKIM_Operation Class	3-16
Description	3-16
Summary of Methods	3-16
Operation Types	3-17
SKIM_Operation Class Methods	3-19
SetType()	3-19
GetType()	3-19
SetInvokeId()	3-20
GetInvokeId()	3-20
SetLinkId()	3-21
HasLinkId()	3-21
GetLinkId()	3-21
SetClass()	3-22
GetClass()	3-22
SetInvokeTimeout()	3-23
SetParameter()	3-23
GetParameter()	3-23
IsLastOperation()	3-24
SetOpCode() (For ITU)	3-24
GetOpCode() (For ITU)	3-24
SetOpCode() (For ANSI)	3-25
GetOpCode() (For ANSI)	3-25
SetErrorCode() (For ITU)	3-26
GetErrorCode() (For ITU)	3-26
SetErrorCode() (For ANSI)	3-27
GetErrorCode() (For ANSI)	3-27
SetRejectCause()	3-28
GetRejectCause()	3-28
SKIM_Notify Class	3-29
Summary of Methods	3-29
3-29	
SKIM_Notify Class Methods	3-30
GetType()	3-30
Return Value:	3-30
GetCallReferenceId()	3-31
GetCause()	3-31

GetCauseType()	3-32
GetSSN()	3-32
GetPC()	3-32
GetInvokeId()	3-33
SKIM_CallingPartyAddress / SKIM_CalledPartyAddress Class	3-34
Description	3-34
Summary of Methods	3-34
SKIM_CallingPartyAddress/SKIM_CalledParty Address Class Methods ...	3-36
HasPointCode()	3-36
GetPointCode()	3-36
SetPointCode()	3-36
HasSSN ()	3-37
GetSSN ()	3-37
SetSSN ()	3-38
IsRoutedByPCSSN()	3-38
SetRouteByPCSSN()	3-38
IsInternationalRouting()	3-39
SetInternationalRouting ()	3-39
HasGlobalTitle()	3-39
SetGlobalTitle()	3-40
GetGlobalTitle	3-41

4 SKIM Sample Application

Environment Variables for Sample Applications	4-2
ITS_ROOT	4-2
INTL_ROOT	4-2
Step One: Creating a connection with LLC using SwitchKit	4-3
Application Code	4-3
Step Two: Creating an Instance of SKIM API Class	4-4
Application Code	4-4
Step Three: Sending a TCAP Transaction	4-5
Application Code	4-5
Step Four: Receiving a TCAP Transaction	4-7
Application Code	4-7
Step Five: Receiving Notifications	4-11
Application Code	4-11
Step Six: Terminating SKIM API Object Disconnection	4-13
Application Code	4-13

5 SwitchKit TCAP Abstraction Layer

Purpose	5-1
---------	-----

SwitchKit TCAP Abstraction Layer Messaging API.....	5-2
Summary of Methods 5-2	
sktal_initializeTCAP().....	5-3
Description 5-3	
Syntax 5-3	
Parameters 5-3	
Return Value 5-3	
sktal_terminateTCAP()	5-4
Description 5-4	
Syntax 5-4	
Parameters 5-4	
Return Value 5-4	
sktal_getTCAPHandle()	5-5
Description 5-5	
Syntax 5-5	
Input Parameters 5-5	
Parameters 5-5	
Return Value 5-5	
sktal_registerTCAPSSNHandler().....	5-6
Description 5-6	
Syntax 5-6	
Input Parameters 5-7	
Input/Output Parameters 5-7	
Output Parameters 5-7	
Return Value 5-7	
sktal_unregisterTCAPSSNHandler ()	5-8
Description 5-8	
Syntax 5-8	
Input Parameters 5-8	
Input/Output Parameters and Output Parameters 5-8	
Return Value 5-8	
sktal_setTCAPDialogueHandler ().....	5-9
Description 5-9	
Syntax 5-9	
Input Parameters 5-9	
Input/Output Parameters and Output Parameters 5-9	
Return Value 5-9	
sktal_clearTCAPDialogHandler().....	5-10
Description 5-10	
Syntax 5-10	
Input Parameters 5-10	
Input/Output Parameters and Output Parameters 5-10	
Return Value 5-10	

sktal_allocateTCAPDialogID()	5-11
Description 5-11	
Syntax 5-11	
Input Parameters 5-11	
Input/Output Parameters 5-11	
Output Parameters 5-11	
Return Value 5-11	
sktal_sendTCAPDialog()	5-12
Description 5-12	
Syntax 5-12	
Input Parameters 5-12	
Input/Output Parameters and Output Parameters 5-12	
Return Value 5-12	
sktal_recvTCAPDialog()	5-13
Description 5-13	
Syntax 5-13	
Input Parameters 5-13	
Input/Output Parameters 5-13	
Output Parameters 5-13	
Return Value 5-13	
sktal_sendTCAPComponent()	5-14
Description 5-14	
Syntax 5-14	
Input Parameters 5-14	
Input/Output Parameters and Output Parameters 5-14	
Return Value 5-14	
sktal_recvTCAPComponent()	5-15
Description 5-15	
Syntax 5-15	
Input Parameters 5-15	
Input/Output Parameters 5-15	
Output Parameters 5-15	
Return Value 5-15	

6 SKTAL Parameter Classes

Class TCAP_Component	6-3
Purpose 6-3	
Summary of Methods 6-3	
6-3	
GetComponentType() 6-3	
GetLast() 6-4	
SetInvokeID() 6-4	
GetInvokeID() 6-5	

Send()	6-5
Receive()	6-6
Print()	6-7
Class TCAP_Invoke : public TCAP_Component	6-8
Purpose	6-8
Summary of Methods	6-8
SetOperation()	6-8
GetOperation()	6-9
SetParameter()	6-10
GetParameter()	6-11
SetTimeOut()	6-11
GetTimeOut()	6-12
SetClass()	6-12
GetClass()	6-13
SetLinkedID()	6-13
HasLinkedID()	6-14
GetLinkedID()	6-14
LinkInvoke()	6-14
Class TCAP_Result : public TCAP_Component	6-16
Purpose	6-16
Summary of Methods	6-16
SetOperation()	6-16
GetOperation()	6-16
SetParameter()	6-17
GetParameter()	6-17
Class TCAP_Error : public TCAP_Component	6-19
Purpose	6-19
Summary of Methods	6-19
SetError()	6-19
GetError()	6-20
SetParameter()	6-20
GetParameter()	6-20
Class TCAP_Reject : public TCAP_Component	6-22
Purpose	6-22
Summary of Methods	6-22
SetProblem()	6-22
GetProblem()	6-23
SetParameter()	6-24
GetParameter()	6-24
Class TCAP_Cancel : public TCAP_Component	6-25
Purpose	6-25
GetDialogueID()	6-25
Class TCAP_ResetTimer : public TCAP_Component	6-26

GetDialogueID() 6-26	
Class TCAP_Dialogue	6-27
Purpose 6-27	
Summary of Methods 6-27	
SetDialogueID() 6-28	
GetDialogueType() 6-28	
GetDialogueID() 6-29	
IsComponentPresent() 6-29	
SetQualityOfService() 6-30	
GetQualityOfService() 6-30	
SetApplicationContext() 6-31	
GetApplicationContext() 6-31	
SetUserInfo() 6-33	
GetUserInfo() 6-33	
SendCheck() 6-34	
ReceiveCheck() 6-35	
Send() 6-35	
Receive() 6-36	
GetHeader() 6-36	
Print() 6-37	
TCAP_Unidirectional : public TCAP_Dialogue class	6-38
Purpose 6-38	
Summary of Methods 6-38	
SendCheck() 6-38	
ReceiveCheck() 6-38	
SetOrigAddr() 6-39	
GetOrigAddr() 6-40	
SetDestAddr() 6-40	
GetDestAddr() 6-41	
Class TCAP_Begin : public TCAP_Dialogue	6-43
Purpose 6-43	
Summary of Methods 6-43	
SendCheck() 6-44	
ReceiveCheck() 6-44	
GetOPC() 6-44	
SetDPC() 6-45	
SetOrigAddr() 6-45	
GetOrigAddr() 6-45	
SetDestAddr() 6-46	
GetDestAddr() 6-46	
Class TCAP_Continue : public TCAP_Dialogue	6-47
Purpose 6-47	
Summary of Methods 6-47	
SendCheck() 6-47	

ReceiveCheck()	6-47
SetOrigAddr()	6-48
GetOrigAddr()	6-48
Class TCAP_End: public TCAP_Dialogue	6-49
Purpose	6-49
Summary of Methods	6-49
IsPreArrangedEnd()	6-49
SetPreArrangedEnd()	6-49
Class TCAP_Abort : public TCAP_Dialogue	6-51
Purpose	6-51
Summary of Methods	6-51
IsComponentPresent()	6-51
SetComponentPresent()	6-51
SetAbortReason()	6-52
GetAbortReason()	6-52
GetAbortInfo()	6-52
SetAbortInfo()	6-53
Class TCAP_Notice : public TCAP_Dialogue	6-54
Purpose	6-54
Summary of Methods	6-54
IsComponentPresent()	6-54
SetComponentPresent()	6-54
SetReportCause()	6-55
GetReportCause()	6-55
SetQualityOfService()	6-56
GetQualityOfService()	6-56
SetApplicationContext()	6-56
GetApplicationContext()	6-56
SetUserInfo()	6-57
GetUserInfo()	6-57

7 SKTAL Sample Application

7-1	
Step One - Creating a connection with LLC using SwitchKit APIs	7-2
Application Code	7-2
Step Two - Initializing SKTAL and registering SSN	7-3
Application Code	7-3
Step Three - Sending a TCAP Dialogue and Component	7-4
Application Code	7-4
Step Four - Receiving a TCAP Transaction	7-6
Application Code	7-6
Step Five - Terminating SKTAL and Disconnection	7-8

A Appendix - TCAP Codes

 A-1
ANSI TCAP CodesA-2
ITU TCAP CodesA-6
 Purpose A-6

B Appendix - SKIM Data Types and Codes

 B-1
SKIM Data Types and CodesB-2

C Appendix - SKTAL Data Types and Codes

 C-1
SKTAL Data TypesC-2

D Appendix - SKTAL Dialog and Component Structure Definitions

 D-1
SKTAL_DLГ and SKTAL_CPT Structure Definitions.....D-2

E Appendix - SKIM and SKTAL API Integration with IntelliNet Codecs

 E-1
Integrating SKIM and SKTAL API with IntelliNet CodecsE-2

1 Introduction to SwitchKit® Development Environment TCAP Interface API

Purpose The SwitchKit® Development Environment TCAP APIs provide a C++ interface to the TCAP application. The SwitchKit Interface Module and SwitchKit TCAP Abstraction Layer API hide the details of the PPL TCAP Interface messages from the SwitchKit TCAP application.

SwitchKit TCAP API Architecture

Overview The SwitchKit TCAP application resides on the host and communicates over TCP/IP with the CSP. SwitchKit TCAP application components include the following:

- SwitchKit Interface Module (SKIM) TCAP API
- SwitchKit TCAP Abstraction Layer (SKTAL)
- SwitchKit API library

SKIM API Application developers can use the SwitchKit Interface Module (SKIM) to speed up development. SKIM provides an API to abstract services from the TCAP and SCCP layers to a transaction-based application. The SKIM API abstracts details of the SS7 TCAP protocol. The SKIM provides some error handling, abstracting TCAP from the error handling. The SKIM must keep track of active dialogues, invoke IDs, operation types, and operation codes.

SKTAL API The SwitchKit TCAP Abstraction Layer (is used to create a generic view of the TCAP layer, so that the underlying stack can be used with the TCAP C++ API. The SKTAL API allows the SwitchKit TCAP user to easily interface with Intelligent Network (IN) application layer protocol APIs (GSM-MAP, ANSI-41, WIN, and CAMEL) provided by IntelliNet Technologies Inc.

SwitchKit API Library The Switchkit API is a library of function calls used to communicate to both the LLC and the CSP. In *Figure 1-1, SKIM and SKTAL Architecture (1-3)* both the TCAP Abstraction Layer and the application must communicate with the SwitchKit API. Although the CSP can be used as a signaling-only platform, its strength is in using it for IN and ISUP signaling and for media resources. An application must communicate with the SwitchKit API in order to handle ISUP traffic and to control DSP resources in the CSP.

Important! The SwitchKit environment variable, SK_DIALOGUE_MONITOR, is mandatory for TCAP applications.

SK_DIALOGUE_MONITOR SK_DIALOGUE_MONITOR is used to activate the LLC's dialogue monitor.

Defaults file variable name:

dialogue_monitor

Valid Values

SK_DIALOGUE_MONITOR=0

- -disables the LLC's dialogue monitor (default)

SK_DIALOGUE_MONITOR=1

- - enables the LLC's dialogue monitor

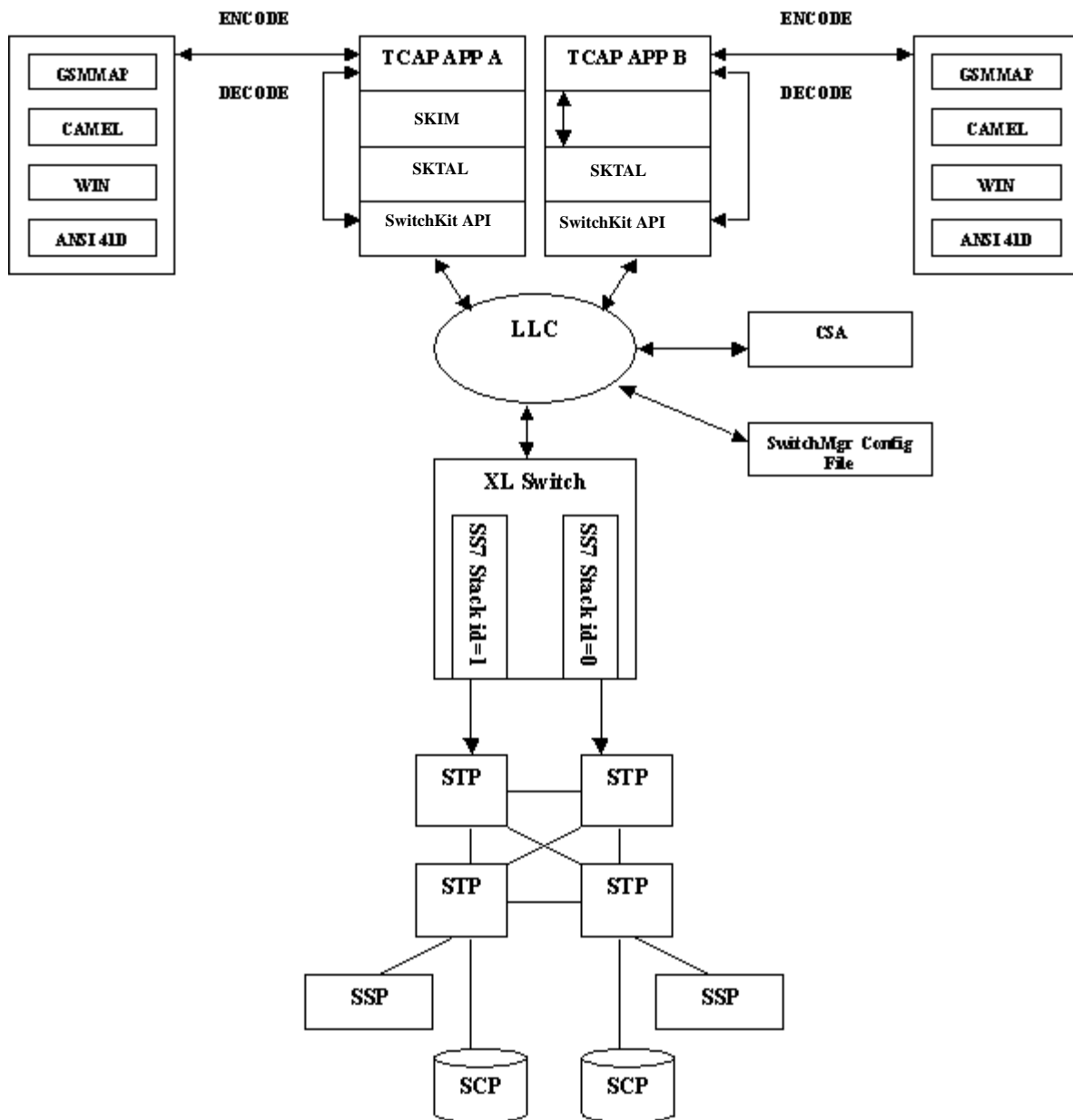
See LLC: Application Load Balancing for TCAP (7-18) for more information on how to use the LLC's dialogue monitoring capabilities.

**Key Features of SwitchKit
TCAP APIs**

SwitchKit TCAP API provides a simple C++ interface to EXS TCAP. SwitchKit TCAP APIs hide details of PPL messages from the user application.

- SwitchKit TCAP API allows the user application to communicate with multiple stack instances.
- SwitchKit TCAP API provides message sanity checking and error handling at the host.
- SwitchKit TCAP API enables user applications to easily integrate with IntelliNet's user part APIs such as GSM-MAP, CAMEL, WIN, ANSI-41, AIN, and INAP.
- The SwitchKit TCAP API provides interface functions that take ASN.1 encoded information from IntelliNet user part APIs for populating TCAP operation parameters. TCAP operation parameters received using SwitchKit TCAP API can be decoded using user part APIs.

Diagram Figure 1-1 SKIM and SKTAL Architecture



SKIM and SKTAL API

Overview The SwitchKit SKIM and SKTAL APIs interface with the SwitchKit library. The SKIM APIs utilize services of the SKTAL layer to communication with the LLC and the CSP.

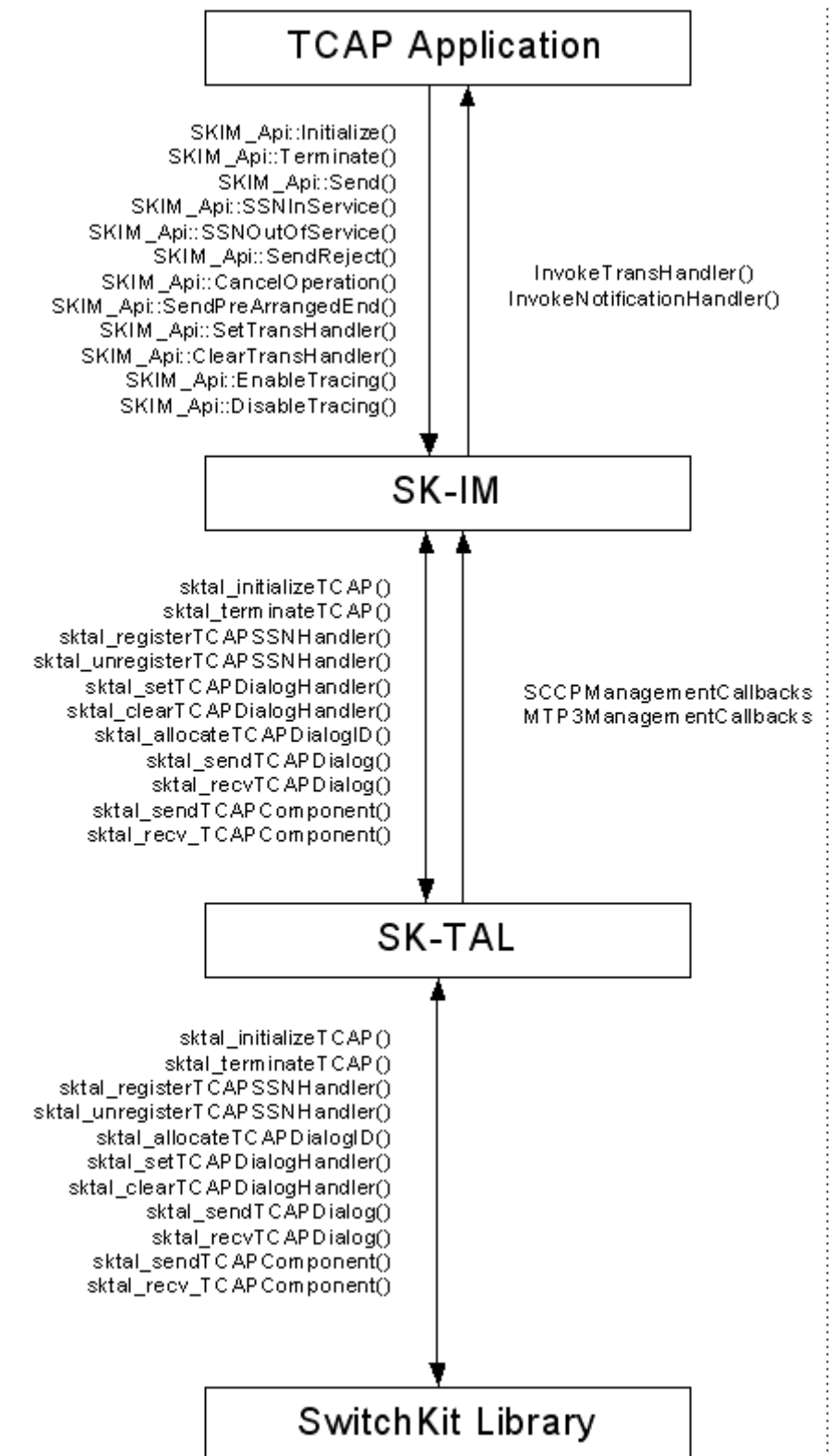
SKIM APIs The next table describes the SKIM API classes available to applications.

Name	Description
SKIM_Api::Initialize()	Initializes a SKIM API object. This method also initializes SKTAL and registers a TCAP SSN with the CSP.
SKIM_Api::Terminate()	Terminates the SKIM API object and performs cleanup functions. This method terminates SKTAL and deregisters a TCAP SSN with the CSP. All pending dialogs associated with SKIM_Api object are cleared by sending a pre-arranged end message to the CSP.
SKIM_Api::Send	Sends TCAP transaction message along with a list of operations. A given transaction message can contain zero or more operations (TCAP Components).
SKIM_Api::SSNInService	Sends N-State request to bring the local subsystem network (SSN) in service.
SKIM_Api::SSNOutOfService	Sends N-State request to take the local SSN out of service.
SKIM_Api::SendReject	Allows the user to send TCAP Reject message for a given invoke ID and call reference ID. This method also terminates the transaction by sending TCAP End.
SKIM_Api::CancelOperation	Allows the user to cancel a previously invoked operation for a given call reference ID and invoke ID.
SKIM_Api::SendPreArrangedEnd	Sends a pre-arranged end message for a given call reference ID.
SKIM_Api::SetTransHandler	Sets a transaction handler for a given call reference ID.
SKIM_Api::ClearTransHandler	Clears a transaction handler for a given call reference ID.
SKIM_Api::EnableTracing	Enables SKIM library tracing for a given trace type.
SKIM_Api::DisableTracing	Disables SKIM library tracing for a given trace type.

