USSD S-Gateway USSDJNI JavaAPI

User's Guide v1.5

Version control

Version	Date	Author	Comment
R0	2010/04/13	Alejandro Leib	Initial Version
V1.1	2010/06/16	Alejandro Leib	SC function added
V1.2	2011/07/04	Alejandro Leib	NI functions added
V1.3	2012/02/07	Alejandro Leib	ServiceEx added
V1.4	2012/09/25	Alejandro Leib	High Availability added
V1.5	2012/12/12	Alejandro Leib	Library handling details

Notice

LeibICT

Montevideo, Uruguay Tel: +598 2 614 11 93 Fax: +598 2 203 66 54

http://www.leibict.com

Warranty

LeibICT Solutions keep a technical support department with the only purpose of providing efficient and reliable service.

All LeibICT products are warranted against defects in material and workmanship. The period of coverage and other warranty details are specified in the LeibICT terms and conditions warranty. In no event shall LeibICT be liable for incidental or