SKTAL APIs The following are some of the SKTAL API messages available to the application:

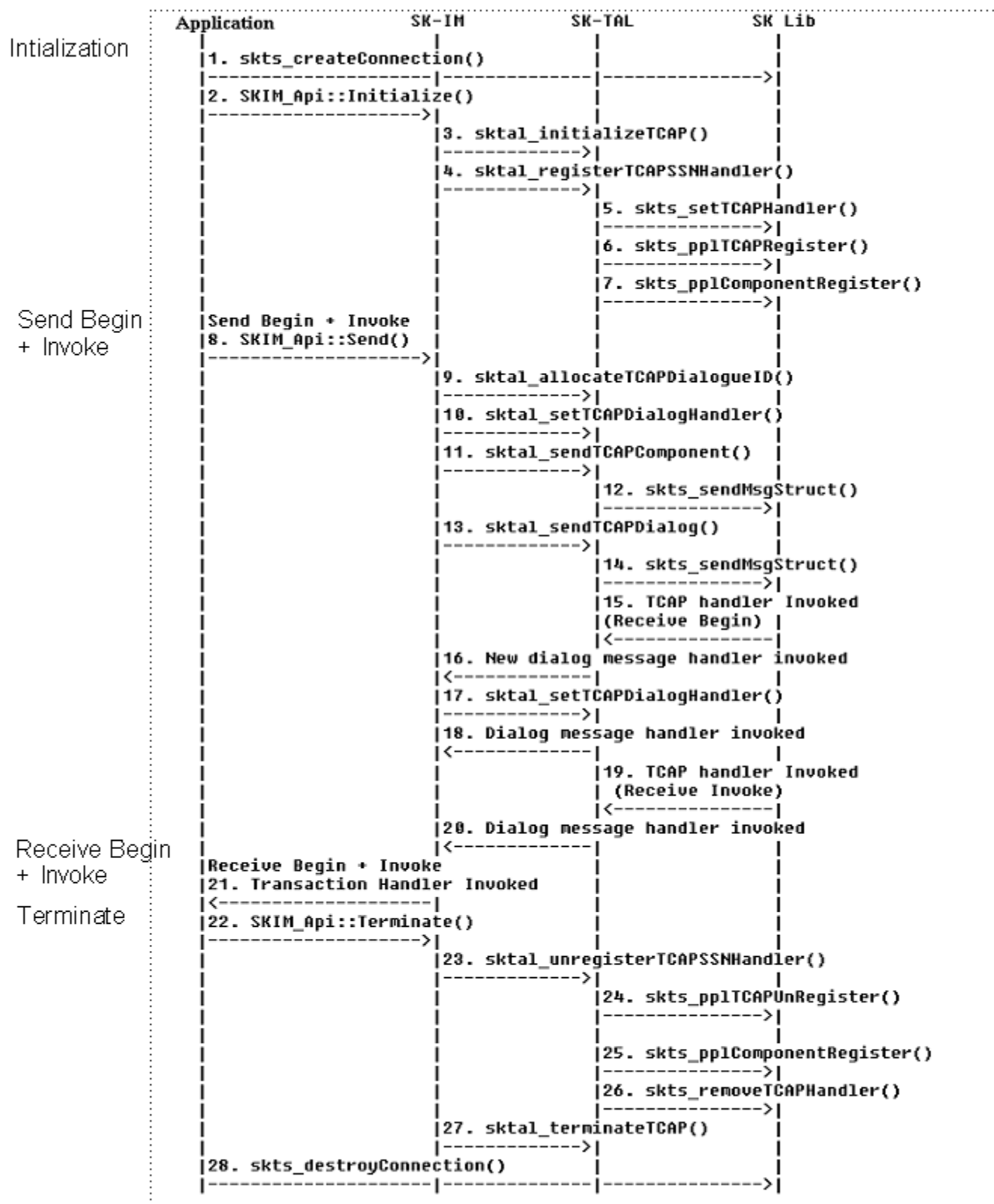
Name	Description
sktal_initializeTCAP	A one-time call for global initialization of the TCAP Abstraction layer.
sktal_terminateTCAP	A one-time call for global termination of the TCAP Abstraction layer.
sktal_registerTCAPSSNHandler	Registers the application with a given node/stack/subsystem. A generic handler will be added to the handler stack that will filter messages for the given subsystem. When a new TCAP transaction is seen, or when sk_TCAP_AllocateDialogueID is called, the NewDialogueHandler function argument will be invoked. This function is expected to register a per dialogue handler (via sktal_setTCAPSSNHandler) for the new dialogue. All incoming messages within that dialogue sequence will be passed to that handler.
sktal_unregisterTCAPSSNHandler	Removes (and terminates) the subsystem identified during registration. All ongoing transactions are terminated and the subsystem is marked out-of-service.
sktal_allocateTCAPDialogID	Allocates the TCAP dialogue ID.
sktal_setTCAPDialogHandler	Specifies a function to be invoked for incoming messages within a given transaction (identified by the Dialogue ID). The incoming event and the Dialogue ID invoke the handler; the handler should examine the event and invoke either sktal_recvTCAPDialogue or sktal_recvTCAPComponent.
sktal_clearTCAPDialogHandler	The user removes the transaction handler once the transaction has been completed (by receiving) TC-U-ABORT, TC-P-ABORT, TC-END, or TC-UNI, or sending TC-U-ABORT, TC-END, or TC-UNI). It is imperative to not take this step until the last message has been sent.
sktal_sendTCAPDialog	Sends a TCAP dialogue message.
sktal_recvTCAPDialog	Receives a TCAP dialogue info from a SKTAL_EVENT.
sktal_sendTCAPComponent	Sends a TCAP component message.
sktal_recv_TCAPComponent	Receives TCAP component information from a SKTAL_EVENT.

Diagram The next diagram shows how SKIM and SKTAL APIs interface with the SwitchKit library.



SKIM and SKTAL API Call Flow

- Overview** This section provides an example call flow with a description of the steps in the call flow.
- Diagram** The next diagram shows messaging of a call from initialization to termination. See the description of the steps in the next diagram.



- | | |
|--|--|
| Description of Initialization | <ol style="list-style-type: none">1. Create a connection with LLC using SwitchKit API <code>sktsCreateConnection ()</code>.2. Initialize the SKIM_Api object.3. SKIM initializes the SKTAL API layer.4. SKIM registers a TCAP SSN with SKTAL layer.5. SKTAL registers a TCAP message handler with SwitchKit.6. SKTAL registers a TCAP SSN with SwitchKit. This is done for receiving messages for a given TCAP SSN.7. SKTAL registers SCCP User Interface component with SwitchKit. This is done to receive SCCP management messages. |
| Description of Sending
Begin + Invoke | <ol style="list-style-type: none">8. SKIM application sends a TCAP transaction with a Begin dialogue and Invoke Component.9. SKIM allocates a TCAP dialogue ID for the new transaction.10. SKIM registers a dialog handler for receiving incoming messages for the new TCAP dialogue.11. SKIM sends the TCAP invoke component to the SKTAL layer.12. SKTAL sends the TCAP invoke component to the EXS using SwitchKit API.13. SKIM sends the TCAP Begin dialogue to the SKTAL layer.14. SKTAL sends the TCAP Begin dialogue to the EXS using SwitchKit API. |
| Description of Receiving
Begin + Invoke | <ol style="list-style-type: none">15. SwitchKit library invokes SKTAL TCAP message handler to give TCAP Begin message to SKTAL.16. SKTAL receives TCAP Begin dialogue message and invokes SKIM new dialogue handler.17. SKIM registers a per dialogue message handler as part of new dialog handler.18. SKTAL invokes the SKIM dialogue message handler to pass TCAP Begin dialog to SKIM.19. SwitchKit library invokes SKTAL TCAP message handler to give TCAP Invoke message to SKTAL.20. SKTAL invokes the SKIM dialogue message handler to pass TCAP invoke component to SKIM.21. SKIM invokes application transaction handler to pass TCAP Begin and Invoke to the application. |
| Description of Termination | <ol style="list-style-type: none">22. Application calls SKIM Terminate API to terminate the SKIM API object.23. SKIM deregisters TCAP SSN with SKTAL. |

24. SKTAL deregisters TCAP SNN with SwitchKit.
25. SKTAL deregisters SCCP User Interface component with SwitchKit.
26. SKTAL removes TCAP message handler
27. SKIM terminates the SKTAL API layer.
28. Application destroys the connection with LLC.

TCAP Concepts

Overview TCAP protocol functionality is divided into two main functional blocks: Component Sub-layer and Transaction Sub-layer. Services provided by these two functional blocks are described in the following sections.

Component Sub-layer Services

A component is the means by which TCAP conveys a request to perform an operation or reply. An operation is an action performed by the remote end and may have associated parameters. An Invoke ID identifies the invocation of an operation, allowing several invocations of the same or different operations to be active simultaneously. Only one reply may be sent to an operation, indicating success or failure of the operation.

Components are passed individually between a TC-user (or application) and the component sub-layer. The originating TC-user may send several components to the component sub-layer before the message is transmitted to the destination TC-user. When several components are received in a single message, each one is delivered individually to the destination TC-user. Components in a message are delivered to the remote TC-user in the same order they are provided at the originating interface. The importance of the order is established by prior agreement between the TC-users.

Component Correlation

The component sub-layer provides two facilities:

1. Association of operations and replies
2. Abnormal situations handling

Association of Operations and Replies

The value of the Invoke ID, which identifies an operation invocation without ambiguity, is returned in a response to that operation invocation. A TC-user may have a number of operations active simultaneously; the maximum number depends on the unique Invoke IDs available to a TC-user. If the response to this operation invocation is another operation invocation from the responding end, the original Invoke ID is returned as a Linked ID indicating that this responding operation invocation is linked to the original operation. Four classes of operations are considered:

- Class 1 Both success and failure are reported
- Class 2 Only failure is reported
- Class 3 Only success is reported
- Class 4 Neither success, nor failure is reported

Where necessary, the TC-user provides segmentation of a successful response. In addition, any number of linked operations may be sent prior to the reply. The reply may be:

- Return result indicating success
- Return error indicating operation failure
- Reject indicating inability to perform the operation

The application protocol designer decides what constitutes success or failure in performing the operation. Any component, except a reject component, may be rejected. Rejection of a result causes termination of the corresponding operation; rejection of a linked operation does not affect the linked-to operation. Rejection of a result segment (that is, rejection of a Return Result — Not Last component) implies the rejection of all subsequent segments and the entire result. A TC-user may cancel an invoked operation; any reply received for this invocation will be rejected.

Abnormal Situations Handling

The Component sub-layer covers a number of abnormal situations in relation to a component:

- Component reject: When the component sub-layer receives a malformed component, or a component that violates the rules of exchange, it informs the TC-user(s)
- Operation expiration: When the component sub-layer detects that a Class 1, 2, or 3 operation has not received a final reply after a certain amount of time (which is specified by the application and depends on the operation), it releases the corresponding Invoke ID and informs the TC-user. Note that this situation is abnormal only in the case of a Class 1 operation. Application of this error to Class 4 operations is a local matter.
- Invocation cancel: A TC-user may decide to release a given Invoke ID and ignore any responses using this identifier

Error Handling

In a situation where the component sub-layer is prevented from providing the expected services, it notifies the TC-user and may terminate pending operations. In addition, a TC-user may decide to abort a dialogue, which ends any pending operations.

Transaction Sub-Layer Services

The Transaction (TR) sub-layer provides capabilities for the exchange of components and dialogue between TR-users. The TR sub-layer also provides the capability to send transaction messages between peer TR sub-layer entities by means of the lower layer network services.

Dialogue

When successive components are exchanged between two TC-users in order to perform an application, this is called a dialogue. The component sub-layer provides dialogue facilities, allowing several dialogues to run concurrently between two TC-users. In addition, dialogue handling allows for the transfer and negotiation of the application context and the transparent transfer of user information (that is, data that are not components) between two TC-users. The component sub-layer provides two dialogue facilities: Unstructured and structured.

Unstructured dialogue infers that TC-users can send components that do not require replies without forming an explicit association. Implicit association always exists between communicating TC-users. An unstructured dialogue is supported by the uni-directional message within the protocol. Use of the unstructured dialogue facility is indicated when one TC-user sends a uni-directional message to its peers. When a TC-user receives a uni-directional message, if a protocol error is reported, it is returned in a uni-directional message.

Structured dialogue infers that TC-users indicate the beginning or the formation of a dialogue, the continuation, and the end of a dialogue. Structured dialogue allows two TC-users to run several dialogues concurrently, each identified by a particular Dialogue ID. Each Dialogue ID has a separate Invoke ID name space, allowing duplication of Invoke IDs in different dialogues. Sequenced delivery of messages is provided by means of application protocols outside the scope of TCAP, or by use of the appropriate SCCP class of service. When using the structured dialogue service, the TC-user indicates one of four possibilities upon sending a component to a peer TC-user:

1. Dialogue begin
2. Dialogue confirmation
3. Dialogue continue - Full-duplex exchange of components is possible
4. Dialogue end - The sending TC-user will not send more components, nor will it accept any more components from the remote TC-user

As an option, in the context of options 1 and 2, an exchange of application context information and user information is possible. When this option is chosen, user information can also be sent during options 3 and 4.

2 SwitchKit Interface Module

Purpose This chapter contains information on the SwitchKit TCAP Interface API. The SwitchKit TCAP Interface provides a C++ API to a TCAP application. SKIM/SKTAL TCAP API hides details of the PPL TCAP Interface messages from the SwitchKit TCAP application.

SKIM API Messaging

Overview The SKIM API layer provides the following functionality:

- Allocation / Deallocation of Call Reference ID
- Handler Functions
- Handling Multiple Stack/SSN
- SKIM Error Handling
- Resource Limitation Error

This functionality is described further in this section.

SKIM data types and return codes are defined in Appendix B Each SKIM transaction is uniquely identified by a call reference ID. The Call Reference ID is allocated by SKIM when a new transaction is initiated or received. The call reference ID is released when a transaction is ended or aborted by the SKIM application.

Memory Management

A SKIM application does not need to do memory management with respect to SKIM API usage. A SKIM application must manage the ongoing transactions and should clear the transaction by sending or receiving prearranged end, basic end, or abort.

SKIM API allocates a transaction context for each active transaction. The transaction context is cleared when transaction is ended / aborted. In error scenarios, sending a prearranged end will clear the transaction.

Allocation / Deallocation of Call Reference ID

Each ongoing TCAP transaction is identified by a call reference ID. The SKIM API layer allocates a call reference ID when a user initiates a transaction or a new transaction is received from the remote end. After initiating a transaction the SKIM API user can retrieve the call reference ID assigned for the new transaction by calling GetCallReferenceId() method of a transaction object.

Call Reference ID and associated resources are deallocated when a transaction is ended/aborted by the SKIM user or remote SS7 entity. The SKIM user can free the call reference ID by calling

SendPreArrangedEnd() for a given call reference ID value. TCAP prearranged end terminate transaction locally without sending any TCAP message to remote SS7 entity.

Handler Functions

SKIM API maintains a transaction handler to pass incoming transaction messages to the SKIM user. SKIM API maintains notification handlers to pass incoming notification messages to the SKIM user. SKIM notifications consist of SCCP N-STATE/N-PCSTATE indications, TCAP timeouts, SKIM internal error, and NACK messages from the CSP. The SKIM user must register a transaction handler function and notification handler function as part of SKIM_Api object initialization. The SKIM user can override the default transaction handler function for a given call reference ID.

Most set or get methods return void. If a set or get method is called on in an invalid transaction type, then the method returns void. The set or get methods will have no affect on invalid transaction or operation types.

Handling Multiple Stack/SSN

The SKIM API user can communicate with multiple stack/SSN combinations configured on the CSP. For each stack/SSN combination the user must create and initialize a SKIM_Api object. Communication to a stack/SSN is controlled using the public interface provided by SKIM_Api object.

SKIM Error Handling

The SKIM API layer maintains a transaction context for each active transaction and provides error handling over and above the TCAP protocol stack. These error conditions are reported to the SKIM user as a notification of type SKIM_NOTIFY_INTERNAL_ERROR. The SKIM user can get the cause of internal error notification by calling the GetCause public method of SKIM_NOTIFY object. Some of the error conditions handled by SKIM API layer are described in the next table:

Error	Description
Transaction Record Error	This error message is returned SKIM cannot retrieve a transaction context for an incoming or outgoing transaction message. This occurs when the user tries to send a TCAP Continue or TCAP End message for a non-existent transaction.
Duplicate Invoke ID Error	This error occurs when a TCAP invoke operation with a invoke ID identical to an ongoing TCAP operation is sent or received for a given call reference ID.

Error	Description
Transaction Record Error	This error message is returned SKIM cannot retrieve a transaction context for an incoming or outgoing transaction message. This occurs when the user tries to send a TCAP Continue or TCAP End message for a non-existent transaction.
Unknown Invoke ID Error	This error occurs when a response to a invoke operation contains an unknown invoke ID.
Invalid Linked ID	This error occurs when a TCAP invoke operation containing in valid linked ID (linked ID for which corresponding invoke operation does not exist) is received or sent.
Unknown Message Type Error	This error occurs when an unknown transaction message type is received.
Unknown Component Error	This error occurs when a TCAP operation with unknown type is received.
Resource Limitation Error	This error is returned when user tries to initiate a transaction but maximum simultaneous transaction limit is reached.

Resource Limitation Error This error is returned when a user tries to initiate a transaction but the maximum simultaneous transaction limit has already been reached.

Code Syntax Each SKIM transaction is uniquely identified by a call reference ID. The Call reference ID is allocated by SKIM when a new transaction is initiated or received. The Call reference ID is released when a transaction is ended or aborted by the SKIM application. SKIM data types and return codes are defined in Appendix 2.

The SwitchKit Interface Module provides the following methods to the application developer.

```
class SKIM_Api
{
public:

    int Initialize(int localNodeId, int stackId, int
connId, int pc, int ssn, SKIM_TransHandler h1,
SKIM_NotificationHandler h2, void *tag);

    int Terminate();
    int SSNInService();
    int SSNOutOfService();
    int Send(SKIM_Trans &trans);
    int SendReject(int callrefid, SKIM_OCTET invokeId,
SKIM_OCTET probType, SKIM_OCTET probCode);
```

```
int CancelOperation(int callrefid, SKIM_OCTET invokeId);
    int SendPreArrangedEnd(int calrefid);
int SetTransHandler(int callrefid, SKIM_TransHandler h,
    void *tag);
int ClearTransHandler(int callrefid);
static int EnableTracing(int traceType);
static int DisableTracing(int traceType);
};
```

Initialize()

Description SKIM application calls the initialize method to initialize the SKIM API object. This method initializes SKTAL and registers a TCAP subsystem (SSN) with the CSP.

When using multiple running applications on the same subsystem number, the startRange and endRange arguments should be modified to divide the dialogue IDs between them. For example, when using two applications the values are:

Application 1	Application 2
startRange = 0	startRange = 4096
endRange = 4095	endRange = 8191

Syntax `int Initialize(int localNodeId, int stackId, int connId, int pc, int ssn, SKIM_TransHandler h1, SKIM_NotificationHandler h2, void *tag, int startRange=0, int endrange=SK_MAX_TCAP_DIALOGS-1)`

Parameters The input parameters are shown in the next table. There are no input/output parameters.

Argument	Description
localNodeId	Local node ID
stackId	stack ID
connId	connection ID
pc	point code
ssn	subsystem number (SSN)
h1	Default transaction handler callback. The transaction handler is invoked by SKIM after receiving a TCAP transaction message. The prototype for SKIM transactionhandler is: <pre>typedef int (*SKIM_TransHandler)(SKIM_Trans &trans, void *tag);</pre>

Argument	Description
h2	Notification handler callback. The notification handler is invoked when notification message is received by SKIM. The prototype for notification handler is: <pre>typedef int (*SKIM_NotificationHandler)(SKIM_Notify &notify, void *tag);</pre>
tag value	Tag to be returned with the callbacks
startRange	Starting dialogue ID allocated to this application (For the specific SSN)
endRange	Ending dialogue ID allocated to this application (For the specific SSN)

Return Values Possible return values for this API are:

SKIM_SUCCESS No failure has occurred.

SKIM return code As defined in Appendix B, *Table B-2, SKIM Return Codes (B-2)*.

Terminate()

Description This API terminates the SKIM API object. This method terminates SKTAL and deregisters a TCAP subsystem (SSN) with the CSP.

Syntax `int Terminate();`

Parameters There are no input, input/output or output parameters.

Return Values The possible return value for this API is:

SKIM_SUCCESS Always returns SKIM_SUCCESS.

SSNInService()

Description The SSNInService method sends N-State request to bring the local subsystem (SSN) in service.

Syntax `int SSNInService();`

Parameters There are no input, input/output or output parameters.

Return Values Possible return values for this API are:

SKIM_SUCCESS	No failure has occurred.
SKIM return code	As defined in Appendix B, <i>Table B-2, SKIM</i>

SSNOutOfService()

Description The SSNOutOfService method sends N-State request to take the local subsystem (SSN) out-of-service.

Syntax `int SSNOutOfService();`

Parameters There are no input, input/output or output parameters.

Return Values Possible return values for this API are:

SKIM_SUCCESS	No failure has occurred.
SKIM return code	As defined in Appendix B, <i>Table B-2, SKIM</i>

Send()

Description This method sends TCAP transaction message along with a list of operations. A given transaction message can contain zero or more operations.

Syntax `int Send(SKIM_Trans &trans);`

Parameters

Parameters The input parameter is shown in the next table. There are no input/output parameters.

Argument	Description
SKIM_Trans	Transaction object For details of SKIM_Trans class interface refer to <i>Chapter 3, SKIM Parameter Classes</i> .

Return Values Possible return values for this API are:

SKIM_SUCCESS No failure has occurred.

SKIM return code As defined in Appendix B, *Table B-2, SKIM Return Codes (B-2)*.

SendReject()

Description This method allows the user to send Reject for a given invoke ID and call reference ID. This does not clear the call reference ID therefore, you must send a prearranged end or basic end.

Syntax

```
int SendReject(int callrefid, SKIM_OCTET invokeId,  
              SKIM_OCTET probType,  
              SKIM_OCTET probCode);
```

Parameters The input parameter is shown in the next table. There are no input/output parameters.

Argument	Description
callrefid	Call reference ID to identify TCAP transaction for which Reject is being sent.
invokeId	Invoke ID identifies TCAP operation being rejected.
probType	Problem type as specified in Appendix A.
probCode	Reject problem code. As specified in Appendix A.

Return Values Possible return values for this API are:

SKIM_SUCCESS Method is successful. No failure has occurred.

SKIM return code As defined in Appendix B *Table B-2, SKIM Return Codes (B-2)*.

CancelOperation()

Description This method allows the user to cancel a previously invoked operation for a given call reference ID and invoke ID.

Syntax `int CancelOperation(int callrefid, SKIM_OCTET invokeId);`

Parameters The input parameter is shown in the next table. There are no input/output parameters.

Argument	Description
callrefid	Call reference ID
invokeId	Invoke ID

Return Values Possible return values for this API are:

SKIM_SUCCESS No failure has occurred.

SKIM return code As defined in Appendix B *Table B-2, SKIM Return Codes (B-2)*.

SendPreArrangedEnd()

Description This method sends a Pre-Arranged End message to the CSP for the given call reference ID. This method releases the call reference ID and associated resources.

Syntax `int SendPreArrangedEnd(int callrefid);`

Parameters The input parameter is shown in the next table. There are no input/output parameters, or output parameters.

Argument	Description
callrefid	Call reference ID

Return Values Possible return values for this API are:

SKIM_SUCCESS Method is successful.

SKIM return code As defined in Appendix B *Table B-2, SKIM Return Codes (B-2)*.

SetTransHandler()

Description This method sets a transaction handler for a given call reference ID.

Syntax `int SetTransHandler(int callrefid, SKIM_TransHandler h,
void *tag);`

Parameters The input parameters are shown in the next table. There are no input/output parameters.

Argument	Description
callrefid	Call reference ID for which transaction handler is being registered.
h	Transaction handler. Prototype for transaction handler function is: <code>typedef int (*SKIM_TransHandler)(SKIM_Trans &trans, void *tag);</code>
tag	This tag is returned when transaction handler is invoked.

Return Values Possible return values for this API are:

`SKIM_SUCCESS` No failure has occurred.

`SKIM_TRANS_RE` No transaction is active for given call reference
`CORD_ERROR` ID.

ClearTransHandler()

Description This method clears a transaction handler for a given call reference ID.

Syntax `int ClearTransHandler(int callrefid);`

Parameters The input parameters are shown in the next table. There are no input/output parameters.

Argument	Description
callrefid	Call reference ID for which the transaction handler is being cleared.

Return Values Possible return values for this API are:

SKIM_SUCCESS No failure has occurred.

SKIM_TRANS_RE
CORD_ERROR No transaction is active for given call reference ID.

EnableTracing()

Description This method enables SKIM library tracing for a given trace type. Execution traces are generated to a file and standard output. Trace files are created in the current execution directory. If a given trace type is already enabled then this method will not have any effect.

Syntax `static void EnableTracing(int traceType);`

Parameters The input parameters are shown in the next table. There are no input/output parameters.

Argument	Description
traceType	Possible values: SKIM_SUCCESS_TRACE SKIM_ERROR_TRACE

Return Values None.

DisableTracing()

Description This method disables SKIM library tracing for a given trace type. If a given trace type is already disabled then this method will have no effect.

Syntax `static void DisableTracing(int traceType);`

Parameters The input parameters are shown in the next table. There are no input/output parameters.

Argument	Description
TraceType	Possible values: SKIM_SUCCESS_TRACE SKIM_ERROR_TRACE

Return Values None.

3 SKIM Parameter Classes

Purpose This chapter describes the parameter classes for the SwitchKit Interface Module, which include:

- SKIM_Trans Class
- SKIM_Operation Class
- SKIM_Notify Class
- SKIM_CallingPartyAddress / SKIM_CalledPartyAddress Class

SKIM_Trans Class

Purpose The SKIM_Trans class provides an interface to the SKIM user for populating and retrieving TCAP dialogue and components (operations). The SKIM user can populate or retrieve a single TCAP dialogue and list of associated TCAP components (operations) using a single SKIM_Trans object instance. SKIM_Trans object supports the public methods that are explained in the following sections.

Important! Users are expected to know what methods from the SKIM_Trans public interface need to be invoked for a given transaction type. If you invoke a set method for an invalid transaction type, set method returns without changing underlying interface data.

Summary of Methods The following lists the methods available with the SKIM_Trans class.

```
Class SKIM_Trans
{
public:

SKIM_Trans();

    void SetTransType(SKIM_OCTET type);

    void GetTransType(SKIM_OCTET &type);

    SKIM_UINT GetCallReferenceId();

    void SetCallReferenceId(SKIM_UINT id);

    void AddOperation(SKIM_Operation &op);

    int RemoveOperation(SKIM_Operation &op);

    int GetNumOfOperation();

    void SetSourceAddress(SKIM_CallingPartyAddr &addr);

    void GetSourceAddress(SKIM_CallingPartyAddr &addr);

    void SetDestAddress(SKIM_CalledPartyAddr &addr);

    void GetDestAddress(SKIM_CalledPartyAddr &addr);

    void GetAbortReason(SKIM_OCTET &reason);
```

```
void SetAbortReason(SKIM_OCTET reason);

void GetAbortInfo(SKIM_OCTET *buf, int &len);

void SetAbortInfo(SKIM_OCTET *buf, int len);

void
    SetQualityOfService(const SKIM_OCTET flags,
                        const SKIM_OCTET priority = 0);

void
    GetQualityOfService(SKIM_OCTET& flags,
                        SKIM_OCTET& priority);

void SetPreArrangedEnd();

SKIM_OCTET GetReportCause () const;


void GetApplicationContext(SKIM_OCTET* buf, int& len)
const;

void SetApplicationContext(const SKIM_OCTET* buf,
const int len);

void SetUserInfo(const SKIM_OCTET* buf, const int
len);

void GetUserInfo(SKIM_OCTET* buf, int& len) const;

};
```

SKIM_Trans Class Methods

Purpose This section provides details about the methods available for the SKIM_Trans class.

SKIM_Trans() **Description**

SKIM_Trans is the default constructor.

Syntax

```
SKIM_Trans();
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

SetTransType() **Description**

Sets SKIM transaction type.

Syntax

```
void SetTransType(SKIM_OCTET type);
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Values

Possible values for ITU TCAP:

- SKIM_TC_UNI
- SKIM_TC_BEGIN
- SKIM_TC_CONTINUE
- SKIM_TC_END
- SKIM_TC_P_ABORT
- SKIM_TC_U_ABORT

Possible values for ANSI TCAP:

- SKIM_TC_UNI
- SKIM_TC_BEGIN (Same as Query with Permission)

- SKIM_TC_CONTINUE (Same as Conversation With Permission)
- SKIM_TC_END
- SKIM_TC_ABORT
- SKIM_TC_QUERY_WO_PERM
- SKIM_TC_CONV_WO_PERM

GetTransType() Description

Gets SKIM transaction type.

Syntax

```
void GetTransType(SKIM_OCTET &type);
```

Input Parameters

type

Return Values

Possible values for ITU TCAP:

Possible values for ITU TCAP:

- SKIM_TC_UNI, SKIM_TC_BEGIN
- SKIM_TC_CONTINUE
- SKIM_TC_END
- SKIM_TC_P_ABORT
- SKIM_TC_U_ABORT
- SKIM_TC_NOTICE:

This message is received in case of network problem at SCCP layer while sending a TCAP transaction message.

Possible values for ANSI TCAP:

- SKIM_TC_UNI
- SKIM_TC_BEGIN (Same as Query With Permission)
- SKIM_TC_CONTINUE (Same as Conversation With Permission)
- SKIM_TC_END
- SKIM_TC_ABORT
- SKIM_TC_QUERY_WO_PERM
- SKIM_TC_CONV_WO_PERM

- SKIM_TC_NOTICE

GetCallReferenceId()**Description**

This method returns the call reference ID. A call reference ID has one to one mapping with a TCAP dialogue ID.

Syntax

```
SKIM_UINT GetCallReferenceId();
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

Call reference ID.

SetCallReferenceId()**Description**

Sets the call reference ID. The call reference ID is not set when initiating a transaction.

Syntax

```
void SetCallReferenceId(SKIM_UINT id);
```

Input Parameters

id: Call Reference ID

Input/Output Parameters, Output Parameters

None

AddOperation()**Description**

Adds an operation to the given transaction object.

Syntax

```
void AddOperation(SKIM_Operation &op);
```

Input Parameters

op: SKIM_Operation object

Input/Output Parameters, Output Parameters

None

RemoveOperation()**Description**

Removes an operation from a transaction object.

Syntax

```
void RemoveOperation(SKIM_Operation &op);
```

Input Parameters, Input/Output Parameters

None

Output Parameters

op: SKIM_Operation object

Return Value

- SKIM_SUCCESS:
An operation removal is successful.
- SKIM_ENOMSG:
No operation is available.

GetNumOfOperation()**Description**

Gets the number of operations associated with a SKIM_Trans object at any given time.

Syntax

```
int GetNumOfOperation();
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

Number of operations

SetSourceAddress()**Description**

Sets the SCCP source address for a given transaction object. See Also: SKIM_CallingPartyAddress API.

Syntax

```
void SetSourceAddress(SKIM_CallingPartyAddr &addr);
```

Input Parameters

addr: SCCP Calling Party Address object.

Input/Output Parameters, Output Parameters

None

GetSourceAddress()**Description**

Gets the SCCP source address for a given transaction object. See Also: SKIM_CallingPartyAddress API.

Syntax

```
void GetSourceAddress(SKIM_CallingPartyAddr &addr);
```

Input Parameters, Input/Output Parameters

None

Output Parameters

addr: SCCP Calling Party Address object.

SetDestAddress()**Description**

Sets the SCCP destination address for a given transaction object. See also: SKIM_CalledPartyAddress API.

Syntax

```
void SetDestAddress(SKIM_CalledPartyAddr &addr);
```

Input Parameters

addr: SCCP Called Party Address object.

Input/Output Parameters, Output Parameters

None

GetDestAddress()**Description**

Gets the SCCP destination address for a given transaction object. See also: SKIM_CalledPartyAddress API.

Syntax

```
void GetDestAddress(SKIM_CalledPartyAddr &addr);
```

Input Parameters, Input/Output Parameters

None

Output Parameters

addr: SCCP Called Party Address object.

GetAbortReason() (For ITU)**Description**

Gets the abort reason for SKIM_TC_U_ABORT/
SKIM_TC_P_ABORT (ITU).

Syntax

```
void GetAbortReason(SKIM_OCTET &reason);
```

Input Parameters

reason: Abort reason. Possible values of abort reason are defined in
Appendix A.

Input/Output Parameters, Output Parameters

None

SetAbortReason (For ITU)**Description**

Sets the abort reason for SKIM_TC_U_ABORT.

Note: The CSP sends SKIM_TC_P_ABORT automatically.

Syntax

```
void SetAbortReason(SKIM_OCTET reason);
```

Input Parameters,

reason: Abort reason. Possible values of abort reason are defined in
Appendix A.

Input/Output Parameters and Output Parameters

None

GetAbortInfo() (For ANSI)**Description**

Gets the abort information for SKIM_TC_ABORT (ANSI transaction type.)

Syntax

```
void GetAbortInfo(SKIM_OCTET *buf, int &len);
```

Input Parameters

buf : Abort information.

len: Abort information length.

Input/Output Parameters, Output Parameters

None

SetAbortInfo (For ANSI)**Description**

Sets the abort information for SKIM_TC_ABORT (ANSI transaction type.)

Syntax

```
void SetAbortInfo(SKIM_OCTET *buf, int &len);
```

Input Parameters

info: Abort information.

length: Abort information length.

Input/Output Parameters and Output Parameters

None

SetQualityOfService()**Description**

Sets the quality of service.

Syntax

```
void SetQualityOfService(const SKIM_OCTET flags,
                        const SKIM_OCTET priority = 0);
```

Input Parameters,

flags: Flags can be combination of the following types (for ITU):

- 0 : specifies SCCP Class 0 with no return option
- SKIM_QOSI_RET_OPT: specifies SCCP Class 0 with return option
- SKIM_QOSI_SEQ_CTRL: specifies SCCP) Class 1 with no return option.
- SKIM_QOSI_SEQ_CTRL | SKIM_QOSI_RET_OPT: specifies SCCP Class 1 with return option set

flags: Flags can be combination of the following types (for ANSI)

- 8: specifies SCCP Class 0 with no return option.
- SKIM_QOSI_RET_OPT | |SKIM_QOSI_PRIORITY: specifies SCCP Class 0 with return option set.
- SKIM_QOSI_SEQ_CTRL | |SKIM_QOSI_PRIORITY: specifies SCCP Class 1 with no return option
- SKIM_QOSI_SEQ_CTRL | SKIM_QOSI_RET_OPT | |SKIM_QOSI_PRIORITY: specifies SCCP Class 1 with return option set.

Input/Output Parameters, Output Parameters

None

GetQualityOfService()**Description**

Gets the quality of service.

Syntax

```
void GetQualityOfService(SKIM_OCTET& flags, SKIM_OCTET&
                        priority);
```

Input Parameters, Input/Output Parameters

None

Output Parameters

flags: Flags can be combination of the following types (for ITU):

- 0 : specifies SCCP Class 0 with no return option
- SKIM_QOSI_RET_OPT: specifies SCCP Class 0 with return option
- SKIM_QOSI_SEQ_CTRL: specifies SCCP) Class 1 with no return option.
- SKIM_QOSI_SEQ_CTRL | SKIM_QOSI_RET_OPT: specifies SCCP Class 1 with return option set

flags: Flags can be combination of the following types (for ANSI)

- 8: specifies SCCP Class 0 with no return option.
- SKIM_QOSI_RET_OPT | |SKIM_QOSI_PRIORITY: specifies SCCP Class 0 with return option set.
- SKIM_QOSI_SEQ_CTRL | |SKIM_QOSI_PRIORITY: specifies SCCP Class 1 with no return option
- SKIM_QOSI_SEQ_CTRL | SKIM_QOSI_RET_OPT | |SKIM_QOSI_PRIORITY: specifies SCCP Class 1 with return option set.

priority: message priority.

SetPreArrangedEnd()**Description**

Sets the pre-arranged end for the SKIM_TC_END transaction type. By default the end type is set to basic.

Syntax

```
void SetPreArrangedEnd();
```

Input Parameters, Input/Output Parameters & Output Parameters

None

GetReportCause()**Description**

Gets the reported cause for SKIM_TC_NOTICE transaction type.

Syntax

```
SKIM_OCTET GetReportCause() const;
```

```
#if defined(CCITT) || defined(PRC)
```

Input Parameters, Input/Output Parameters & Output Parameters

None

Return Value:

Report cause value.

- SCCP_RET_NO_TRANS_ADDR_NAT
- SCCP_RET_NO_TRANS_THIS_ADDR
- SCCP_RET_SUBSYS_CONG
- SCCP_RET_SUBSYS_FAIL
- SCCP_RET_UNEQUIPPED_USER
- SCCP_RET_NETWORK_FAIL
- SCCP_RET_NETWORK_CONG
- SCCP_RET_UNQUAL
- SCCP_RET_ERR_IN_TRANSPORT
- SCCP_RET_ERR_IN_LOCAL_PROCESS
- SCCP_RET_ERR_DEST_CANNOT_PERF_REASSEMBLY
- SCCP_RET_ERR_SCCP_FAILURE (ITU Only)
- SCCP_RET_ERR_HOP_COUNT_VIOLATION (ANSI only)
- SCCP_RET_ERR_INV_ISNI_ROUT_REQ (ANSI only)
- SCCP_RET_ERR_UNAUTH_MSG (ANSI only)
- SCCP_RET_ERR_MSG_INCOMPATIBILITY (ANSI only)
- SCCP_RET_ERR_CANNOT_PERFORM_ISNI_ROUTING (ANSI only)
- SCCP_RET_ERR_REDUNDANT_ISNI_ROUTING_INFO (ANSI only)
- SCCP_RET_ERR_UNABLE_TO_IDENTIFY_ISNI_INFO (ANSI only)

**GetApplicationContext()
(ITU only)****Description**

Allows the user to get the application context for a transaction.

Syntax

```
void GetApplicationContext(SKIM_OCTET* buf, int& len)
    const;
```

Input Parameters, Input/Output Parameters

None

Output Parameters:

buf: Buffer to copy application context information.

len: Number of bytes copied in application context information.

**SetApplicationContext()
(ITU only)****Description**

Allows the user to set the application context for a transaction.

Syntax

```
void SetApplicationContext(const SKIM_OCTET* buf, const  
    int len);
```

Input Parameters

buf: Buffer to copy application context information.

len: Number of bytes copied in application context information.

Input/Output and Output Parameters:

None

SetUserInfo() (ITU only)**Description**

Allows the user to set the user information for a transaction.

Syntax

```
void SetUserInfo(const SKIM_OCTET* buf, const int len);
```

Input Parameters

buf: Buffer containing user information.

len: Number of bytes in user information.

Input/Output and Output Parameters:

None

GetUserInfo() (ITU only)**Description**

Allows the user to get the user information for a transaction.

Syntax

```
void GetUserInfo(SKIM_OCTET* buf, int& len) const;
```

Input Parameters, Input/Output Parameters

None

Output Parameters:

buf: Buffer containing user information.

len: Number of bytes in user information.

SKIM_Operation Class

Description The SKIM_Operation object supports the public methods that are explained in the following sections. A user is expected to know what methods from SKIM_Operation public interface need to be invoked for a given operation type. If you invoke a set method for an invalid operation type, the set method will simply return without changing the underlying interface data.

Summary of Methods The following lists the methods available with the SKIM_Operation class.

```
Class SKIM_Operation
{
public:

    int SetType(SKIM_USHORT type);

    SKIM_USHORT GetType();

    int SetInvokeId(SKIM_OCTET invokeId);

    SKIM_OCTET GetInvokeId();

    int SetLinkId(SKIM_OCTET id);

    SKIM_OCTET GetLinkId();

    void SetClass(SKIM_USHORT cl);

    SKIM_USHORT GetClass();

    voidint SetInvokeTimeout(SKIM_UINT timeout);

    voidint SetParameter(vector <byte> &param);

    voidint GetParameter(vector <byte> &param);

    bool IsLastOperation();

    voidint SetOpCode(const SKIM_LONG code);

    voidint GetOpCode(SKIM_LONG & code);

    voidint SetOpCode(const bool isNational, const
        SKIM_OCTET family,
        const SKIM_OCTET code);
```

```

voidint GetOpCode(bool& isNational, SKIM_OCTET&
family,
                    SKIM_OCTET& code);

voidint SetErrorCode(SKIM_OCTET errorCode);

voidint GetErrorCode(SKIM_OCTET &errorCode);

voidint SetErrorCode(SKIM_BOOLEAN isNational,
SKIM_OCTET errorCode);

voidint GetErrorCode(SKIM_BOOLEAN &isNational,
SKIM_OCTET &errorCode);

voidint SetRejectCause(const SKIM_OCTET type, const
SKIM_OCTET code);

voidint GetRejectCause(SKIM_OCTET& type, SKIM_OCTET&
code);

};

```

Operation Types

You are expected to know what methods from the **SKIM_Operation** public interface need to be invoked for a given operation type. If you invoke a set method for an invalid operation type, set method will simply return without changing underlying interface data.

Refer to the following table for applicable methods for operation types.

Method	Operation Types
SetType ()	All
GetType ()	All
SetInvokeId ()	All
GetInvokeId ()	All
SetLinkId ()	SKIM_TC_INVOKE SKIM_TC_INVOKE_NLI
HasLinkId ()	SKIM_TC_INVOKE SKIM_TC_INVOKE_NL
GetLinkId ()	SKIM_TC_INVOKE SKIM_TC_INVOKE_NL
SetInvokeTimeout ()	SKIM_TC_INVOKE

Method	Operation Types
Set Parameter ()	SKIM_TC_INVOKE, SKIM_TC_INVOKE_NL, SKIM_TC_RESULT_L, SKIM_TC_RESULT_NL, SKIM_TC_U_ERROR, SKIM_TC_ERROR, and SKIM_TC_REJECT
Get Parameter ()	SKIM_TC_INVOKE, SKIM_TC_INVOKE_NL, SKIM_TC_RESULT_L, SKIM_TC_RESULT_NL, SKIM_TC_U_ERROR, SKIM_TC_ERROR SKIM_TC_REJECT
IsLastOperation ()	All
SetOpCode ()	SKIM_TC_INVOKE, SKIM_TC_INVOKE_NL, SKIM_TC_RESULT_L (ITU) SKIM_TC_RESULT_NL (ITU)
GetOpCode ()	SKIM_TC_INVOKE, SKIM_TC_INVOKE_NL, SKIM_TC_RESULT_L (ITU) SKIM_TC_RESULT_NL (ITU)
SetClass ()	SKIM_TC_INVOKE (ITU)
GetClass ()	SKIM_TC_INVOKE (ITU)
SetErrorCode ()	SKIM_TC_U_ERROR SKIM_TC_ERROR
GetErrorCode ()	SKIM_TC_U_ERROR SKIM_TC_ERROR
SetRejectCause ()	SKIM_TC_U_REJECT SKIM_TC_R_REJECT SKIM_TC_L_REJECT SKIM_TC_REJECT
GetRejectCause ()	SKIM_TC_U_REJECT SKIM_TC_R_REJECT SKIM_TC_L_REJECT SKIM_TC_REJECT

SKIM_Operation Class Methods

Purpose This section provides details about the methods available for the SKIM_Operation class.

SetType() **Description**

Sets the type of this operation

Syntax

```
int SetType(SKIM_USHORT type);
```

Input Parameters

type: the operation type is one of the following:

- SKIM_TC_INVOKE (Same as Invoke Last for ANSI)
- SKIM_TC_INVOKE_NL
- SKIM_TC_RESULT
- SKIM_TC_RESULT_L
- SKIM_TC_ERROR
- SKIM_TC_REJECT (For ANSI)
- SKIM_TC_U_REJECT (For ITU)
- SKIM_TC_CANCEL

Input/Output Parameters, Output Parameters

None

GetType() **Description**

Returns the type of this operation.

Syntax

```
SKIM_USHORT GetType();
```

Input Parameters, Input/Output Parameters & Output Parameters:

None

Return Values

The type of this component is one of the following:

- SKIM_TC_INVOKE

- SKIM_TC_INVOKE_NL (Same as Invoke Last for ANSI)
- SKIM_TC_RESULT
- SKIM_TC_RESULT_L
- SKIM_TC_ERROR
- SKIM_TC_REJECT
- SKIM_TC_U_REFJECT (For ITU)
- SKIM_TC_R_REFJECT (For ITU)
- SKIM_TC_L_REFJECT (For ITU)
- SKIM_TC_CANCEL

SetInvokeld() Description

Populates the invoke ID for this operation. Invoke IDs are common to all operation types. For ANSI TCAP, use this API to set the correlation ID for RESULT/ERROR/REJECT operations.

Syntax

```
void SetInvokeId(SKIM_OCTET invokeId);
```

Input Parameters

value: The invoke ID value. The valid range is 0-255.

Input/Output Parameters

None

Output Parameters

None

GetInvokeld() Description

Returns the invoke ID of an operation. Invoke IDs are common to all operation types. For ANSI TCAP, use this API to get the correlation ID for RESULT, ERROR, and REJECT operations.

Syntax

```
SKIM_OCTET GetInvokeId();
```

Input Parameters, Input/Output Parameters & Output Parameters

None

Return Value

The invoke ID as an unsigned character.

SetLinkId()**Description**

Sets the linked ID (ITU) or correlation ID (ANSI) to the value supplied. This is valid only when the operation type is SKIM_TC_INVOKE or SKIM_TC_INVOKE_NL. For any other operation type this method simply returns and nothing is set. You do not need to call this method if the linked ID is not relevant. It is assumed that no linked ID is present unless this method is called.

Syntax

```
void SetLinkId(SKIM_OCTET id);
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

HasLinkId()**Description**

Returns SKIM_TRUE if the linked ID (ITU) or correlation ID (ANSI) is present in Invoke operation. This is valid only when the operation type is SKIM_TC_INVOKE.

Syntax

```
SKIM_BOOLEAN HasLinkId();
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

SKIM_TRUE: Linked ID present.

SKIM_FALSE: Linked ID not present.

GetLinkId()**Description**

Gets the linked ID (ITU) or correlation ID (ANSI) of an invoke, if supplied. This is valid only when the operation type is SKIM_TC_INVOKE or SKIM_TC_INVOKE_NL.

Syntax

```
SKIM_OCTET GetLinkId();
```

Return Value

The linked ID is returned as an unsigned character.

SetClass()**Description**

Sets the operation class for TCAP components. The SetClass() method is used for ITU TCAP only.

Syntax

```
void SetClass(const SKIM_USHORT val);
```

Input Parameters

val: A number from 1 to 4 indicating the operation class:

- Class 1: Both success and failure are reported
- Class 2: Only failure is reported
- Class 3: Only success is reported
- Class 4: Neither success, or failure is reported

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

None

GetClass()**Description**

Retrieves the operation class for TCAP invokes. The GetClass method is used for ITU TCAP only.

Syntax

```
SKIM_USHORT GetClass() const;
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Values

A number from 1 to 4 indicating the operation class:

- Class 1: Both success and failure are reported
- Class 2: Only failure is reported
- Class 3: Only success is reported

- Class 4: Neither success, nor failure is reported

SetInvokeTimeout()**Description**

Sets the timeout value for TCAP invokes. This is valid only when the operation type is SKIM_TC_INVOKE (used for ITU only).

Syntax

```
void SetInvokeTimeout(SKIM_UINT timeout);
```

Input Parameters

val: the timeout value in seconds. Range is 1-500.

Input/Output Parameters and Output Parameters

None

SetParameter()**Description**

Copies a user-defined parameter into the given operation. If an operation does not have a parameter, you do not need to call this method. The parameter is assumed to be not present unless this method is called with a non-zero vector.

Syntax

```
void SetParameter(vector <byte> &param);
```

Input Parameters,

buf: A vector containing the parameter to copy into this operation.

Input/Output Parameters and Output Parameters:

None

GetParameter()**Description**

Copies the parameter in a given operation into a user-supplied vector. Returning a vector.size () of zero indicates that no parameter is present.

Syntax

```
void GetParameter(vector <byte> &param);
```

Input Parameters and Input/Output Parameters

None

Output Parameters:

buf: A vector for the parameter to copy.

IsLastOperation()**Description**

Specifies whether the given operation is the last incoming operation associated with a TCAP transaction.

Syntax

```
bool IsLastOperation();
```

Input Parameters, Input/Output Parameters & Output Parameters

None

Return Value

SKIM_TRUE: If given operation is the last operation.

SKIM_False: If given operation is not the last operation.

SetOpCode() (For ITU)**Description**

Sets the operation field of a given operation. This is valid only when operation type is:

- SKIM_TC_INVOKE
- SKIM_TC_RESULT_L
- SKIM_TC_RESULT_NL

Syntax

```
void SetOpCode(const SKIM_LONG code);
```

Input Parameters, Input/Output Parameters & Output Parameters

None

GetOpCode()(For ITU)**Description**

Gets the operation field of a given operation. This is valid only when the operation type is:

- SKIM_TC_INVOKE
- SKIM_TC_RESULT_L
- SKIM_TC_RESULT_NL

Syntax

```
void GetOpCode(SKIM _LONG & code);
```

Input Parameters, Input/Output Parameters

None

Output Parameters

code: The operation code.

SetOpCode() (For ANSI)**Description**

Sets the operation field of a given operation. This is valid only when operation type is:

- SKIM_TC_INVOKE
- SKIM_TC_RESULT_L
- SKIM_TC_RESULT_NL

Syntax

```
void SetOpCode(const bool isNational, const SKIM_OCTET
    family, const SKIM_OCTET code);
```

Input Parameters,

IsNational: True if National, false if private

family: Operation family. Possible values are provided in Appendix A *Table A-1, ANSI TCAP Operation Family (A-2)*.

code: The operation code specifier.

Input/Output Parameters and Output Parameters:

None

GetOpCode()(For ANSI)**Description**

Gets the operation field of a given operation. This is valid only when operation type is:

- SKIM_TC_INVOKE
- SKIM_TC_RESULT_L
- SKIM_TC_RESULT_NL

Syntax

```
GetOpCode(bool& isNational, SKIM_OCTET& family,  
           SKIM_OCTET& code);
```

Input Parameters and Input/Output Parameters

None

Output Parameters

IsNational: True if National, false if private

family: Operation family. Possible values are provided in Appendix A *Table A-1, ANSI TCAP Operation Family (A-2)*.

code: the operation code.

SetErrorCode() (For ITU)**Description**

Sets the error code in an error operation.

Syntax

```
void SetErrorCode(SKIM_OCTET errorCode);
```

Input Parameters

code: the error code.

Input/Output Parameters and Output Parameters:

None

GetErrorCode() (For ITU)**Description**

Gets the error code in an error operation.

Syntax

```
void GetErrorCode(SKIM_OCTET &errorCode);
```

Input Parameters and Input/Output Parameters

None

Output Parameters

code: the error code.

SetErrorCode() (For ANSI)**Description**

Sets the error code in an error operation.

Syntax

```
void SetErrorCode(SKIM_BOOLEAN isNational, SKIM_OCTET  
    errorCode);
```

Input Parameters

IsNational: True if National, false if private.

code: The error code. Possible values are provided in *Appendix - TCAP Codes (A-1)*.

Input/Output Parameters and Output Parameters

None

GetErrorCode() (For ANSI)**Description**

Gets the error code in an error operation.

Syntax

```
void GetErrorCode(SKIM_BOOLEAN &isNational, SKIM_OCTET  
    &errorCode);
```

Input Parameters, Input/Output Parameters

None

Output Parameters

IsNational: True if National, false if private.

code: The error code. Possible value provided in *Appendix - TCAP Codes (A-1)*.

SetRejectCause()**Description**

Sets the problem code for a reject operation.

Syntax

```
void SetRejectCause(const SKIM_OCTET type, const  
SKIM_OCTET code);
```

Input Parameters

type: The problem family. As specified in Appendix A.

code: The problem code. As specified in Appendix A.

Input/Output Parameters and Output Parameters

None

GetRejectCause()**Description**

Gets the problem code for a reject operation.

Syntax

```
void GetRejectCause(SKIM_OCTET& type, SKIM_OCTET& code);
```

Input Parameters and Input/Output Parameters

None

Output Parameters

type: The problem family

code: The problem code

SKIM_Notify Class

Description This section describes the SKIM_Notify public methods. The SKIM_Notify class provides a public interface for retrieving SKIM notification information.

Summary of Methods The following lists the methods available with the SKIM_Notify class.

```
Class SKIM_Notify{
public:
    int GetType();
    int GetCallReferenceId(SKIM_UINT &id);
    int GetCause();
    SKIM_OCTET GetCauseType();
    SKIM_OCTET GetSSN();
    SKIM_UINT GetPC();
    SKIM_GetInvokeID();
};
```

SKIM_Notify Class Methods

Purpose This section provides details about the methods available for the SKIM_Notify class.

GetType() **Description**

Gets the notification type.

1. When the user receives a notification, the user should use appropriate get methods based on notification object. The default value returned by all GetXX methods is 0. Zero means that the parameter is not set for the notification.
2. For the SKIM_NOTIFY_NACK_IND, GetCauseType() provides a transaction type / operation type for the NACK that is received.
3. The GetCallReferenceId() method is relevant for SKIM_NOTIFY_OP_TIMEOUT notifications. This method returns the call reference ID of the transaction for which a SKIM_NOTIFY_OP_TIMEOUT is received.
4. The SKIM_NOTIFY_OP_TIMEOUT corresponds to TC-L-CANCEL when an outgoing Invoke operation timeout occurs in the TCAP stack.
5. For the SKIM_NOTIFY_SCCP_MGMT_IND notification, GetCause() provides the type of SCCP N-State indication received. GetSSN() and GetPC() methods provide the affected PC and SSN.
6. For the SKIM_NOTIFY_MTP3_MGMT_IND notification, GetCause() provides the type of SCCP N-PCState indication received. GetPC() method provides the affected point code.

Syntax

```
int GetType();
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value: Notification type. Possible values are:

Value	Description
SKIM_NOTIFY_SCCP_MGMT_IND	SCCP management indication.
SKIM_NOTIFY_MTP3_MGMT_IND	N-PC State management indication.

SKIM_NOTIFY_OP_TIMEOUT	Operation timeout. For ITU only.
SKIM_NOTIFY_INTERNAL_ERROR	SKIM internal error.
SKIM_NOTIFY_NACK_IND	PPL NACK Indication form EXS.

GetCallReferenceId() **Description**

Gets the call reference ID for a given notification.

Syntax

```
SKIM_UINT GetCallReferenceId;
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

id: Call reference ID

GetCause() **Description**

Returns the notification cause corresponding to a given notification type. For the SKIM_NOTIFY_NACK_IND notification type, *GetCause ()* returns a PPL event NACK cause value (ack status) received from the CSP.

The SKIM_NOTIFY_INTERNAL_ERROR notification is received for the following causes:

- SKIM_TRANS_RECORD_ERROR -Transaction record error
- SKIM_INVALID_COMP_ERROR - Invalid component received
- SKIM_COMP_RECV_ERROR - Receive error/failed
- SKIM_DUP_INVID_ERROR - Duplicate invoke ID
- SKIM_UNK_INVID_ERROR - Unknown invoke ID
- SKIM_INV_MSG_TYPE_ERROR - Invalid message type
- SKIM_RES_LIMT_ERROR - Resource limitation.

Syntax

```
int GetCause();
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

Notification cause value.

GetCauseType()**Description**

Returns cause type values associated with the SKIM_NOTIFY_NACK_IND notification. For the SKIM_NOTIFY_NACK_IND notification, the cause type value indicates the transaction type or operation type for which the NACK is received from the CSP.

Syntax

```
SKIM_OCTET GetCauseType();
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

Cause Type.

GetSSN()**Description**

Gets affected subsystem number associated with SKIM_NOTIFY_SCCP_MGMT_IND notification

Syntax

```
SKIM_OCTET GetSSN();
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

Subsystem number.

GetPC()**Description**

Gets affected point code associated with SKIM_NOTIFY_SCCP_MGMT_IND and SKIM_NOTIFY_MTP3_MGMT_IND notification.

Syntax

```
SKIM_UINT GetPC();
```

Input Parameters, Input/Output Parameters & Output Parameters

None

Return Value

Point code value.

GetInvokeld()**Description**

Returns the invoke ID of an operation. Invoke IDs are common to all operation types.

Syntax

```
SKIM_OCTET GetInvokeId();
```

Input Parameters, Input/Output Parameters & Output Parameters

None

Return Value

The invoke ID as an unsigned char.

SKIM_CallingPartyAddress / SKIM_CalledPartyAddress Class

Description SKIM_CallingPartyAddress / SKIM_CalledPartyAddress object supports the public methods that are explained in the following sections.

Summary of Methods The following lists the methods available with the SKIM_CallingPartyAddress / SKIM_CalledPartyAddress class.
Class SKIM_CallingPartyAddress / SKIM_CalledPartyAddress
{
public:

Bool HasPointCode() const

SKIM_UINT GetPointCode() const

void

```
SetPointCode(const SKIM_UINT pc)
{
    hasPC = true;
    pointCode = pc;
}
```

Bool HasSSN() const

SKIM_OCTET GetSSN() const

Void SetSSN(const SKIM_OCTET sn)

Bool IsRoutedByPCSSN() const

Void SetRouteByPCSSN(const bool which)

Bool IsInternationalRouting() const

void

```
SetInternationalRouting(const bool which)
```

bool

```
HasGlobalTitle() const
```

int

```
SetGlobalTitle(const SKIM_OCTET type,
               const SKIM_OCTET* gttInfo, const
SKIM_USHORT gttLength)
```

```
int  
    GetGlobalTitle(SKIM_OCTET& type,  
                   SKIM_OCTET* gttInfo, SKIM_USHORT&  
gttLength) const  
};
```

SKIM_CallingPartyAddress/SKIM_CalledParty Address Class Methods

Purpose This section provides details about the methods available for the SKIM_CallingPartyAddress / SKIM_CalledPartyAddress class.

HasPointCode() **Description**

Determines whether an SCCP_Address object has a point code.

Syntax

```
Bool HasPointCode() const
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

A Boolean value, true if the point code was set, false otherwise.

GetPointCode() **Description**

Retrieves the point code value from an SCCP_Address object.

Syntax

```
SKIM_UINT GetPointCode() const
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

Returns the point code value as an unsigned integer.

SetPointCode() **Description**

Sets the point code value for an SCCP_Address object.

Syntax

```
void  
SetPointCode(const SKIM_UINT pc)  
{  
    hasPC = true;
```

```
pointCode = pc;
```

Input Parameters

pc: The value to store for the point code.

Input/Output Parameters and Output Parameters

None

Return Value

None

HasSSN ()

Description

Determines whether an SCCP_Address object has a subsystem number (SSN).

Syntax

```
Bool HasSSN() const
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

A boolean value, true if the SSN was set, false otherwise.

GetSSN ()

Description

Retrieves the subsystem number value from an SCCP_Address object.

Syntax

```
void
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

```
SKIM_OCTET GetSSN() const
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

Returns the subsystem number value as an unsigned char.

SetSSN ()

Description

Sets the subsystem number value for an SCCP_Address object.

Syntax

```
Void SetSSN(const SKIM_OCTET sn)
```

Input Parameters:

ssn: The value to store for subsystem number (SSN).

Input/Output Parameters and Output Parameters

Return Value

None

IsRoutedByPCSSN()

Description

Determines whether the “routed by” flag value for an SCCP_Address object has been set. Setting this to false implies routing by global title translation (GTT). To send this message properly with the false setting, you must use the SetGlobalTitle() method.

Syntax

```
Bool IsRoutedByPCSSN() const
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

True if the value has been set, otherwise False.

SetRouteByPCSSN()

Description

Sets the “routed by” flag value for an SCCP_Address object. Setting this to false implies routing by global title translation (GTT).

Syntax

```
Void SetRouteByPCSSN(const bool which)
```

Input Parameters

which: the value to store for the “routed by” flag.

Input/Output Parameters and Output Parameters

None

Return Value

None

IsInternationalRouting()

Description

Determines whether an SCCP_Address object had the international routing flag set.

Syntax

```
Bool IsInternationalRouting() const
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

A boolean value, true if the flag was set, false otherwise.

SetInternationalRouting ()

Description

Sets the international routing flag value for an SCCP_Address object.

Syntax

```
void SetInternationalRouting(const bool which)  
    // Global title
```

Input Parameters

which: The value to store for the international routing flag.

Input/Output Parameters and Output Parameters

None

HasGlobalTitle()

Description

Determines whether an SCCP_Address object has global title information.

Syntax

```
bool
    HasGlobalTitle() const
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

A boolean value, true if the global title was set, false otherwise.

SetGlobalTitle()

Description

Sets global title information for this address.

Syntax

```
int
    SetGlobalTitle(const SKIM_OCTET type,
                   const SKIM_OCTET* gttInfo, const
SKIM_USHORT gttLength)
```

Input Parameters

which: The value to store for the international routing flag

Input/Output Parameters and Output Parameters

None

type: The address indicator flag for this global title translation (GTT) type.

Possible values for ITU SCCP are:

Value	Description
SCCP_CPA_GTTI_NATURE	Global title contains nature of address.
SCCP_CPA_GTTI_TRANS	Global Title contains translation type.
SCCP_CPA_GTTI_TNE	Global Title contains translation type, numbering plan and encoding scheme.
SCCP_CPA_GTTI_ALL	Global title contains translation type, nature of address, numbering plan, and encoding scheme

Possible values for ANSI SCCP are:

Value	Description
SCCP_CPA_GTTI_TRANS	Global Title contains translation type.
SCCP_CPA_GTTI_TNE	Global Title contains translation type, numbering plan and encoding scheme.

gttInfo: The global title information. Global title information should be encoded in accordance with GTT type, as specified in ITU / ANSI SCCP specification.

gttLength: the size of the GTT information.

Return Value

None

GetGlobalTitle Description

Gets the global title information for this address.

Syntax

```
int
    GetGlobalTitle(SKIM_OCTET& type,
                  SKIM_OCTET* gttInfo, SKIM_USHORT&
                  gttLength) const
```

Input Parameters and Input/Output Parameters

None

Output Parameters

type: The address indicator flag for this GTT type.

Possible values for ITU SCCP are:

Value	Description
SCCP_CPA_GTTI_NATURE	Global title contains nature of address.
SCCP_CPA_GTTI_TRANS	Global Title contains translation type.
SCCP_CPA_GTTI_TNE	Global Title contains translation type, numbering plan, and encoding scheme.

Value	Description
SCCP_CPA_GTTI_ALL	Global title contains translation type, nature of address, numbering plan and encoding scheme.

Possible values for ANSI SCCP are:

Value	Description
SCCP_CPA_GTTI_TRANS	Global Title contains translation type.
SCCP_CPA_GTTI_TNE	Global Title contains translation type, numbering plan and encoding scheme.

gttInfo: The global title translation information. Global title information should be encoded in accordance with GT type, as specified in ITU / ANSI SCCP specification.

gttLength: The size of the GTT information.

Return Value

None

4 SKIM Sample Application

Purpose This chapter describes a sample application using the SKIM API. Six key steps are explained.

Environment Variables for Sample Applications

Requirement Sample applications are included in the installation directory and require setting environment variables. The default installation directory is:

SK-IM-TAL-API/skim/test

The following environment variables are required:

- ITS_ROOT
- INTL_ROOT

ITS_ROOT ITS_ROOT is used during compiling, linking and at runtime to locate the SKIM and SKTAL API.

Valid Values

Full path name where the SKIM SKTAL API is installed

INTL_ROOT INTL_ROOT is used during compiling, linking and at runtime to locate the Intellinet Codec.

Valid Values

Full path name where the Intellinet Codec is installed.

Step One: Creating a connection with LLC using SwitchKit

Purpose The API function described in this section allows you to create a connection with LLC using SwitchKit.

Application Code

```
{
    int ret;

    /* create the connection */
    ret = skts_createConnection(CONN_ID, CONN_LABEL,
                                CONN_NAME,
                                SKIM_FALSE,
                                PRI_IP, PRI_PORT,
                                BACK_IP, BACK_PORT);

    if (ret == 0)
    {
        fprintf(stderr, "Error connecting to switch\n");
        return (SKIM_EINVALIDARGS);
    }

    return (SKIM_SUCCESS);
}
```

Step Two: Creating an Instance of SKIM API Class

Purpose The API function described in this section allows you to create an instance of a SKIM API class.

Application Code

```
SKIM_Api api_classptr;

// LOCAL_NODE - value created during configuration
// STACK_ID - Stack Id for the stack instance created
// during
// configuration.
// CONN_ID - Same as created in step one.
// LOCAL_PC - Local Point Code for stack instance.
// LOCAL_SSN - local subsystem number for stack
// instance.
// APP_TransHandler - Default Handler for handling
// incoming
// TCAP Transaction messages.
// APP_NotifyHandler - Default Handler for handling
// incoming
// SKIM notifications.
// tag - Tag to be returned with handler functions.
res = api_classptr.Initialize(LOCAL_NODE, STACK_ID,
CONN_ID, LOCAL_PC,
                                LOCAL_SSN,
                                APP_TransHandler,
                                APP_NotifyHandler, NULL);

if (res != SKIM_SUCCESS)
{
    printf("\n Initialization FAILED !!!!!!!!!");
    api_classptr.Terminate();
    exit(0);
}
else
{
    printf("\n SUCCESS FROM INITIALIZATION API !!\n");
}
```

Step Three: Sending a TCAP Transaction

Purpose The API function described in this section allows you to send a TCAP transaction.

Application Code

```
// Create SKIM_Trans object of type SKIM_TC_BEGIN
SKIM_Trans trans (SKIM_TC_BEGIN);

// Create objects for calling party and called party
// address.
SKIM_CallingPartyAddr origAddr;
SKIM_CalledPartyAddr destAddr;

// Populate OPC and DPC
trans.SetOPC(LOCAL_PC);
trans.SetOPC(REMOTE_PC);

// populate calling party address
origAddr.SetSSN(LOCAL_SSN);
origAddr.SetPointCode(LOCAL_PC);
origAddr.SetRouteByPCSSN(true);

#if defined(CCITT)
    origAddr.SetInternationalRouting(true);
#elif defined (ANSI)
    origAddr.SetInternationalRouting(false);
#endif

// Set calling party address for begin transaction.
trans.SetSourceAddress(origAddr);

//Set Called Party Address
destAddr.SetSSN(REMOTE_SSN);
destAddr.SetPointCode(REMOTE_PC);
destAddr.SetRouteByPCSSN(true);
#if defined(CCITT)
    destAddr.SetInternationalRouting(true);
#elif defined (ANSI)
    destAddr.SetInternationalRouting(false);
#endif
// Set called party address for begin transaction.
trans.SetDestAddress(destAddr);

// Set quality of service for the transaction message.
SKIM_OCTET priority;
```

```
priority = 0;

trans.SetQualityOfService(QOSI_RET_OPT|SKIM_QOSI_RET_0
PT, priority);

// Adding a operation to transaction object.

// Create operation of type SKIM_TC_INVOKE
SKIM_Operation op(SKIM_TC_INVOKE);

// Set Invoke Id.
op.SetInvokeId(1 + i);

// Set Operation Code
#ifdef CCITT
    op.SetOpCode(100);

// Set Operation Class for invoke operation.
op.SetClass(1); // Set Invoke operation timeout
op.SetInvokeTimeout(10);
#else
    op.SetOpCode(true, 1, 1);
#endif

// Set ASN.1 encoded TCAP User parameters
op.SetParameter(SKIM_Encode());

// Add operation to the transaction object.
trans.AddOperation(op);

// Send transaction to EXS
ret = api_classptr.Send(trans);
if (ret != SKIM_SUCCESS)
{
    printf("Send transaction failed ret = %d", ret);
    return ret;
}

// Get the call reference id allocated for this
transaction.
SKIM_UINT callrefid = ;
trans.GetCallReferenceId();
```

Step Four: Receiving a TCAP Transaction

Purpose The API function described in this section allows you to receive a TCAP transaction. Transactions are received via transaction handler and registered during initialization. Sample implementation for transaction handler is shown in the following code excerpt.

Application Code

```
int APP_TransHandler( SKIM_Trans &trans, void *tag)
{
    SKIM_OCTET type;

    printf("APP_TransHandler Called\n");

    SKIM_UINT id;
    trans.GetCallReferenceId(id);

    trans.GetTransType(type);
    switch(type)
    {
        case SKIM_TC_UNI:
            rcvd_callrefid = id;
            printf ("Received TCAP UNI call reference id
is %x\n", id);
            AppReceiveOperation(trans);

            break;

        case SKIM_TC_BEGIN :
            rcvd_callrefid = id;
            printf ("Received TCAP Begin call reference id
is %x\n", id);
            AppReceiveOperation(trans);

            break;

        case SKIM_TC_CONTINUE :
            rcvd_callrefid = id;
            printf ("Received TCAP Continue call reference
id is %x\n", id);
            AppReceiveOperation (trans);

            break;

        case SKIM_TC_END :
            rcvd_callrefid = id;
```

```
        printf ("Received TCAP End call reference id
is %x\n", id);
        AppReceiveOperation (trans);

        break;

#ifdef ANSI
        case SKIM_TC_ABORT :
            rcvd_callrefid = id;
            printf ("Received TCAP Abort call reference id
is %x\n", id);

            break;
#endif
#ifdef CCITT
        case SKIM_TC_U_ABORT :
            rcvd_callrefid = id;
            printf ("Received TCAP Abort call reference id
is %x\n", id);

            break;

        case SKIM_TC_P_ABORT :
            rcvd_callrefid = id;
            printf ("Received TCAP Abort call reference id
is %x\n", id);

            break;

        case SKIM_TC_P_ABORT :
            rcvd_callrefid = id;
            printf ("Received TCAP Abort call reference id
is %x\n", id);

            break;
#endif

        default :
            printf("Unknown transaction received\n");
            break;
    };

    return SKIM_SUCCESS;
}

AppReceiveOperation(SKIM_Trans &trans)
{
    SKIM_Operation op;
```

```

while (trans.RemoveOperation(op) == SKIM_SUCCESS)
{
    if (op.GetType() == SKIM_TC_INVOKE)
    {
        rcvd_invoke_id = op.GetInvokeId();
        printf("Received Invoke with Invoke id =
%d\n",
                op.GetInvokeId());
        vector<byte> paramData;
        op.GetParameter(paramData);

        SKIM_Decode(paramData);
    }
    else if (op.GetType() == SKIM_TC_RESULT_L)
    {
        printf("Received Return Result Last with
Invoke id = %d\n",
                op.GetInvokeId());
        vector<byte> paramData;
        op.GetParameter(paramData);

        SKIM_DecodeResponse(paramData);
    }
    #if defined(ANSI)
    else if (op.GetType() == SKIM_TC_ERROR)
    {
        printf("Received Return Error with Invoke id
= %d\n",
                op.GetInvokeId());
    }
    #endif
    #if defined(CCITT)
    else if (op.GetType() == SKIM_TC_U_ERROR)
    {
        printf("Received Return Error Last with
Invoke id = %d\n",
                op.GetInvokeId());
    }
    #endif
    #if defined(ANSI)
    else if (op.GetType() == SKIM_TC_REJECT)
    {
        printf("Received Return Reject with Invoke id
= %d\n",
                op.GetInvokeId());
    }
    #endif
    #if defined(CCITT)
    else if (op.GetType() == SKIM_TC_U_REJECT ||
            op.GetType() == SKIM_TC_L_REJECT ||

```

```
        op.GetType() == SKIM_TC_R_REJECT)
    {
        printf("Received Return Error Last  with
Invoke id = %d\n",
            op.GetInvokeId());
    }
#endif
else
{
    printf("Unknown operation type received\n");
}
}
}
```

Step Five: Receiving Notifications

Purpose Notifications are received via notification handler, registered during initialization. A sample implementation for the notification handler is shown in the following code excerpt.

Application Code

```
int APP_NotifyHandler( SKIM_Notify &notify, void *tag)
{
    printf("APP_NotifyHandler Invoked\n");
    int type = notify.GetType();

    switch(type)
    {
        case SKIM_NOTIFY_OP_TIMEOUT :

            printf ("Operation Timeout Notification
Message Received\n");

            break;

        case SKIM_NOTIFY_SCCP_MGMT_IND :

            printf ("SCCP Management Notification Message
Received\n");

            break;

        case SKIM_NOTIFY_MTP3_MGMT_IND :

            printf ("MTP3 Management Notification Message
Received\n");

            break;

        case SKIM_NOTIFY_INTERNAL_ERROR :

            printf ("SKIM Internal Error Notification
Message Received\n");

            break;

        default :

            printf("Unknown Notification Received \n");

            break;
    };
};
```

```
        return SKIM_SUCCESS;  
    }
```

Step Six: Terminating SKIM API Object Disconnection

Purpose The API function described in this section is used to terminate a SwitchKit Interface Module API object and disconnect from the LLC.

Application Code

```
api_classptr.Terminate();

if (CONN_ID)
{
    skts_destroyConnection(connection);
    connection = NULL;
}

classc
```

5 SwitchKit TCAP Abstraction Layer

Purpose This chapter provides information about the SwitchKit TCAP Abstraction Layer (SKTAL) messaging API. SKTAL provides the following APIs to the application.

Important! SKTAL data types and return codes are defined in Appendix C.

SwitchKit TCAP Abstraction Layer Messaging API

Purpose SwitchKit TCAP Abstraction Layer (SKTAL) provides various methods to develop an application.

Summary of Methods This section describes the interface methods that are provided with the SwitchKit TCAP Abstraction Layer (SKTAL). Further on, you will see details of each method.

```
int sktal_initializeTCAP();

SKTAL_HANDLE sktal_getTCAPHandle(int instanceId);

int sktal_registerTCAPSSNHandler (int localNodeId,
                                int stackId,
                                int connId,
                                int pc,
                                int ssn,
                                SK_TCAP_NewDialogueHandler h,
                                int *instanceId);

int sktal_unregisterTCAPSSNHandler (int instanceId);

int sktal_setTCAPDialogueHandler(int instanceId,
                                unsigned dialogueId,
                                SKTAL_TCAP_DialogueHandler handler,
                                void *tag);

int sktal_clearTCAPDialogueHandler(int instanceId,
                                unsigned dialogueId);

int sktal_allocateTCAPDialogueID(int instanceId, unsigned *did);

int sktal_sendTCAPDialog(int instanceId, unsigned dialogueId, SKTAL_DLG *dlg);

int sktal_recvTCAPDialog(int instanceId, SKTAL_EVENT *ev,
                        unsigned *dialogueId, SKTAL_DLG *dlg);

int sktal_sendTCAPComponent(int instanceId, unsigned dialogueId, SKTAL_CPT *cpt);

int sktal_recvTCAPComponent(int instanceId, SKTAL_EVENT *ev,
                        unsigned *dialogueId, SKTAL_CPT *cpt);
```

sktal_initializeTCAP()

Description This method is one time call for global initialization of TCAP Abstraction layer.

Syntax `int sktal_initializeTCAP();`

Parameters There are no input parameters, input/output parameters, or output parameters.

Return Value SKTAL_SUCCESS: If method is successful.

SKTAL return code as defined in Appendix C *Table C-2, SKTAL Return Codes (C-3)*.

sktal_terminateTCAP()

Description This method is a one time call for global termination of TCAP Abstraction layer.

Syntax `void sktal_terminateTCAP();`

Parameters There are no input parameters, input/output parameters, or output parameters.

Return Value None.

sktal_getTCAPHandle()

Description This function returns a handle for a given instance ID (node/stack/SSN). The handle is a pointer to a control block for the given instance ID.

Syntax SKTAL_HANDLE sk_getTCAPHandle(int instanceId);

Input Parameters

Argument	Description
instance id	Instance corresponding to node ID/stack ID/SSN.

Parameters There are no input/output parameters, or output parameters.

Return Value Returns a handle to control block.

sktal_registerTCAPSSNHandler()

Description This function registers the application with a given node/stack/subsystem. A generic handler will be added to the handler stack that will filter messages for the given subsystem. When a new TCAP transaction is seen, or when sk_TCAP_AllocateDialogueID is called, the NewDialogueHandler function argument will be invoked. This function is expected to register a per dialogue handler (via sktal_setTCAPDialogHandler) for the new dialogue. All incoming messages within that dialogue sequence will be passed to that handler. When using multiple running applications on the same subsystem number, the startRange and endRange arguments should be modified to divide the dialogue IDs between them. For example, when using two applications the values are:

Application 1	Application 2
startRange = 0	startRange = 4096
endRange = 4095	endRange = 8191

Syntax

```
int sktal_registerTCAPSSNHandler (int localNodeId, int
stackId, int pc, int pc, int ssn,
SK_TCAP_NewDialogHandler h,int *instanceId int connID,
int startRange=0, int endRange=SK_MAX_TCAP_DIALOGS-1);
```

Input Parameters

Argument	Description
localNodeId	local node ID
stackId	stack ID
pc	point code
ssn	Subsystem Number
h	Handler function for new TCAP dialogues. The prototype for new dialogue handler is: <pre>typedef int (*SKTAL_NewDialogHandler)(i nt instanceId, unsigned newDialogId);</pre>
connId	connection ID
startRange	Starting dialogue ID allocated to this application (For the specific SSN)
endRange	Ending dialogue ID allocated to this application (For the specific SSN)

Input/Output Parameters None**Output Parameters** .

Argument	Description
instanceId	Instance for given node/stack/SNN combination

Return Value SKTAL_SUCCESS: If method is successful.

SKTAL return code as defined in Appendix C, *Table C-2, SKTAL Return Codes (C-3)*.

sktal_unregisterTCAPSSNHandler ()

Description This function removes (and terminates) the subsystem identified during registration. All active transactions are terminated by the PreArrangedEnd() method.

Syntax `int sktal_unregisterTCAPSSNHandler (int instanceId);`

Input Parameters

Argument	Description
instanceId	Instance for given node/stack/SSN combination

Input/Output Parameters and Output Parameters None.

Return Value SKTAL_SUCCESS: If method is successful.

SKTAL return code as defined in Appendix C, *Table C-2, SKTAL Return Codes (C-3)*.

sktal_setTCAPDialogueHandler ()

Description Specifies a function to be invoked for incoming messages within a given transaction (identified by the Dialogue ID). The incoming event and the Dialogue ID invoke the handler; the handler should examine the event and invoke either sk_recvTCAPDialogue or sk_recvTCAPComponent.

Syntax

```
int sktal_setTCAPDialogueHandler(int instanceId, unsigned
    dialogueId, SKTAL_TCAP_DialogueHandler handler, void
    *tag);
```

Input Parameters .

Argument	Description
instanceID	Instance for given node/stack/SNN combination
dialogueId	TCAP dialogue ID
handler	TCAP dialogue handler function. The prototype for new dialogue handler is:
	<pre>typedef int (*SKTAL_DialogHandler)(int instanceId, SKTAL_EVENT *ev, void *tag);</pre>

Input/Output Parameters and Output Parameters None

Return Value SKTAL_SUCCESS: If method is successful.

SKTAL return code as defined in Appendix C, *Table C-2, SKTAL Return Codes (C-3)*.

sktal_clearTCAPDialogHandler()

Description The user must remove the transaction handler once the transaction has been completed (by receiving TC-U-ABORT, TC-P-ABORT, TC-END, or TC-UNI, or sending TC-U-ABORT, TC-END, or TC-UNI). It is imperative to NOT take this step until the last message for a transaction has been sent or received.

Syntax

```
int sktal_clearTCAPDialogueHandler(int instanceId,  
                                  unsigned dialogueId);
```

Input Parameters

Argument	Description
instanceID	Instance for given node/stack/SNN combination
dialogueId	TCAP dialogue ID

Input/Output Parameters and Output Parameters

None.

Return Value SKTAL_SUCCESS: If method is successful.

SKTAL return code as defined in Appendix C, *Table C-2, SKTAL Return Codes (C-3)*.

sktal_allocateTCAPDialogID()

Description This function allocates TCAP dialogue ID.

Syntax

```
int sktal_allocateTCAPDialogID(int instanceId, unsigned *did);
```

Input Parameters

Argument	Description
instanceID	Instance for given node/stack/SNN combination
*did	Pointer to the TCAP dialogue.

Input/Output Parameters None

Output Parameters

Argument	Description
dialogueId	TCAP dialogue ID

Return Value SKTAL_SUCCESS: If method is successful.

SKTAL return code as defined in Appendix C, *Table C-2, SKTAL Return Codes (C-3)*.

sktal_sendTCAPDialog()

Description This method sends a TCAP dialogue message.

Syntax

```
int sktal_sendTCAPDialog(int instanceId, unsigned
    dialogueId, SKTAL_DLG *dlg);
```

Input Parameters

Argument	Description
instanceID	Instance for given node/stack/SNN combination
dialogueId	TCAP dialogue ID
dlg	Pointer to the structure containing TCAP dialogue information

For definition See Appendix D, *Table , SKTAL_DLG and SKTAL_CPT Structure Definitions (D-2)*

Input/Output Parameters and Output Parameters None.

Return Value SKTAL_SUCCESS: If method is successful.

SKTAL return code as defined in Appendix C, *Table C-2, SKTAL Return Codes (C-3)*.

sktal_recvTCAPDialog()

Description This method receives TCAP dialogue information from a SKTAL_EVENT.

Syntax

```
int sktal_recvTCAPDialog(int instanceId, SKTAL_EVENT *ev,  
                        unsigned *dialogueId, SKTAL_DLG  
                        *dlg);
```

Input Parameters instanceId: instance for registered node/stack/snn combination.

ev: event containing dialogue information.

The SKTAL_EVENT is defined in Appendix C.

Input/Output Parameters None

Output Parameters

Argument	Description
dialogueId	TCAP dialogue ID
dlg	Pointer to the structure containing TCAP dialogue information

For definition see Appendix D, *Table , SKTAL_DLG and SKTAL_CPT Structure Definitions (D-2)*.

Return Value SKTAL_SUCCESS: If method is successful.

SKTAL return code as defined in Appendix C, *Table C-2, SKTAL Return Codes (C-3)*.

sktal_sendTCAPComponent()

Description This method sends a TCAP component message.

Syntax

```
int sktal_sendTCAPComponent(int instanceId, unsigned
                             dialogueId, SKTAL_CPT *cpt);
```

Input Parameters

Argument	Description
instanceID	Instance for given node/stack/SNN combination
dialogueId	TCAP dialogue ID
cpt	Pointer to the structure containing component information

For definition see Appendix D, *SKTAL_DLG and SKTAL_CPT Structure Definitions (D-2)*.

Input/Output Parameters and Output Parameters None.

Return Value SKTAL_SUCCESS: If method is successful.

SKTAL return code as defined in Appendix C, *Table C-2, SKTAL Return Codes (C-3)*.

sktal_recvTCAPComponent()

Description This method receives a TCAP component information from a SKTAL_EVENT.

Syntax

```
int sktal_recvTCAPComponent(int instanceId,  
                             SKTAL_EVENT *ev,  
                             unsigned *dialogueId, SKTAL_CPT  
                             *cpt);
```

Input Parameters

Argument	Description
instanceID	Instance for given node/stack/SNN combination
ev	Event containing dialogue information

Input/Output Parameters None

Output Parameters

Argument	Description
dialogueId	TCAP dialogue ID
cpt	Pointer to the structure containing component information

For definition see Appendix D, *Table , SKTAL_DLG and SKTAL_CPT Structure Definitions (D-2)*.

Return Value SKTAL_SUCCESS: If method is successful.

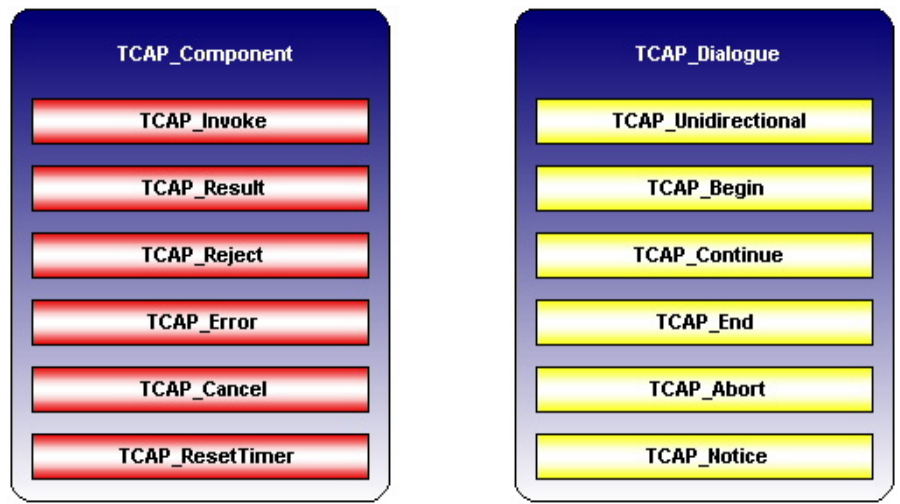
SKTAL return code as defined in Appendix C, *Table C-2, SKTAL Return Codes (C-3)*.

6 SKTAL Parameter Classes

Overview

From an end-user point of view, transaction capabilities are above the network layer of the OSI seven layer reference model. The provision of network layer services to end-users requires communication between TCAP-users at various network nodes; these intra-network communications may in turn be modeled using the OSI Model.

As illustrated in *Figure 6-1, Structure of TCAP Classes (6-2)*, the TCAP is structured in two interfaces for the TCAP components and TCAP dialogs.

Figure 6-1 Structure of TCAP Classes

DG_TCAP_Structure.vsd

This chapter describes the two main classes, TCAP_Component and TCAP_Dialogue, their subclasses, and the methods of each.

Important! SKTAL C++ interface is defined under namespace *sktal*.

Class TCAP_Component

Purpose The purpose of the TCAP_Component interface is to define the base class for all TCAP components. The class, while not abstract, should be considered as such, it is not intended that a user ever create a TCAP_Component class object.

Summary of Methods The following methods are used within the TCAP_Component

```
// Type
SKTAL_USHORT GetComponentType() const;
// Last component
SKTAL_USHORT GetLast() const;
// Invoke ID
void SetInvokeID(const SKTAL_OCTET val);
SKTAL_OCTET GetInvokeID() const;
// Send
static int Send(SKTAL_HANDLE handle,
TCAP_Dialogue* dlg,
                TCAP_Component* cpt);
// Receive
static int Receive(SKTAL_HANDLE handle,
                  SKTAL_Event& ev,
                  TCAP_Dialogue* dlg,
                  TCAP_Component** cpt);
// Print
virtual void Print(std::ostream& os) const;
```

GetComponentType() **Description**

Returns the type of this component.

Syntax

```
SKTAL_USHORT GetComponentType() const;
```

Input Parameters, Input/Output Parameters & Output Parameters

None

Return Value

The type of this component is an unsigned short:

If defined ITU:

- TCPPT_TC_INVOKE
- TCPPT_TC_RESULT_L (where L means last)

- TCPPT_TC_RESULT_NL (where NL means not last)
- TCPPT_TC_U_ERROR
- TCPPT_TC_L_REJECT
- TCPPT_TC_U_REJECT
- TCPPT_TC_R_REJECT
- TCPPT_TC_L_CANCEL
- TCPPT_TC_U_CANCEL
- TCAP_PT-TC_TIMER_RESET

If defined ANSI:

- TCPPT_TC_INVOKE_NL
- TCPPT_TC_INVOKE (Same as invoke last)
- TCPPT_TC-RESULT-L
- TCPPT_TC_RESULT_NL
- TCPPT_TC_ERROR
- TCPPT_TC_REJECT
- TCPPT_TC_CANCEL

GetLast() Description

Returns a Boolean indicating if this is the last component in this PDU

Syntax

```
SKTAL_USHORT GetLast() const;
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

true: True if this instance is the last component in this PDU.

false: If this instance is not the last component in this PDU

SetInvokeID() Description

This method takes an invoke ID from user and converts it into the form expected by TCAP. As invoke IDs are common to all TCAP components, the SetInvokeID method is defined in the TCAP_Component base class.

Syntax

```
void SetInvokeID(const SKTAL_OCTET val);
```

Input Parameters

val: The invoke ID value. Range is 0-255

Input/Output Parameters and Output Parameters

None

Return Value

None

GetInvokeID()**Description**

This method returns the invoke ID of a component. Since invoke IDs are common to all TCAP components, the GetInvokeID method is defined in the TCAP_Component base class. If no invoke ID has been received then the invoke ID is set to zero.

Syntax

```
SKTAL_OCTET GetInvokeID() const;
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

The invoke ID as an unsigned character.

Send()**Description**

This method sends a TCAP message and returns a Boolean indication if the message is successfully sent. This send method is static. The value of SKTAL_SUCCESS is 0.

Syntax

```
static int Send(SKTAL_HANDLE handle,  
               TCAP_Dialogue* dlg,  
               TCAP_Component* cpt);
```

Input Parameters

handle: The transport being sent. For getting the handle use sktal_getTCAPHandle() API

dlg: The TCAP dialogue primitive of which this component is part.

cpt: The TCAP component primitive of which this component is part.

Input/Output Parameters and Output Parameters

None

Return Value

If the message is successfully sent, SKTAL_SUCCESS is returned. Any other return value indicates an error.

Receive()	Description
	This method receives a TCAP message and returns a Boolean indication if the message is successfully received. This Receive method is static.

Syntax

```
static int Receive(SKTAL_HANDLE handle,  
                  SKTAL_Event& ev,  
                  TCAP_Dialogue* dlg,  
                  TCAP_Component** cpt);
```

Input Parameters

handle: The transport that received this event. For getting the handle use sktal_getTCAPHandle() API

ev: The event primitive this component is associated with.

dlg: The dialogue primitive this component is associated with.

cpt: The component primitive this component is associated with.

Input/Output Parameters

None

Output Parameters

cpt: The constructed component

Return Value

SKTAL_SUCCESS: If the component is successfully received. Any other return code indicates an error.

Print() **Description**

This method prints the contents of the component to the output stream that is passed in to the method. This print method is virtual.

Syntax

```
virtual void Print (std::ostream& os) const;
```

Input Parameters

os: The output stream to print the contents to

Input/Output Parameters and Output Parameters

None

Return Value

None

Class TCAP_Invoke : public TCAP_Component

Purpose The purpose of the TCAP_Invoke component is to define the basic interface for applications wishing to invoke remote operations.

Summary of Methods The following methods are used within the TCAP_Invoke component:

```
// Operation
    // If defined(CCITT)
void SetOperation(const SKTAL_LONG code);
void GetOperation(SKTAL_LONG& code) const;
    // If defined(ANSI)
void SetOperation(const bool isNational,
                  const SKTAL_OCTET family,
                  const SKTAL_OCTET code);
void GetOperation(bool& isNational,
                  SKTAL_OCTET& family,
                  SKTAL_OCTET& code) const;

// Parameter
void SetParameter(const SKTAL_OCTET* buf, const int len);
void GetParameter(SKTAL_OCTET* buf, int& len) const;
void SetParameter(const SKTAL_ByteArray& buf);
void GetParameter(SKTAL_ByteArray& buf) const;

// Timeout
void SetTimeOut(const SKTAL_USHORT val);
SKTAL_USHORT GetTimeOut() const;

// Operation class
    // If defined(CCITT)
void SetClass(const SKTAL_USHORT val);
SKTAL_USHORT GetClass() const;

// Linked id (aka "correlation id")
void SetLinkedID(const SKTAL_OCTET val);
SKTAL_OCTET GetLinkedID() const;
void LinkInvoke(const TCAP_Invoke linkTo);
```

SetOperation() **Description**

This method sets the operation field of an invoke component. The signature and behavior of this method is different depending on the standard used. For national use, the values of the operation family are defined in Appendix C.

Syntax

If defined ITU:

```
void SetOperation (const SKTAL_LONG code);
```

If defined ANSI:

```
void SetOperation (const bool isNational,
                  const SKTAL_OCTET family,
                  const SKTAL_OCTET code);
```

Input Parameters (ITU)

code: The operation code

Input Parameters (ANSI)

isNational: indicates whether this operation is national or private

family: the operation family

Input/Output Parameters and Output Parameters

None

Return Value

None

GetOperation()**Description**

This method gets the operation field of an invoke component. The signature and behavior of this method is different depending on the standard used. If the operation is not present, an error code is returned.

Syntax

If defined ITU:

```
void GetOperation(SKTAL_LONG& code) const;
```

If defined ANSI:

```
void GetOperation(bool& isNational,
                  SKTAL_OCTET& family,
                  SKTAL_OCTET& code) const;
```

Input Parameters and Input/Output Parameters

None

Output Parameters (ITU)

code: The operation code

Output Parameters (ANSI):

isNational: indicates whether this operation is national or private

family: the operation family

code: the operation code

Return Value

None

SetParameter()**Description**

This method copies a user-defined parameter into the invoke component. If an invoke does not have User Payload, the user need not call this method; the parameter is assumed to not be present unless this method is called with a non-zero length. The default space for the "parameter.". Parameter is 256 bytes.

Syntax

```
void SetParameter(const SKTAL_OCTET* buf, const int len);
```

```
void SetParameter(const SKTAL_ByteArray& buf);
```

Input Parameters

buf: a buffer containing the parameter to copy into this component

len: the length of the parameter in the buffer

Input/Output Parameters

None

Output Parameters

None

Return Value

None

GetParameter() **Description**

This method copies the parameter in an invoke component into a user supplied buffer. The len parameter will become an in/out parameter and should be set to the maximum size of the buffer to copy to; this method will then check to make sure the parameter to copy is smaller than the buffer supplied. Users should therefore set the length now to avoid problems in the future. Returning a length of zero indicates that no parameter is present.

Syntax

```
void GetParameter(SKTAL_OCTET* buf, int& len) const;
```

```
void GetParameter (SKTAL_ByteArray& buf) const;
```

Input Parameters

buf: a buffer to copy the parameter into

Input/Output Parameters

None

Output Parameters

len: The length of the parameter in the buffer

Return Value

None

SetTimeout() **Description**

Sets the timeout value for TCAP invokes. (ITU only.)

Syntax

```
void SetTimeout(const SKTAL_USHORT val);
```

Input Parameters

val: The timeout value. Range is 1-500 seconds

Input/Output Parameters and Output Parameters

None

Return Value

None

GetTimeOut()**Description**

Retrieves the timeout value for TCAP invokes. (ITU only.)

Syntax

```
SKTAL_USHORT GetTimeOut() const;
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

The timeout value. Range is 1-500 seconds

SetClass()**Description**

This method sets the operation class for TCAP components. The SetClass method is for ITU TCAP only.

Syntax

```
void SetClass(const SKTAL_USHORT val);
```

Input Parameters

val: a number from 1 to 4 indicating the operation class:

- Class 1 - Both success and failure are reported
- Class 2 - Only failure is reported
- Class 3 - Only success is reported
- Class 4 - Neither success, nor failure is reported

Input/Output Parameters and Output Parameters

None

Return Value

None

GetClass() **Description**

This method retrieves the operation class for TCAP invokes. The GetClass method is for ITU TCAP only.

Syntax

```
SKTAL_USHORT GetClass() const;
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

A number from 1 to 4 indicating the operation class:

- Class 1 - Both success and failure are reported
- Class 2 - Only failure is reported
- Class 3 - Only success is reported
- Class 4 - Neither success, nor failure is reported

SetLinkedID() **Description**

This method sets the Linked ID (ITU) or Correlation ID (ANSI) to the value supplied. The SetLinkedID method is not called if the Linked ID is not relevant. It is assumed that no Linked ID is present unless this method is called.

Syntax

```
void SetLinkedID(const SKTAL_OCTET val);
```

Input Parameters

val: The linked ID value. Range is 0-255

Input/Output Parameters and Output Parameters

None

Return Value

None

HasLinkedID()**Description**

HasLinkId method returns true if linked ID (ITU) or correlation ID (ANSI).is present in invoke operation.

Syntax

```
bool HasLinkedID();
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

true - Linked ID present.

false - Linked ID not present

GetLinkedID()**Description**

This method gets the Linked ID (ITU) or Correlation ID (ANSI) of an invoke, if supplied. If the invoke does not have a Linked ID, a Link ID of zero is returned.

Syntax

```
SKTAL_OCTET GetLinkedID() const;
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

The linked ID as an unsigned character. Range is 0-255

LinkInvoke()**Description**

This method sets the Linked ID (ITU) or Correlation ID (ANSI) to the Invoke ID from the invoke component supplied. The LinkInvoke method is not called if the Linked ID is not relevant. It is assumed that no Linked ID is present unless this method is called.

Syntax

```
void LinkInvoke(const TCAP_Invoke* linkTo);
```

Input Parameters

linkTo: the invoke component to correlate to

Input/Output Parameters and Output Parameters

None

Return Value

None

Class TCAP_Result : public TCAP_Component

Purpose The purpose of the TCAP_Result component is to define the basic interface to remote operation results. This class is used to reply to successful invoke operations.

Summary of Methods The TCAP_Result component includes the following methods:

```
// Operation
//If defined(CCITT)
void SetOperation(const SKTAL_LONG code);
void GetOperation(SKTAL_LONG& code) const;

// parameter
void SetParameter(const SKTAL_OCTET* buf, const int len);
void GetParameter(SKTAL_OCTET* buf, int& len) const;
void SetParameter(const SKTAL_ByteArray& buf);
void GetParameter(SKTAL_ByteArray& buf) const;
```

SetOperation() **Description**

This method sets the operation field of a result component. The SetOperation method is relevant only to ITU TCAP.

Syntax

```
void Set Operation(const SKTAL_LONG code);
```

Input Parameters

code: The operation code.

Input/Output Parameters and Output Parameters

None

Return Value

None

GetOperation() **Description**

This method gets the operation field of an invoke component. The GetOperation method is relevant only to ITU TCAP.

Syntax

```
void GetOperation(SKTAL_LONG& code) const;
```

Input Parameters and Input/Output Parameters

None

Output Parameters

code - The operation code

Return Value

None

SetParameter()**Description**

Copies a user-defined parameter into the invoke component.

Syntax

See class TCAP_Invoke

```
void SetParameter(const SKTAL_OCTET* buf, const int len);
```

```
void SetParameter(const SKTAL_ByteArray& buf);
```

Input Parameters

buf - A buffer containing the parameter to copy into this component

len - The length of the parameter in the buffer.

Input/Output Parameters & Output Parameters

None

Return Value

None

GetParameter()**Description**

Copies a user-defined parameter from the invoke component to a user supplied buffer.

Syntax

```
void GetParameter(SKTAL_OCTET* buf, int& len) const;
```

```
void GetParameter(SKTAL_ByteArray& buf) const;
```

Input Parameters

buf: a buffer containing the parameter to copy into this component

Input/Output Parameters

None

Output Parameters

len - The length of the parameter in the buffer.

Return Value

None

Class TCAP_Error : public TCAP_Component

Purpose The purpose of the TCAP_Error component is to define the basic interface to the remote operation error codes. This class is used to reply to unsuccessful invoke operations

Summary of Methods TCAP_Error component includes the following methods:

```
// Error
// If defined(CCITT)
void SetError(const SKTAL_OCTET code);
void GetError(SKTAL_OCTET& code) const;
//If defined(ANSI)
void SetError(const bool isNational, const SKTAL_OCTET
    code);
void GetError(bool& isNational, SKTAL_OCTET& code) const;

// Parameter
void SetParameter(const SKTAL_OCTET* buf, const int len);
void GetParameter(SKTAL_OCTET* buf, int& len) const;
void SetParameter(const SKTAL_ByteArray& buf);
void GetParameter(SKTAL_ByteArray& buf) const;
```

SetError() Description

This method sets the error code in an error component. The signature of this method varies depending on the standard used.

Syntax

If defined ITU:

```
void SetError(const SKTAL_OCTET code);
```

If defined ANSI:

```
void SetError(const bool isNational, const SKTAL_OCTET
    code);
```

Input Parameters (ITU)

code: The error code

Input Parameters (ANSI)

isNational: an indicator that this code is national or private

code: The error code

Input/Output Parameters and Output Parameters

None

Return Value

None

GetError() **Description**

This method gets the error code in an error component. The signature of this method varies depending on the standard used.

Syntax

If defined ITU:

```
void GetError(SKTAL_OCTET& code) const;
```

If defined ANSI:

```
void GetError(bool& isNational, SKTAL_OCTET& code) const;
```

Input Parameters and Input/Output Parameters

None

Output Parameters (ITU)

code: The error code

Output Parameters (ANSI)

isNational: an indicator that this code is national or private

code: the error code

Return Value

None

SetParameter() **Syntax**

See class TCAP_Invoke

```
void SetParameter(const SKTAL_OCTET* buf, const int len);  
void SetParameter(const SKTAL_ByteArray& buf);
```

GetParameter() **Syntax**

See class TCAP_Invoke

void GetParameter(SKTAL_OCTET* buf, int& len) const;

void GetParameter(SKTAL_ByteArray& buf) const;

Class TCAP_Reject : public TCAP_Component

Purpose The purpose of the TCAP_Reject component is to define the basic interface to invalid remote operation invokes. The remote stack responds with a TCAP_Reject when an application sends a malformed component.

Summary of Methods The TCAP_Reject component includes the following methods:

```
// Problem
// If defined(CCITT)
void SetProblem(const SKTAL_OCTET type, const SKTAL_OCTET
code);
void GetProblem(SKTAL_OCTET& type, SKTAL_OCTET& code)
const;
// If defined(ANSI)
void SetProblem(const SKTAL_OCTET family, const
SKTAL_OCTET code);
void GetProblem(SKTAL_OCTET& family, SKTAL_OCTET& code)
const;
// Parameter
// If defined(ANSI)
void GetParameter(SKTAL_OCTET* buf, int& len) const;
void SetParameter(const SKTAL_OCTET* buf, const int len);
void GetParameter(SKTAL_ByteArray& buf) const;
void SetParameter(const SKTAL_ByteArray& buf);
```

SetProblem() **Description**

This method sets the problem code for a reject component. The signature of this method depends on the standard used. Based on the reject code, it is the responsibility of the application to take the appropriate action. Each reject type has its own set of codes, or reasons for the component being rejected. The reject values and codes are defined Appendix A.

Syntax

If defined as ITU:

```
void SetProblem(const SKTAL_OCTET type, const SKTAL_OCTET
code);
```

Type: the reject type. Possible reject types are defined in Appendix A.
Code: the reject code

If defined as ANSI:

```
void SetProblem(const SKTAL_OCTET family,
```

```
const SKTAL_OCTET code);
```

family: the problem types. Problem types are defined in Appendix A.

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

None

GetProblem() **Description**

This method gets the problem code for a reject component. The signature of this method depends on the standard used. Based on the reject code, it is the responsibility of the application to take the appropriate action. Each reject type has its own set of codes, or reasons for the component being rejected. The reject values and codes are defined in Appendix A.

Syntax

If defined ITU:

```
void GetProblem(SKTAL_OCTET& type, SKTAL_OCTET& code)
    const;
```

Type = the reject type. Possible reject types are defined in Appendix A.
Code = the reject code as defined in Appendix A.

If defined ANSI:

```
void GetProblem(SKTAL_OCTET& family, SKTAL_OCTET& code)
    const;
```

Input Parameters, Input/Output Parameters

None

Output Parameters

family: the problem types. Problem types are defined in Appendix A.

Return Value

None

Each reject type has its own set of codes or reasons for the component being rejected. The reject values and codes are defined in the Appendix sections.

SetParameter() **Description**

This method is applicable for ANSI TCAP. See Class TCAP_Invoke.

Syntax

See class TCAP_Invoke

```
void SetParameter(const SKTAL_OCTET* buf, const int len);
```

```
void SetParameter(const SKTAL_ByteArray& buf);
```

GetParameter() **Description**

This method is applicable for ANSI TCAP. See Class TCAP_Invoke.

Syntax

See class TCAP_Invoke

```
void GetParameter(SKTAL_OCTET* buf, int& len) const;
```

```
void GetParameter(SKTAL_ByteArray& buf) const;
```

Class TCAP_Cancel : public TCAP_Component

Purpose The purpose of the TCAP_Cancel component is an artificial (at least for ANSI) component that is generated when an invoke operation times out. The ANSI version of TCAP does not have this component; instead, IntelliNet generates this component for the user. The TCAP_Cancel component includes the following methods:

GetDialogueID() **Description**

This method returns the context information for a dialogue (that is, its “Dialogue ID”). This information should be copied from the Begin primitive when constructing continue or end dialogues. The user must obtain a Dialogue ID from the stack when initiating a transaction. This is done by calling TCAP_AllocateDialogueID().

Syntax

```
SKTAL_USHORT GetDialogueID() const;
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

The Dialogue ID as an unsigned short.

Class TCAP_ResetTimer : public TCAP_Component

The purpose of the TCAP_ResetTimer component defines the basic interface to reset the timer of an invoked operation. This component is only valid with ITU White Book 97. The TCAP_ResetTimer component includes the following methods:

GetDialogueID() Syntax

See class TCAP_Cancel

SKTAL_USHORT GetDialogueID() const;

Class TCAP_Dialogue

Purpose The purpose of the TCAP_Dialogue interface is to define the base class for all TCAP dialogues. The class, while not abstract, should be considered as such. It is not intended that a user ever create a naked object.

Summary of Methods The TCAP_Dialogue interface includes the following methods:

```
// Dialogue type
SKTAL_USHORT GetDialogueType() const;

// Dialogue ID
void SetDialogueID(const SKTAL_USHORT did);
SKTAL_USHORT GetDialogueID() const;
// Component present
virtual bool IsComponentPresent() const;

// Quality of service
virtual void SetQualityOfService(const SKTAL_OCTET flags,
                                const SKTAL_OCTET priority = 0);

virtual void GetQualityOfService(SKTAL_OCTET& flags,
                                SKTAL_OCTET& priority) const;
// Application context
virtual void SetApplicationContext(const SKTAL_OCTET*
    buf, const int len);
virtual void GetApplicationContext(SKTAL_OCTET* buf, int&
    len) const;
virtual void SetApplicationContext(const SKTAL_ByteArray&
    buf);
virtual void GetApplicationContext(SKTAL_ByteArray& buf)
    const;

// User info
virtual void SetUserInfo(const SKTAL_OCTET* buf, const
    int len);
virtual void GetUserInfo(SKTAL_OCTET* buf, int& len)
    const;
virtual void SetUserInfo(const SKTAL_ByteArray& buf);
virtual void GetUserInfo(SKTAL_ByteArray& buf) const;
// Xcvr checks
virtual bool SendCheck() const;
virtual bool ReceiveCheck();

// Send
static int Send(SKTAL_HANDLE handle, TCAP_Dialogue*dlg);
```

```
// Receive
static int Receive(SKTAL_HANDLE handle,
                  SKTAL_Event&ev,
                  TCAP_Dialogue**dlg);

// Expose the header
const SKTAL_HDR& GetHeader() const;

// Print
virtual void Print(std::ostream& os) const;
```

SetDialogueID()**Description**

This method is used to set the transaction context (or "Dialogue ID") of a transaction.

Syntax

```
void SetDialogueID(const SKTAL_USHORT did);
```

Input Parameters

did: The Dialogue ID of this transaction is an unsigned short

Input/Output Parameters and Output Parameters

None

Return Value

None

GetDialogueType()**Description**

Returns the type of this dialogue.

Syntax

```
SKTAL_USHORT GetDialogueType() const;
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

The type of this dialogue is an unsigned short:

If defined as ANSI:

- TCPPT_TC_UNI

- TCPPT_TC_QUERY_W-PERM
- TCPPT_TC_QUERY_WO_PERM
- TCPPT_TC_RESP
- TCPPT_TC_CONV_W_PERM
- TCPPT_TC_CONV_WO_PERM
- TCPPT_TC_ABOUT

TCPPT_TC_NOTICE

If defined as ITU:

- TCPPT_TC_UNI
- TCPPT_TC_BEGIN
- TCPPT_TC_END
- TCPPT_TC_CONTINUE
- TCPPT_TC_P_ABORT
- TCPPT_TC_U_ABORT
- TCPPT_TC_NOTICE

GetDialogueID() Syntax

See class TCAP_Cancel

```
SKTAL_USHORT GetDialogueID() const;
```

IsComponentPresent() Description

This method is a predicate for determining if an inbound dialogue has components associated with it. This predicate is meaningless for outbound dialogues. The IsComponentPresent method is virtual.

Syntax

```
virtual bool IsComponentPresent() const;
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

true: If a dialogue primitive has associated components

false: If a dialogue primitive has no associated components

SetQualityOfService()**Description**

This method allows the user to set the quality of service (QoS) parameters for a transaction

Syntax

```
virtual void SetQualityOfService(const SKTAL_OCTET flags,  
const SKTAL_OCTET priority = 0);
```

Input Parameters

- flags: Quality of service indicators (See Appendix D for a list of flags)
- priority: If QOS_PRIORITY is set in the flags, this indicates the priority value

Input/Output Parameters and Output Parameters

None

Return Value

None

Important! Altering the QoS parameters is not normally done. ANSI does not support all QoS parameters. This method is virtual.

GetQualityOfService()**Description**

This method allows the user to get the quality of service (QoS) parameters for a transaction. The GetQualityOfService method is a virtual method.

Syntax

```
virtual void GetQualityOfService(SKTAL_OCTET& flags,  
SKTAL_OCTET& priority)
```

Input Parameters and Input/Output Parameters

None

Output Parameters

- flags: Quality of service indicators
- Return Option: QOSI_RET_OPT

- Sequence Control: QOSI_SEQ_CTRL
- SLS key present: QOSI_SLS_KEY
- Message priority octet present: QOSI_PRIORITY
- priority: If QOS_PRIORITY is set in the flags, this indicates the priority value

Return Value

None

SetApplicationContext()**Description**

This method allows the user to set the application context for a transaction. Older versions of ANSI TCAP do not support application context. This method is a virtual method.

Syntax

```
virtual void SetApplicationContext(const SKTAL_OCTET*
    buf,
    const int len);
SetApplicationContext(const SKTAL_ByteArray& buf);
```

Input Parameters

buf: A buffer to copy the application context into

len: The number of bytes copied

Input/Output Parameters and Output Parameters

None

Return Value

None

Older versions of ANSI TCAP do not support application context. This method is virtual.

GetApplicationContext()**Description**

This method allows the user to get the application context for a transaction. The len parameter will become an in/out parameter and should be set to the maximum size of the buffer to copy to. Users should therefore set the length now to avoid problems in the future.

Returning a length of zero indicates that no context is present. Older versions of ANSI TCAP do not support application context. The GetApplicationContext method is a virtual method.

Syntax

```
virtual void GetApplicationContext(SKTAL_OCTET* buf, int&
    len) const;
```

Input Parameters

buf: A buffer to copy the application context into

Input/Output Parameters

None

Output Parameters

len: the number of bytes copied

Return Value

None

```
void GetApplicationContext(SKTAL_ByteArray& buf) const;
```

Description

This method allows the user to get the application context for a transaction.

Input Parameters and Input/Output Parameters

None

Output Parameters

buf: a buffer to copy the application context into

Return Value

None

Important! Older versions of ANSI TCAP do not support application context. This method is virtual.

SetUserInfo() **Description**

This method can be used to set the user information for a transaction. Older versions of ANSI TCAP do not support user information. This method is virtual.

Syntax

```
virtual void SetUserInfo(const SKTAL_OCTET* buf,  
                        const int len);  
virtual void SetUserInfo(const SKTAL_ByteArray& buf);
```

Input Parameters

buf: A pointer to a buffer to copy the user information into

len: The number of bytes copied

Input/Output Parameters and Output Parameters

None

Return Value

None

GetUserInfo() **Description**

This method allows the user to get the user information for a transaction. Older versions of ANSI TCAP do not support user information. This method is virtual. The len parameter will become an in/out parameter and should be set to the maximum size of the buffer to copy to. Users should set the length now to avoid problems. Returning a length of zero indicates that no information is present.

Syntax

```
virtual void GetUserInfo(SKTAL_OCTET* buf, int& len)  
const;
```

Input Parameters

buf: A pointer to a buffer to copy the user information into

Input/Output Parameters

None

Output Parameters

len: The number of bytes copied

Return Value

None

Description

This method allows the user to get the user information for a transaction. Older versions of ANSI TCAP do not support user information. This method is a virtual method.

Syntax

```
virtual void GetUserInfo(SKTAL_ByteArray& buf) const;
```

Input Parameters and Input/Output Parameters

None

Output Parameters

buf: A buffer to copy the user information into

Return Value

None

SendCheck()**Description**

This method allows the user to check if the transaction is proceeding. This method is virtual.

Syntax

```
virtual bool SendCheck() const;
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

true: If the transaction is proceeding

false: If the transaction is not proceeding

ReceiveCheck() **Description**

This method allows the user to check if the information is received.
This method is virtual.

Syntax

```
virtual bool ReceiveCheck();
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

true: If the information is received

false: If the information is not received

Send() **Description**

This method is provided for the user to send TCAP dialogues to the stack. This is the preferred method of dialogue transmission. The Send method is static.

Syntax

```
static int Send(SKTAL_HANDLE handle, TCAP_Dialogue* dlg);
```

Input Parameters

handle - The transport this dialogue is sent. For getting the handle use sktal_getTCAPHandle() API

dlg - A pointer to the dialogue to send

Input/Output Parameters and Output Parameters

None

Return Value

If the message is successfully sent, SKTAL_SUCCESS is returned. Any other return value indicates an error.

Receive() Description

This method is provided for the user to receive TCAP dialogues from the stack. This is the preferred method of dialogue reception. The Receive method is static. This method acts as a constructor for received dialogues.

Syntax

```
static int Receive(SKTAL_HANDLE handle,  
                  SKTAL_Event& ev,  
                  TCAP_Dialogue** dlg);
```

Input Parameters

handle: the transport this dialogue is sent. For getting the handle use sktal_getTCAPHandle().

ev: the event to receive the dialogue from

Input/Output Parameters

None

Output Parameters

dlg - the address of a pointer that is populated with the received dialogue

Return Value

If the message is successfully sent, SKTAL_SUCCESS is returned. Any other return value indicates an error.

GetHeader() Description

This method exposes the SKTAL_HDR part of the received dialogue.

Syntax

```
const SKTAL_HDR& GetHeader() const;
```

Input Parameters and Input/Output Parameters

None

Output Parameters

hdr: the header present with the received dialogue

Return Value

None

Print() **Description**

This method prints the contents of the dialogue to the output stream that is passed in to the method.

Syntax

```
virtual void Print(std::ostream& os) const;
```

Input Parameters

os: the output stream to print the contents to.

Input/Output Parameters and Output Parameters

None

Return Value

None

TCAP_Unidirectional : public TCAP_Dialogue class

Purpose The purpose of the TCAP_Unidirectional message class is to define the interface for one-way transactions. Unidirectional messages are never replied to. This message represents the TCPPT_TC_UNI dialogue type.

Summary of Methods The TCAP_Unidirectional message class includes the following methods:

```
// Xcvt checks
virtual bool SendCheck() const;
virtual bool ReceiveCheck();

// Full SCCP Address (orig)
void SetOrigAddr(const bool isNational,
                 const SKTAL_UINT pointCode,
                 const SKTAL_OCTET ssn);
void SetOrigAddr(const SCCP_Address& addr);
void GetOrigAddr(bool& isNational,
                 SKTAL_UINT& pointCode,
                 SKTAL_OCTET& ssn) const;
void GetOrigAddr(SCCP_Address& addr) const;
// Full SCCP Address (des)
void SetDestAddr(const bool isNational,
                 const SKTAL_UINT pointCode,
                 const SKTAL_OCTET ssn);
void GetDestAddr(SCCP_Address& addr) const;
```

SendCheck() Syntax

See class TCAP_Dialogue

```
virtual bool SendCheck() const;
```

ReceiveCheck() Syntax

See class TCAP_Dialogue

```
virtual bool ReceiveCheck();
```

SetOrigAddr() **Description**

This method sets the origination address for this dialogue primitive from discrete information.

Syntax

```
void SetOrigAddr(const bool isNational,  
                 const SKTAL_UINT pointCode,  
                 const SKTAL_OCTET ssn);  
  
void SetOrigAddr(SKTAL_OCTET& addrInd,  
                 SKTAL_UINT& pointCode,  
                 SKTAL_OCTET& ssn,  
                 SKTAL_OCTET* gttInfo,  
                 SKTAL_USHORT* gttLen) const;  
  
void SetOrigAddr(const SCCP_Address& addr);
```

Input Parameters

- isNational: A Boolean flag indicating if this address uses national or international routing
- pointCode: The Point Code of this address
- ssn: The Subsystem Number of this address
- gttInfo: The information of the address
- gttLen: The length of the address
- (SCCP_ADDR&) addr: The SCCP_ADDR structure to copy into
- (SCCP_Address&) addr: The SCCP_Address object to copy into

Input/Output Parameters and Output Parameters

None

Return Value

None

GetOrigAddr() Description

This method decodes the origination address for this dialogue primitive into discrete information.

Syntax

```
void GetOrigAddr(bool& isNational,
                 SKTAL_UINT& pointCode,
                 SKTAL_OCTET& ssn) const;
void GetOrigAddr(SKTAL_OCTET& addrInd,
                 SKTAL_UINT& pointCode,
                 SKTAL_OCTET& ssn,
                 SKTAL_OCTET* gttInfo,
                 SKTAL_USHORT* gttLen) const;

void GetOrigAddr(SCCP_Address& addr) const;
```

Input Parameters and Input/Output Parameters

None

Output Parameters

isNational: a Boolean flag indicating if this address uses national or international routing

pointCode: the Point Code of this address

ssn: the Subsystem Number of this address

gttInfo: the information of the address

gttLen: the length of the address

(SCCP_ADDR&) addr: the SCCP_ADDR structure to copy into

(SCCP_Address&) addr: the SCCP_Address object to copy into

Return Value

None

SetDestAddr() Description

This method sets the destination address for this dialogue primitive from discrete information

Syntax

```
void SetDestAddr(const bool isNational,
```

```

        const SKTAL_UINT pointCode,
        const SKTAL_OCTET ssn);
void SetDestAddr(SKTAL_OCTET addrInd,
                SKTAL_UINT pointCode,
                SKTAL_OCTT ssn,
                SKTAL_OCTET* gttInfo,
                SKTAL_USHORT* gttLen) const;

void SetDestAddr(const SCCP_Address& addr);

```

Input Parameters

- isNational: a Boolean flag indicating if this address uses national or international routing
- pointCode: the Point Code of this address
- ssn: the Subsystem Number of this address
- gttInfo: the information of the address
- gttLen: the length of the address
- (SCCP_ADDR&) addr: the SCCP_ADDR structure to copy into
- (SCCP_Address&) addr: the SCCP_Address object to copy into

Input/Output Parameters and Output Parameters

None

Return Value

None

GetDestAddr() Description

This method decodes the destination address for this dialogue primitive into discrete information.

Syntax

```

void GetDestAddr(bool& isNational,
                SKTAL_UINT& pointCode,
                SKTAL_OCTET& ssn) const;

void GetDestAddr(SKTAL_OCTET& addrInd,
                SKTAL_UINT& pointCode,
                SKTAL_OCTET& ssn,
                SKTAL_OCTET* gttInfo,
                SKTAL_USHORT* gttLen) const;

void GetDestAddr(SCCP_Address& addr) const;

```

Input Parameters and Input/Output Parameters

None

Output Parameters

- isNational: a Boolean flag indicating if this address uses national or international routing
- pointCode: the Point Code of this address
- ssn: the Subsystem Number of this address
- gttInfo: the information of the address
- gttLen: the length of the address
- (SCCP_ADDR&) addr: the SCCP_ADDR structure to copy into
- (SCCP_Address&) addr: the SCCP_Address object to copy into

Return Value

None

Class TCAP_Begin : public TCAP_Dialogue

Purpose The purpose of the TCAP_Begin message class is to define the interface for beginning TCAP transactions. The class implements TCPPT_TC_BEGIN (ITU) and TCPPT_TC_QUERY_W_PERM/ TCPPT_TC_QUERY_WO_PERM (ANSI) dialogues. This message begins a communication session for remote operations. In ANSI, a Boolean may be passed to this Class Constructor. If this Boolean is true (default value), TC_QUERY_WITH_PERM is sent out, else TC_QUERY_WO_PERM.

Summary of Methods The TCAP_Begin message class includes the following methods:

```
// Xcvr checks
virtual bool SendCheck() const;
virtual bool ReceiveCheck();

// Point code (orig)
void GetOPC(SKTAL_UINT& pointCode) const;

// Point code (dest)
void SetDPC(const SKTAL_UINT pointCode);

// Full SCCP Address (orig)
void SetOrigAddr(const bool isNational,
                 const SKTAL_UINT pointCode,
                 const SKTAL_OCTET ssn);
void SetOrigAddr(const SKTAL_OCTET addrInd,
                 const SKTAL_UINT pointCode,
                 const SKTAL_OCTET ssn,
                 const SKTAL_OCTET* gttInfo,
                 const SKTAL_USHORT gttLen);
void SetOrigAddr(const SCCP_Address& addr);

void GetOrigAddr(bool& isNational,
                 SKTAL_UINT& pointCode,
                 SKTAL_OCTET& ssn) const;
void GetOrigAddr(SKTAL_OCTET& addrInd,
                 SKTAL_UINT& pointCode,
                 SKTAL_OCTET& ssn,
                 SKTAL_OCTET* gttInfo,
                 SKTAL_USHORT* gttLen) const;
void GetOrigAddr(SCCP_Address& addr) const;

// Full SCCP Address (dest)
void SetDestAddr(const bool isNational,
                 const SKTAL_UINT pointCode,
```

```

        const SKTAL_OCTET ssn);
void SetDestAddr(const SKTAL_OCTET addrInd,
        const SKTAL_UINT pointCode,
        const SKTAL_OCTET ssn,
        const SKTAL_OCTET* gttInfo,
        const SKTAL_USHORT gttLen);
void SetDestAddr(const SCCP_Address& addr);

void GetDestAddr(bool& isNational,
        SKTAL_UINT& pointCode,
        SKTAL_OCTET& ssn) const;
void GetDestAddr(SKTAL_OCTET& addrInd,
        GetDestAddr(SCCP_Address& addr) const;

```

SendCheck() Syntax

See class TCAP_Dialogue

```
virtual bool SendCheck() const;
```

ReceiveCheck() See class TCAP_Dialogue**Syntax**

```
virtual bool ReceiveCheck();
```

GetOPC() Description

This method gets the OPC for this dialogue primitive from discrete information. OPC contained in MTP routing label is received from the CSP for TCAP BEGIN/QWP/QWOP messages as a optional parameter. OPC is included by the CSP, only if SCCP CGPA does not contain a point code.

Syntax

```
void GetOPC(SKTAL_UINT& pointCode) const;
```

```
void GetOPC(MTP3_POINT_CODE& pc) const;
```

```
void GetOPC(MTP3_PointCode& pc) const;
```

Input Parameters and Input/Output Parameters

None

Output Parameters

pointCode: the point code of this address

Return Value

None

SetDPC() **Description**

Sets the DPC for this dialogue primitive. This method sets the DPC in the routing label of outgoing TCAP BEGIN/QWP/QWOP messages. A DPC set using the SetDPC method will come into effect only if SCCP CDPA does not contain a point code.

Syntax

```
void SetDPC( const SKTAL_UINT pointCode);
```

Input Parameters

pointCode - the point code of this address

Input/Output Parameters and Output Parameters

None

SetOrigAddr() See class TCAP_Unidirectional**Syntax**

```
void SetOrigAddr(const bool isNational,
                 const SKTAL_UINT pointCode,
                 const SKTAL_OCTET ssn);
```

```
void SetOrigAddr(const SKTAL_OCTET addrInd,
                 const SKTAL_UINT pointCode,
                 const SKTAL_OCTET ssn,
                 const SKTAL_OCTET* gttInfo,
                 const SKTAL_USHORT gttLen);
```

```
void SetOrigAddr(const SCCP_Address& addr);
```

GetOrigAddr() **Syntax**

See class TCAP_Unidirectional

```
void GetOrigAddr(bool& isNational,
                 SKTAL_UINT& pointCode,
                 SKTAL_OCTET& ssn) const;
```

```
void GetOrigAddr(SKTAL_OCTET& addrInd,
                 SKTAL_UINT& pointCode,
                 SKTAL_OCTET& ssn,
                 SKTAL_OCTET* gttInfo,
```

```
        SKTAL_USHORT* gttLen) const;  
void GetOrigAddr(SCCP_Address& addr) const;
```

SetDestAddr() See class TCAP_Unidirectional

Syntax

```
void SetDestAddr(const bool isNational,  
                const SKTAL_UINT pointCode,  
                const SKTAL_OCTET ssn);
```

```
void SetDestAddr(const SKTAL_OCTET addrInd,  
                const SKTAL_UINT pointCode,  
                const SKTAL_OCTET ssn,  
                const SKTAL_OCTET* gttInfo,  
                const SKTAL_USHORT gttLen);
```

```
void SetDestAddr(const SCCP_Address& addr);
```

GetDestAddr() **Syntax**

See class TCAP_Unidirectional

```
void GetDestAddr(bool& isNational,  
                SKTAL_UINT& pointCode,  
                SKTAL_OCTET& ssn) const;
```

```
void GetDestAddr(SKTAL_OCTET& addrInd,  
                SKTAL_UINT& pointCode,  
                SKTAL_OCTET& ssn,  
                SKTAL_OCTET* gttInfo,  
                SKTAL_USHORT* gttLen) const;
```

```
void GetDestAddr(SCCP_Address& addr) const;
```

Class TCAP_Continue : public TCAP_Dialogue

Purpose The purpose of the TCAP_Continue message class is to define the basic interface for continuing TCAP transactions. The class implements both TCPPT_TC_CONTINUE (ITU), and TCPPT_TC_CONV_W_PERM/TCPPT_TC_CONV_WO_PERM (ANSI) dialogues. In ANSI, a Boolean may be passed to this Class Constructor. If this Boolean is true (default value), TC_QUERY_WITH_PERM is sent out, else TC_QUERY_WO_PERM. When the user does not wish to use simple request/response transactions, the TCAP_Continue dialogue is used to exchange components without terminating the transaction.

Summary of Methods The TCAP_Continue message class includes the following methods:

```
// Xcvr checks
virtual bool SendCheck() const;
virtual bool ReceiveCheck();

// Full SCCP Address (orig)
void SetOrigAddr(const bool isNational,
                 const SKTAL_UINT pointCode,
                 const SKTAL_OCTET ssn);
void SetOrigAddr(const SKTAL_OCTET addrInd,
                 const SKTAL_UINT pointCode,
                 const SKTAL_OCTET ssn,
                 const SKTAL_OCTET* gttInfo,
                 const SKTAL_USHORT gttLen);
void SetOrigAddr(const SCCP_Address& addr);

void GetOrigAddr(bool& isNational,
                 SKTAL_UINT& pointCode,
                 SKTAL_OCTET& ssn) const;
void GetOrigAddr(SKTAL_OCTET& addrInd,
                 SKTAL_UINT& pointCode,
                 SKTAL_OCTET& ssn,
                 SKTAL_OCTET* gttInfo,
                 SKTAL_USHORT* gttLen) const;
void GetOrigAddr(SCCP_Address& addr) const;
```

SendCheck() See class TCAP_Dialogue

Syntax

```
virtual bool SendCheck() const;
```

ReceiveCheck() See class TCAP_Dialogue

Syntax

```
virtual bool ReceiveCheck();
```

SetOrigAddr() See class TCAP_Unidirectional

Syntax

```
void SetOrigAddr(const bool isNational,  
                 const SKTAL_UINT pointCode,  
                 const SKTAL_OCTET ssn);
```

```
void SetOrigAddr(const SKTAL_OCTET addrInd,  
                 const SKTAL_UINT pointCode,  
                 const SKTAL_OCTET ssn,  
                 const SKTAL_OCTET* gttInfo,  
                 const SKTAL_USHORT gttLen);
```

```
void SetOrigAddr(const SKTAL_OCTET addrInd,  
                 const MTP3_PointCode& pointCode,  
                 const SKTAL_OCTET ssn,  
                 const SKTAL_ByteArray& gttInfo);
```

```
void SetOrigAddr(const SCCP_ADDR& addr);  
void SetOrigAddr(const SCCP_Address& addr);
```

GetOrigAddr() See class TCAP_Unidirectional

Syntax

```
void GetOrigAddr(bool& isNational,  
                 SKTAL_UINT& pointCode,  
                 SKTAL_OCTET& ssn) const;  
void GetOrigAddr(SKTAL_OCTET& addrInd,  
                 SKTAL_UINT& pointCode,  
                 SKTAL_OCTET& ssn,  
                 SKTAL_OCTET* gttInfo,  
                 SKTAL_USHORT* gttLen) const;  
void GetOrigAddr(SCCP_Address& addr) const;
```

Class TCAP_End: public TCAP_Dialogue

Purpose The purpose of the TCAP_End message class is to define the basic interface for terminating TCAP transactions. The class implements both TCPPT_TC_END (ITU), and TCPPT_TC_RESP (ANSI) dialogues. The user terminates a TCAP transaction through the TCAP_End message.

Summary of Methods The TCAP_End message class includes the following methods:

```
// Prearranged end  
bool IsPreArrangedEnd() const;  
void SetPreArrangedEnd(bool onOff);
```

IsPreArrangedEnd() **Description**

This method returns a Boolean indicating if this is the prearranged end.

Syntax

```
bool IsPreArrangeEnd() const;
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

true: if this is the prearranged end

false: if this is not the prearranged end

SetPreArrangedEnd() **Description**

This method allows the user to set the end message for this dialogue primitive with prearranged end or basic end.

Syntax

```
void SetPreArrangedEnd(bool onOff);
```

Input Parameters

onOff (true false)

Input/Output Parameters and Output Parameters

None

Return Value

None

Class TCAP_Abort : public TCAP_Dialogue

Purpose The purpose of the TCAP_Abort message class is to define the basic interface for aborting TCAP transactions. Note that this class can be user-generated for ITU TCAP, but is stack-generated only with ANSI TCAP.

Summary of Methods The TCAP_Abort message class includes the following methods:

```
// Component present
virtual bool IsComponentPresent() const;
virtual void SetComponentPresent(bool);

// Abort reason
SKTAL_OCTET GetAbortReason() const;
void SetAbortReason(SKTAL_OCTET reason);
void GetAbortInfo(SKTAL_OCTET *buf, int &len);
void SetAbortInfo(SKTAL_OCTET *buf, int &len);
```

IsComponentPresent() **Syntax**

See class TCAP_Dialogue

```
virtual bool IsComponentPresent() const;
```

SetComponentPresent() **Description**

This method can be used to set the components associated with the dialogue. The SetComponentPresent method is a virtual method.

Syntax

```
virtual void SetComponentPresent(bool);
```

Input Parameters and Output Parameters

None

Input/Output Parameters

True: present the component

False: do not present the component

Return Value

None

SetAbortReason()**Description**

This method can be used to set the abort reason of a dialogue

Syntax

```
void SetAbortReason(SKTAL_OCTET reason);
```

Input Parameters

reason: The abort reason

Input/Output Parameters and Output Parameters

None

Return Value

None

GetAbortReason()**Description**

This method returns the abort reason of a dialogue

Syntax

```
SKTAL_OCTET GetAbortReason() const;
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

The abort reason as an unsigned character.

GetAbortInfo()**Description**

This method gets the abort information for SKIM_TC_ABORT for an ANSI transaction type.

Syntax

```
void GetAbortInfo(SKTAL_OCTET *buf, int &len);
```

Input Parameters, Input/Output Parameters

None

Output Parameters

buf: Abort information

len: Information length

SetAbortInfo()**Description**

This method sets the abort information for SKIM_TC_ABORT for an ANSI transaction type.

Syntax

```
void SetAbortInfo(SKTAL_OCTET *buf, int &len);
```

Input Parameters

buf: Abort information

len: Information length

Input/Output Parameters, & Output Parameters

None

ClassTCAP_Notice : public TCAP_Dialogue

Purpose The purpose of the TCAP_Notice message class is to define the basic interface by which TCAP can deliver quality-of-service messages to the user. At the present time, this is implemented only for ITU TCAP, and is never generated by the user.

Summary of Methods The TCAP_Notice message class includes the following methods:

```
// Component present
virtual bool IsComponentPresent() const
virtual void SetComponentPresent(bool)

// Report cause
void SetReportCause(SKTAL_OCTET reason);
SKTAL_OCTET GetReportCause() const;

// Quality of service
virtual void SetQualityOfService(const SKTAL_OCTET flags,
                                const SKTAL_OCTET priority = 0)
virtual void GetQualityOfService(SKTAL_OCTET& flags,
                                SKTAL_OCTET& priority) const;

// Application context
virtual void SetApplicationContext(const SKTAL_OCTET* buf, const int len);
virtual void GetApplicationContext(SKTAL_OCTET* buf, int& len) const;
virtual void SetApplicationContext(const SKTAL_ByteArray& buf);
virtual void GetApplicationContext(SKTAL_ByteArray& buf) const;
// User info
virtual void SetUserInfo(const SKTAL_OCTET* buf, const int len);
virtual void GetUserInfo(SKTAL_OCTET* buf, int& len) const;
virtual void SetUserInfo(const SKTAL_ByteArray& buf);
virtual void GetUserInfo(SKTAL_ByteArray& buf) const;
```

IsComponentPresent() See class TCAP_Dialogue

Syntax

```
virtual bool IsComponentPresent() const;
```

SetComponentPresent() See class TCAP_Abort

Syntax

```
virtual void SetComponentPresent(bool);
```

SetReportCause()**Description**

This method can be used to set the report cause of a dialogue

Syntax

```
void SetReportCause(SKTAL_OCTET reason);
```

Input Parameters

reason: the report cause reason

Input/Output Parameters and Output Parameters

None

Return Value

None

GetReportCause()**Description**

This method returns the report cause of a dialogue

Syntax

```
SKTAL_OCTET GetReportCause() const;
```

Input Parameters, Input/Output Parameters, & Output Parameters

None

Return Value

The report cause as an unsigned character. The following are the report cause values:

- SCCP_RET_NO_TRANS_ADDR_NAT
- SCCP_RET_NO_TRANS_THIS_ADDR
- SCCP_RET_SUBSYS_CONG
- SCCP_RET_SUBSYS_FAIL
- SCCP_RET_UNEQUIPPED_USER
- SCCP_RET_NETWORK_FAIL
- SCCP_RET_NETWORK_CONG
- SCCP_RET_UNQUAL

- SCCP_RET_ERR_IN_TRANSPORT
- SCCP_RET_ERR_IN_LOCAL_PROCESS
- SCCP_RET_ERR_DEST_CANNOT_PERF_REASSEMBLY
- SCCP_RET_ERR_SCCP_FAILURE (ITU Only)
- SCCP_RET_ERR_HOP_COUNT_VIOLATION (ANSI only)
- SCCP_RET_ERR_INV_ISNI_ROUT_REQ (ANSI only)
- SCCP_RET_ERR_UNAUTH_MSG (ANSI only)
- SCCP_RET_ERR_MSG_INCOMPATIBILITY (ANSI only)
- SCCP_RET_ERR_CANNOT_PERFORM_ISNI_ROUTING (ANSI only)
- SCCP_RET_ERR_REDUNDANT_ISNI_ROUTING_INFO (ANSI only)
- SCCP_RET_ERR_UNABLE_TO_IDENTIFY_ISNI_INFO (ANSI only)

SetQualityOfService() See class TCAP_Dialogue

Syntax

```
virtual void SetQualityOfService(const SKTAL_OCTET flags,  
                                const SKTAL_OCTET priority = 0);
```

GetQualityOfService() See class TCAP_Dialogue

Syntax

```
virtual void GetQualityOfService(SKTAL_OCTET& flags,  
                                SKTAL_OCTET& priority) const;
```

SetApplicationContext() See class TCAP_Dialogue

Syntax

```
virtual void SetApplicationContext(const SKTAL_OCTET*  
    buf, const int len);
```

```
virtual void SetApplicationContext(const SKTAL_ByteArray&  
    buf);
```

GetApplicationContext() See class TCAP_Dialogue

Syntax

```
virtual void GetApplicationContext(SKTAL_OCTET* buf, int&  
    len) const;
```

```
virtual void GetApplicationContext(SKTAL_ByteArray& buf)
    const;
```

SetUserInfo() See class TCAP_Dialogue

Syntax

```
virtual void SetUserInfo(const SKTAL_OCTET* buf, const
    int len);
```

```
virtual void SetUserInfo(const SKTAL_ByteArray& buf);
```

GetUserInfo() See class TCAP_Dialogue

Syntax

```
virtual void GetUserInfo(SKTAL_OCTET* buf, int& len)
    const;
```

```
virtual void GetUserInfo(SKTAL_ByteArray& buf) const;
```

7 SKTAL Sample Application

Overview

This chapter describes a sample application using SKTAL API. The various steps involved are explained in the following sections.

Step One - Creating a connection with LLC using SwitchKit APIs

Purpose The API function described in this section allows you to create a connection with the LLC using SwitchKit APIs.

Important! Before you begin using this sample application, you must set environment variables. See *Environment Variables for Sample Applications (4-2)*.

Application Code

```
connect()
{
    int ret;

    /* create the connection */
    connection = skts_createConnectionWithID(CONN_ID,
    CONN_LABEL, CONN_NAME,
    SKTAL_FALSE,
    PRI_IP, PRI_PORT,
    BACK_IP,
    BACK_PORT);
    if (ret == 0)
    {
        fprintf(stderr, "Error connecting to switch\n");
        return (SKTAL_EINVALIDARGS);
    }

    return (SKTAL_SUCCESS);
}
```

Step Two - Initializing SKTAL and registering SSN

Purpose The API function described in this section initializes TAL and registers with LLC for use of a particular stack/SSN.

Application Code

```
// Initialize TAL layer. TAL initialization is done once
// at application startup.
ret = sktal_initializeTCAP()
if (ret != SK_OK)
{
    fprintf(stderr, "Error initializing TCAP: %s\n",
sk_errorText(ret));
    skts_destroyConnection(CONN_ID);
    return (SKTAL_EINVALIDARGS);
}

// register SSN with TCAP

// LOCAL_NODE - value created during configuration
// STACK_ID - Stack Id for the stack instance created
// during
// configuration.
// CONN_ID - Same as created in step one.
// LOCAL_PC - Local Point Code for stack instance.
// LOCAL_SSN - local subsystem number for stack
// instance.
// newDialogue - Default Handler for handling new
// TCAP Transaction messages.
// instance - instance value returned after
// registration is successful.
// This instance value is used while sending
// and receiving TCAP
// dialogues and components.
ret = sktal_registerTCAPSSNHandler (LOCAL_NODE,
STACK_ID,
LOCAL_PC, LOCAL_SSN,
newDialogue,
&instance, CONN_ID);
if (ret != SK_OK)
{
    fprintf(stderr, "Error registering handler: %s\n",
sk_errorText(ret));
    skts_destroyConnection(CONN_ID);
}
```

Step Three - Sending a TCAP Dialogue and Component

Purpose The API function described in this section allows you to send a dialog and component through SwitchKit to the SS7 network.

Application Code

```
// allocate dialogue id. instance value is returned during
// registration.
ret = sk_allocateTCAPDialogueID(instance, &did);
if (ret != SK_OK)
{
    printf("Failed to allocate dialog id: %d\n", ret);
}

// Create TCAP Begin dialogue object
sktal::TCAP_Begin begin;

// populate originating and destination
// point codes.
begin.SetOPC(LOCAL_PC);
begin.SetDPC(REMOTE_PC);

// populate originating SCCP address
begin.SetOrigAddr(true, LOCAL_PC, LOCAL_SSN);

// Populate destination SCCP address.
begin.SetOrigAddr(true, LOCAL_PC, LOCAL_SSN);

// Populate destination SCCP address.
begin.SetDestAddr(true, REMOTE_PC, REMOTE_SSN);

// Set dialogue id allocated.
begin.SetDialogueID(did);

// Create TCAP invoke component instance.
sktal::TCAP_Invoke invoke;

invoke.SetOperation(true, 1, 1);

// Set invoke id.
invoke.SetInvokeID(1);

// set invoke timeout.
invoke.SetTimeOut(20);

// Set encoded parameters of TCAP User part
// API's like ANSI-41, MAP, CAMEL etc.
```

```
        invoke.SetParameter(encodedParam);

        // Send TCAP Component
        ret =
sktal::TCAP_Component::Send(sktal_getTCAPHandle(instance),
                                &begin, &invoke);
        if (ret != SKTAL_SUCCESS)
        {
            printf("Failed to send component: %d\n", ret);
        }
        else
        {
            // Send TCAP dialogue.
            ret =
sktal::TCAP_Dialogue::Send(sktal_getTCAPHandle(instance),
                                &begin);

            if (ret != SKTAL_SUCCESS)
            {
                printf("Failed to send dialogue: %d\n", ret);
            }
        }
    }
```

Step Four - Receiving a TCAP Transaction

Purpose The API function described in this section allows you to receive a TCAP transaction from the SS7 network via SwitchKit.

Application Code Transactions are received via dialogue handler and registered with SKTAL. Sample implementation for dialogue handler is shown in the following code excerpt.

```
i/*
 * new dialogue handler
 */
extern "C" int
newDialogue(int instance, unsigned newDID)
{
    printf("NEW DIALOGUE: %08x\n", newDID);

    // register a message handler for new dialogue.
    return sktal_setTCAPDialogHandler(instance, newDID,
                                      onMessage, NULL);
}

/*
 * Message handler.
 */
extern "C" int
onMessage(int instance, SKTAL_EVENT *ev, void *tag)
{
    static sktal::TCAP_Dialogue *dlg = NULL;
    SKTAL_CPT cpt;
    sktal::Event event;
    unsigned did;

    printf("ON MESSAGE\n");
    if (TCAP_MSG_TYPE(ev) == SKTAL_TCAP_DLG)
    {
        event = ev;

        sktal::TCAP_Dialogue::Receive(sktal_getTCAPHandle(instance),
                                      event, &dlg);

        if (dlg)
        {
            dlg->Print(std::cout);
        }
    }
}
```

```
else if (TCAP_MSG_TYPE(ev) == SKTAL_TCAP_CPT)
{
    printf("Got component\n");
    sktal::TCAP_Component *result;

    event = ev;

    sktal::TCAP_Component::Receive(sktal_getTCAPHandle(instance),
                                   event, dlg, &result);

    result->Print(std::cout);

    delete result;
    delete dlg;
    dlg = NULL;
}
else
{
    fprintf(stderr, "Unknown message type: %d\n",
TCAP_MSG_TYPE(ev));
}

printf("RETURN FROM USER HANDLER\n");

return (SK_OK);
}
}
```

Step Five - Terminating SKTAL and Disconnection

Purpose The API function described in this section is used to terminate a SwitchKit TCAP Abstraction Layer and disconnect from the LLC.

Application Code

```
// Unregister SSN instance.
if (instance >= 0)
{
    sktal_unregisterTCAPSSNHandler (instance);
    instance = -1;
}

// Terminate SKTAL
stalk_terminateTCAP();

// disconnect
if (connection)
{
    skts_destroyConnection(CONN_ID);
    connection = NULL;
}
```

A Appendix - TCAP Codes

Overview

Refer to the following tables in this appendix for the TCAP codes.

ANSI TCAP Codes

Purpose Refer to the following tables for the ANSI TCAP codes.

Table A-1 ANSI TCAP Operation Family

Name	Value	Description
TCPN_OF_NOT_USED	0x0	
TCPN_OF_REPLY_REQUIRED	0x80	Operation family reply required.
TCPN_OF_PARAMETER	0x01	Operation family parameter.
TCPN_OF_CHARGING	0x02	Operation family charging.
TCPN_OF_PROV_INST	0x03	Operation family provisioning instruction.
TCPN_OF_CONN_CTRL	0x04	Operation family connection control.
TCPN_OF_CALLER_INT	0x05	Operation family caller inter.
TCPN_OF_SEND_NOT	0x06	Operation family send notification.
TCPN_OF_NET_MAN	0x07	Operation family network management.
TCPN_OF_PROCEDURAL	0x08	Operation family procedural.
TCPN_OF_IS41	0x09	Operation family IS41.
TCPN_OF_MISC	0xFE	Operation family Misc.
TCPN_OF_RESERVED	0xFF	

Table A-2 ANSI TCAP Operation Specifier

Name	Value	Description
TCPN_OS_PROV_VAL	0x01	Operation specifier provision value.
TCPN_OS_SET_VAL	0x02	Operation specifier set value.
TCPN_OS_BILL_CALL	0x01	Operation specifier bill call.
TCPN_OS_START	0x01	Operation specifier start.
TCPN_OS_ASSIST	0x02	Operation specifier start.

Name	Value	Description
TCPN_OS_CONN	0x01	Operation specifier connection.
TCPN_OS_TEMP_CONN	0x02	Operation specifier temporary connection.
TCPN_OS_DISCONN	0x03	Operation specifier disconnect.
TCPN_OS_FWD_DISCONN	0x04	Operation specifier forward disconnect.
TCPN_OS_PLAY_A	0x01	Operation specifier play announcement.
TCPN_OS_PLAY_A_CD	0x02	Operation specifier play announcement.
TCPN_OS_AUTO_CALL_GAP	0x01	Operation specifier automatic call gap.
TCPN_OS_TEMP_HO	0x01	

Table A-3 ANSI TCAP P-Abort Reason

Name	Value	Description
TCPABT_REASON_UNREC_PACK_TYPE	0x01	Unrecognized package type.
TCPABT_REASON_INCORRECT_TRANS_PORT	0x02	Incorrect Transaction portion.
TCPABT_REASON_BADLY_STRUCT_TRANS_PORT	0x03	Badly structured transaction portion.
TCPABT_REASON_UNREC_TRANS_ID	0x04	Unrecognized transaction Id.
TCPABT_REASON_PERM_TO_RELEASE	0x05	Permission To release.
TCPABT_REASON_RES_UNAVAIL	0x06	Resource unavailable.

Table A-4 ANSI TCAP Error Codes

Name	Value	Description
TCPERR_UNEX_COMP_SEQ	0x01	Unexpected component sequence
TCPERR_UNEX_DATA_VAL	0x02	Unexpected data value
TCPERR_UNAV_RESOURCE	0x03	Unavailable resource
TCPERR_MISSING_REC	0x04	Missing record
TCPERR_REPLY_OVERDUE	0x05	Reply overdue

Name	Value	Description
TCPERR_DATA_UNAV	0x06	Data unavailable
TCPERR_TSK_RE	0x07	
TCPERR_Q_FULL	0x08	Queue full
TCPERR_NO_Q	0x09	No queue
TCPERR_TMR_EX	0x0A	Timer expiry
TCPERR_DAT_EX	0x0B	Data already exists
TCPERR_UNAUTH	0x0C	Unauthorized Request
TCPERR_NOT_QD	0x0D	Not queued
TCPERR_UAS_DN	0x0E	Unassigned DN
TCPERR_SPARE	0x0F	Spare
TCPERR_NOT_AV	0x10	Not Available
TCPERR_VMSR_E	0x11	VMSR error

Table A-5 ANSI TCAP Problem Code Specifier

Name	Value	Description
TCPPROB_GENERAL	0x01	General Problem.
TCPPROB_INVOKE	0x02	Invoke Problem.
TCPPROB_RETURN_RES	0x03	Return Result Problem.
TCPPROB_RETURN_ERR	0x04	Return Error Problem.
TCPPROB_TRANS_PORTION	0x05	Transaction Portion Problem.

Table A-6 ANSI TCAP Problem Codes

Name	Value	Description
TCPPROB_SPEC_GEN_UNREC_COMP	0x01	General problem unrecognized component
TCPPROB_SPEC_GEN_INCORRECT_COMP	0x02	General problem incorrect component

Name	Value	Description
TCPPROB_SPEC_GEN_BADLY_STRUCT_COMP	0x03	General problem badly structured component
TCPPROB_SPEC_INV_DUPLICATE_INV_ID	0x01	Invoke problem duplicate invoke id
TCPPROB_SPEC_INV_UNREC_OP_CODE	0x02	Invoke problem unrecognized operation code
TCPPROB_SPEC_INV_INCORRECT_PARAM	0x03	Invoke problem incorrect parameter
TCPPROB_SPEC_INV_UNREC_COREL_ID	0x04	Invoke problem unknown correlation id.
TCPPROB_SPEC_RES_UNREC_COREL_ID	0x01	Result problem unrecognized correlation id.
TCPPROB_SPEC_RES_UNEXPECTED_RET_RES	0x02	Result problem unexpected return result
TCPPROB_SPEC_RES_INCORRECT_PARAM	0x03	Result problem incorrect parameter
TCPPROB_SPEC_ERR_UNREC_COREL_ID	0x01	Error problem unrecognized correlation id
TCPPROB_SPEC_ERR_UNEXPECTED_RET_ERROR	0x02	Error problem unexpected return error
TCPPROB_SPEC_ERR_UNREC_ERROR	0x03	Error problem unrecognized error
TCPPROB_SPEC_ERR_UNEXPECTED_ERROR	0x04	Error problem unexpected error
TCPPROB_SPEC_ERR_INCORRECT_PARAM	0x05	Error incorrect parameter
TCPPROB_SPEC_TRANS_UNREC_PACK_TYPE	0x01	Unrecognized package type
TCPPROB_SPEC_TRANS_INCORRECT_TRANS_PORT	0x02	Incorrect transaction Portion
TCPPROB_SPEC_TRANS_BADLY_STRUCT_TRANS_PORT	0x03	Badly structured transaction
TCPPROB_SPEC_TRANS_UNREC_TRANS_ID	0x04	Unrecognized transaction ID
TCPPROB_SPEC_TRANS_PERM_TO_RELEASE	0x05	Permission To release
TCPPROB_SPEC_TRANS_RES_UNAVAIL	0x06	Resource unavailable

ITU TCAP Codes

Purpose Refer to the following tables for the CCITT TCAP codes.

Table A-7 ITU TCAP User Abort Reason

Name	Value	Description
TCPUABT_AC_NOT_SUP	0x01	Application context not supported.
TCPUABT_USER_DEFINED	0x02	Reason user defined.
TCPUABT_DLG_REFUSED	0x03	Dialogue refused.

Table A-8 TU TCAP P-Abort Reason

I

Name	Value	Description
TCPABT_ABNORMAL_DLG	126	Abnormal dialogue.
TCPABT_NO_COMMON_DLG	127	No common dialog.
TCPABT_REASON_UNREC_MSG_TYPE	0x00	Unrecognized message type.
TCPABT_REASON_UNREC_TRANS_ID	0x01	Unrecognized transaction Id.
TCPABT_REASON_BADLY_STRUCT_TRANS_PORT	0x02	Badly structured transaction portion.
TCPABT_REASON_INCORRECT_TRANS_PORT	0x03	Incorrect transaction portion.
TCPABT_REASON_RES_UNAVAIL	0x04	Resource unavailable.

Table A-9 ITU TCAP Reject Problem Types

Name	Value	Description
TCPPROB_GENERAL	0x00	General problem.
TCPPROB_INVOKE	0x01	Invoke problem.
TCPPROB_RETURN_RES	0x02	Return Result problem.
TCPPROB_RETURN_ERR	0x03	Return Error problem

Table A-10 ITU TCAP Reject Problem Codes

Name	Value	Description
TCPPROB_SPEC_GEN_UNREC_COMP	0x00	General problem unrecognized component.
TCPPROB_SPEC_GEN_MISTYPED_COMP	0x01	General problem mistyped component.
TCPPROB_SPEC_GEN_BADLY_STRUCT_COMP	0x02	General problem badly structured component.
TCPPROB_SPEC_INV_DUPLICATE_INV_ID	0x00	Invoke problem duplicate invoke id.
TCPPROB_SPEC_INV_UNREC_OP_CODE	0x01	Invoke problem-unrecognized operation code.
TCPPROB_SPEC_INV_MISTYPED_PARAM	0x02	Invoke problem mistyped parameter.
TCPPROB_SPEC_INV_RESOURCE_LIMIT	0x03	Invoke problem resource limitation.
TCPPROB_SPEC_INV_INITIATE_RELEASE	0x04	Invoke problem initiate release.
TCPPROB_SPEC_INV_UNREC_LINKED_ID	0x05	Invoke problem-unrecognized linkedid.
TCPPROB_SPEC_INV_UNEXPECTED_LINK_RESP	0x06	Invoke problem unexpected linked response.
TCPPROB_SPEC_INV_UNEXPECTED_LINKED_OP	0x07	Invoke problem unexpected linked operation.
TCPPROB_SPEC_RES_UNREC_INVOKE_ID	0x00	Result problem unrecognized invoke id.
TCPPROB_SPEC_RES_UNEXPECTED_RET_RES	0x01	Result problem unexpected return result.
TCPPROB_SPEC_RES_MISTYPED_PARAM	0x02	Result problem mistyped parameter.
TCPPROB_SPEC_ERR_UNREC_INVOKE_ID	0x00	Error problem unrecognized invoke id.
TCPPROB_SPEC_ERR_UNEXPECTED_RET_ERROR	0x01	Error problem unexpected return error.
TCPPROB_SPEC_ERR_UNREC_ERROR	0x02	Error problem unrecognized error.
TCPPROB_SPEC_ERR_UNEXPECTED_ERROR	0x03	Error problem unexpected error.
TCPPROB_SPEC_ERR_MISTYPED_PARAM	0x04	Error problem mistyped parameter.

B Appendix - SKIM Data Types and Codes

Overview

This appendix provides information on SKIM data types and error codes.

SKIM Data Types and Codes

Purpose This section provides information on SKIM data types and error codes.

Table B-1 SKIM Data Types

Data Types	System Data Type
SKIM_BOOLEAN	unsigned int.
SKIM_OCTET	unsigned char.
SKIM_USHORT	unsigned short.
SKIM_UINT	unsigned int.
SKIM_ULONG	unsigned long.
SKIM_CHAR	char
SKIM_SHORT	short
SKIM_INT	int
SKIM_LONG	long

Table B-2 SKIM Return Codes

Error Codes	Values
SKIM_SUCCESS	0
SKIM_FALSE	0
SKIM_TRUE	0x1
SKIM_ENOMEM	-1
SKIM_ERCVFAIL	-6
SKIM_ENOMSG	-8
SKIM_ESENDFAIL	-9
SKIM_ETCAPMSGSENDFAIL	-13
SKIM_BADTCAPMESSAGE	-16
SKIM_ETOOMANYDIALOGS	-18
SKIM_ENOINVID	-20
SKIM_ENOMUTEX	-24

Error Codes	Values
SKIM_EBADMUTEX	-25
SKIM_EINVALDARGS	-39
SKIM_ENOLICENSE	-45
SKIM_EPROTERR	-46
SKIM_EOVERFLOW	-49
SKIM_EINITFAIL	-52
SKIM_EINUSE	-55
SKIM_EDESTPROHIBIT	-56
SKIM_EINVPTYPE	-57
SKIM_EINVOPFAM	-58
SKIM_EINVOPSPEC	-59
SKIM_EINVLEN	-60
SKIM_ENULLPTR	-61
SKIM_EBADSTATE	-64
SKIM_ENOTFOUND	-65
SKIM_EASNENCODE	-66
SKIM_EASNDECODE	-67
SKIM_EINVOPC	-72
SKIM_EINVDPDPC	-73
SKIM_EINVINITSTATE	-86

C Appendix - SKTAL Data Types and Codes

Overview

This appendix provides information on SKTAL data types and error codes.

SKTAL Data Types

Purpose This section provides information on SKTAL data types and error codes.

Table C-1 SKTAL Data Types

Data Types	Description
SKTAL_BOOLEAN	unsigned int.
SKTAL_OCTET	unsigned char.
SKTAL_USHORT	unsigned short.
SKTAL_UINT	unsigned int.
SKTAL_ULONG	unsigned long.
SKTAL_CHAR	char
SKTAL_SHORT	short
SKTAL_INT	int
SKTAL_LONG	long
SKTAL_HANDLE	void
SKTAL_POINTER	char
SKTAL_ByteArray	vector <SKTAL_OCTET>

SKTAL Event

```
typedef struct
{
    SKTAL_USHORT len; /* length of event data */
    SKTAL_USHORT src; /* not Used */
    ITS_OCTET* data; /* Event data containing TCAP
    Dialogue or Component
    * Information */
}
```

To distinguish between TCAP Dialogue events and TCAP Component Events use the following macros:

- `#define TCAP_MSG_TYPE(ev) ((ev)->data[0])`
- `#define SKTAL_TCAP_DLG 1`
- `#define SKTAL_TCAP_CPT 2`

Table C-2 SKTAL Return Codes

Code	Value
SKTAL_SUCCESS	0
SKTAL_FALSE	0
SKTAL_TRUE	0x1
SKTAL_BITS_PER_BYTE	8
SKTAL_ENOMEM	-1
SKTAL_ERCVFAIL	-6
SKTAL_ENOMSG	-8
SKTAL_ESENDFAIL	-9
SKTAL_ETCAPMSGSENDFAIL	-13
SKTAL_BADTCAPMESSAGE	-16
SKTAL_ETOOMANYDIALOGS	-18
SKTAL_ENOINVID	-20
SKTAL_ENOMUTEX	-24
SKTAL_EBADMUTEX	-25
SKTAL_EINVALIDARGS	-39
SKTAL_ENOLICENSE	-45
SKTAL_EPROTERR	-46
SKTAL_EOVERFLOW	-49
SKTAL_EINITFAIL	-52
SKTAL_EINUSE	-55
SKTAL_EDESTPROHIBIT	-56
SKTAL_EINVPTYPE	-57
SKTAL_EINVOPFAM	-58
SKTAL_EINVOPSPEC	-59
SKTAL_EINVLEN	-60
SKTAL_ENULLPTR	-61
SKTAL_EBADSTATE	-64
SKTAL_ENOTFOUND	-65

Code	Value
SKTAL_EASNENCODE	-66
SKTAL_EASNDECODE	-67
SKTAL_EINVOPC	-72
SKTAL_EINVDPC	-73
SKTAL_EINVINITSTATE	-86
SKTAL_ALREADY_REGISTERED	-121
SKTAL_STACK_IN_USE	-122
SKTAL_NO_MORE_STACKS	-123
SKTAL_NO_MORE_DIDS	-124
SKTAL_INVALID_ARG	-125
SKTAL_BAD_MUTEX	-126

D Appendix - SKTAL Dialog and Component Structure Definitions

Overview

This appendix provides information on SKTAL dialog and component structure definitions.

SKTAL_DLG and SKTAL_CPT Structure Definitions

```

/*
 * Quality of service indicator octet definitions:
 * (To select more than one option OR together options)
 */
#define QOSI_RET_OPT (0x01) /* Return Option */
#define QOSI_SEQ_CTRL (0x02) /* Sequence Control */
#define QOSI_SLS_KEY (0x04) /* SLS key present */
#define QOSI_PRIORITY (0x08) /* Message priority octet
    present */
#define QOSI_NETWK_IND (0x10) /* Use provided network
    indicator */
#define QOSI_PROT_VER (0x20) /* Force inclusion of the
    TCAP ver */
/*
 * TCAP_PN_TERMINATION Values:
 */
#define TCAP_END_BASIC (0) /* Basic end */
#define TCAP_END_PREARRANGED (1) /* Pre-arranged end */
/*
 * TCAP_PN_CPT_PRESENT Values:
 */
#define TCAP_NO_CPT (0) /* No component(s) present */
#define TCAP_CPT_PRESENT (1) /* Component(s) present */
/*
 * TCAP_PN_LAST_CPT Values:
 */
#define TCAP_MORE_CPTS (0) /* More component(s) to follow
    */
#define TCAP_LAST_CPT (1) /* This is the last component
    */
#define TCPEND_BASIC TCAP_END_BASIC
#define TCPEND_PREARRANGED TCAP_END_PREARRANGED
#define TCP_NO_CPT TCAP_NO_CPT
#define TCP_CPT_PRESENT TCAP_CPT_PRESENT
#define TCP_MORE_CPTS TCAP_MORE_CPTS
#define TCP_LAST_CPT TCAP_LAST_CPT
/* Size values for Manual TCAP Parser */
#define MAX_TCAP_CPT_SIZE (304)
#define MAX_NO_OF_TCAP_CPTS (4)
#define MAX_TCAP_DLG_SIZE (336)
#define MAX_TCAP_TRANS_SIZE \
    (MAX_TCAP_DLG_SIZE + (MAX_TCAP_CPT_SIZE *
    MAX_NO_OF_TCAP_CPTS))
/
*****
*****

```

```

*
*
*      Structure definitions for Component Primitives
*
*
*
*
*
*****
*****/
/*
* Definitions for buffer sizes in the 'C' structured
* representation of TCAP protocol primitives.
*
* Each value must allow space for the tag, length and
* associated data to be stored.
* The user may need to change the values given in
* order to support larger parameters or to reduce
* the size of the structures if it is known that
* certain parameters lengths will never be exceeded.
*/
#define IV_SIZE (4) /* space for 'invoke_id' parameter */
#define OP_SIZE (32) /* space for 'operation' parameter
*/
#define PR_SIZE (256) /* space for 'parameter' parameter
*/
#define ER_SIZE (32) /* space for 'error' parameter */
#define PB_SIZE (16) /* space for 'problem' parameter */
#define AC_SIZE (64) /* space for 'ac_name' parameter */
#define UI_SIZE (256) /* space for 'user_info' parameter
*/
#define AB_SIZE (256) /* space for 'abt_info' parameter
*/
/*
* Substructures for Components
*/
typedef struct cpt_inv_id
{
    SKTAL_USHORT len;
    SKTAL_OCTET data [IV_SIZE];
}

CPT_INV_ID;
typedef struct cpt_op
{
    SKTAL_USHORT len;
    SKTAL_OCTET data [OP_SIZE];
}
CPT_OP;
typedef struct cpt_param
{
    SKTAL_USHORT len;

```

```

        SKTAL_OCTET data [PR_SIZE];
    }
    CPT_PARAM;
    typedef struct cpt_error_code
    {
        SKTAL_USHORT len;
        SKTAL_OCTET data [ER_SIZE];
    }
    CPT_ERROR_CODE;
    typedef struct cpt_problem
    {
        SKTAL_USHORT len;
        SKTAL_OCTET data [PB_SIZE];
    }
    CPT_PROBLEM;
    /*
    * Invoke primitive. REQ and IND
    * Invoke not last primitive. REQ and IND
    */
    typedef struct cpt_invoke
    {
        CPT_INV_ID invoke_id;
        CPT_OP operation;
        CPT_PARAM param;
#ifdef CCITT
        SKTAL_USHORT opClass; /* 1, 2, 3 or 4 */
#endif
        SKTAL_USHORT timeout; /* 0 .. 409 */
        CPT_INV_ID linked_id;
#define correlation_id linked_id /* FOR ANSI */
    }
    CPT_INVOKE;
    /*
    * Return result last primitive. REQ and IND
    * Return result not last primitive. REQ and IND
    */
    typedef struct cpt_result
    {
        CPT_INV_ID invoke_id;
#ifdef CCITT
        CPT_OP operation;
#endif
        CPT_PARAM param;
    }
    CPT_RESULT;
    /*
    * User error primitive. REQ and IND
    */
    typedef struct cpt_error
    {

```

```

        CPT_INV_ID invoke_id;
        CPT_ERROR_CODE error;
        CPT_PARAM param;
    }
    CPT_ERROR;
    /*
    * User reject primitive. REQ and IND
    * Local reject primitive. IND only.
    * Remote reject primitive. IND only.
    */
    typedef struct cpt_reject
    {
        CPT_INV_ID invoke_id;
        CPT_PROBLEM problem;
#ifdef ANSI
        CPT_PARAM param;
#endif
    }
    CPT_REJECT;
    /*
    * User cancel primitive. REQ only.
    * Local cancel primitive. IND only.
    */
    typedef struct cpt_cancel
    {
        CPT_INV_ID invoke_id;
    }
    CPT_CANCEL;
    /*
    * Timer Reset primitive (ITU White Book 97 only). REQ
    * only.
    */
#ifdef CCITT
    typedef struct cpt_timerReset
    {
        CPT_INV_ID invoke_id;
    }
    CPT_TIMER_RESET;
#endif
    /*
    * Union of all of the above
    */
    typedef struct tcap_cpt
    {
        SKTAL_USHORT last_component; /* either 0 or non-zero
        */
        SKTAL_USHORT ptype; /* prim type (TCPPT_xxx values) */
        Union
        {
            CPT_INVOKE invoke;

```

```

        CPT_RESULT result;
        CPT_ERROR error;
        CPT_REJECT reject;
        CPT_CANCEL cancel;
#if defined(CCITT)
        CPT_TIMER_RESET timerReset;
#endif
    }
    u;
}
TCAP_CPT;
/
*****
*****
*
*
*      Structure definitions for Dialogue Primitives
*
*
*
*****
*****/
/*
* Dialog substructures
*/
typedef struct dlg_qos
{
    SKTAL_OCTET indicator;
    SKTAL_OCTET sls_key;
    SKTAL_OCTET priority;
    SKTAL_OCTET networkInd;
}
DLG_QOS;
typedef struct ac_name
{
    SKTAL_USHORT len;
    SKTAL_OCTET data [AC_SIZE];
}
DLG_AC_NAME;
typedef struct usr_inf
{
    SKTAL_USHORT len;
    SKTAL_OCTET data [UI_SIZE];
}
DLG_USR_INF;
typedef struct abt_inf
{
    SKTAL_USHORT len;
    SKTAL_OCTET data[AB_SIZE];
}

```

```

DLG_ABT_INF;
/*
 * ITU and ANSI UNI. REQ and IND.
 */
typedef struct dlg_uni
{
    SKTAL_USHORT cpt_present; /* 0 or 1 */
    DLG_QOS qos;
    DLG_AC_NAME ac_name;
    DLG_USR_INF user_info;
    SCCP_ADDR orig_addr;
    SCCP_ADDR dest_addr;
    MTP3_POINT_CODE opc; /* for use when address doesn't
include */
    MTP3_POINT_CODE dpc; /* for use when address doesn't
include */
}
DLG_UNI;
/*
 * ITU BEGIN, ANSI QUERY W/PERM and WO/PERM. REQ and IND.
 */
typedef struct dlg_begin
{
    SKTAL_USHORT cpt_present; /* 0 or 1 */
    DLG_QOS qos;
    DLG_AC_NAME ac_name;
    DLG_USR_INF user_info;
    SCCP_ADDR orig_addr;
    SCCP_ADDR dest_addr;
    MTP3_POINT_CODE opc; /* for use when address doesn't
include */
    MTP3_POINT_CODE dpc; /* for use when address doesn't
include */
}
DLG_BEGIN;
/*
 * ITU CONTINUE, ANSI CONV W/PERM and WO/PERM. REQ and
IND.
 */
typedef struct dlg_continue
{
    SKTAL_USHORT cpt_present; /* 0 or 1 */
    DLG_QOS qos; /* Indication only. Ignore for request
*/
    DLG_AC_NAME ac_name;
    DLG_USR_INF user_info;
    SCCP_ADDR orig_addr;
    MTP3_POINT_CODE opc; /* for use when address doesn't
include */
}
DLG_CONTINUE;

```

```

/*
 * ITU END, ANSI RESP. REQ and IND.
 */
typedef struct dlg_end
{
    SKTAL_USHORT cpt_present; /* 0 or 1 */
    DLG_QOS qos; /* Indication only. Ignore for request */
    DLG_AC_NAME ac_name;
    DLG_USR_INF user_info;
    SKTAL_OCTET termination; /* 0 or 1 */
}

DLG_END;
/*
 * ITU U-ABORT. REQ only.
 * ITU P-ABORT. IND only.
 * ANSI ABORT, IND only.
 */
typedef struct dlg_abort
{
    SKTAL_USHORT abort_reason;
    DLG_QOS qos; /* Indication only. Ignore for request */
    DLG_AC_NAME ac_name; /* P-ABORT does not include this */
    DLG_USR_INF user_info; /* P-ABORT does not include this */
    DLG_ABT_INF abort_info; /* ANSI only */
}

DLG_ABORT;
/*
 * ITU NOTICE and ANSI NOTICE. IND only (based on QOS
 * return option).
 */
typedef struct dlg_notice
{
    SKTAL_OCTET report_cause;
    SKTAL_OCTET user_data_len; /* Presence depends on
 * manufacturer */
    SCCP_ADDR orig_addr; /* Presence depends on
 * manufacturer */
    SCCP_ADDR dest_addr; /* Presence depends on
 * manufacturer */
    SCCP_DATA user_data; /* Presence depends on
 * manufacturer */
}

DLG_NOTICE;
/*
 * Union of above types
 */
typedef struct tcap_dlg
{

```

```
    SKTAL_USHORT    ptype; /* primitive type (TCPPT_XXX
values) */
    Union
    {
        DLG_UNI uni;
        DLG_BEGIN begin;
        DLG_CONTINUE cont;
        DLG_END end;
        DLG_ABORT abort;
        DLG_NOTICE notice;
    }
    u;
}
SKTAL_DLG;
```

E Appendix - SKIM and SKTAL API Integration with IntelliNet Codecs

Overview

This appendix describes the integration of SKIM and SKTAL API with IntelliNet Technologies' codecs. A step-by-step sequence is provided to facilitate the integration.

.

Integrating SKIM and SKTAL API with IntelliNet Codecs

Steps Follow these steps to integrate SKIM and SKTAL API with IntelliNet codecs.

1 Install the SKIM and SKTAL API.

```
uncompress <Filename>
tar -xvf <Filename>
```

2 Install a codec.

The codecs available from Dialogic are:

- Customized Application for Mobile network Enhanced Logic (CAMEL)
- Universal Mobile Telecommunications System- Mobile Application Part (UMTS- MAP) which includes Global System for Mobile communication - Mobile Application Part (GSM- MAP)
- ANSI - 41
- Wireless Intelligent Network (WIN)

For installation instructions on these codecs see, the following documents (available from Dialogic):

- CAMEL User's Guide, document part number
P_EXCL_CAMEL_UG_2.0
- UMTS MAP User's Guide, document part number
P_EXCL_UMTS_MAP_UG_2.0
- WIN User's Guide, document part number
P_EXCL_WIN_UG_2.0
- ANSI 41D User's Guide, document part number
P_EXCL_AS41D_UG_2.0

The default directory for a codec is: /opt/Intellinet/<CODEC>/

Note

- If you have installed the CCITT based codec, the codec header files get installed under /opt/IntelliNet/<CODEC>/include/itu/ directory.
- If you have installed the ANSI based codec, the codec header files get installed under /opt/IntelliNet/<CODEC>/include/ansi/ directory.
- The Codec library files get installed under /opt/IntelliNet/<CODEC>/lib/ directory

-
- 3** Build a provided sample application. Open the relevant makefile. For a sample SKIM application, use the steps described in Chapter 4. The SKIM sample application files, *test1.cpp* and *test2.cpp*, can be found in the directory:

\$ITS_ROOT/SKIM/TEST

For a sample SKTAL sample application, use the steps described in Chapter 7. The SKTAL sample application files, *test1.cpp* and *test2.cpp*, can be found in the directory:

\$ITS_ROOT/TAL/TEST

For the sample application to use the installed codec, modify the makefile to link the codec header files and codec library files pointing to the directory chosen in Step 2.

-
- 4** Compile the sample application using the makefile.

-
- 5** Before running the sample application, which uses one of the IntelliNet codecs, download the IntelliNet codec license file *its.lic* to the directory where the sample application executable is started.

END OF STEPS
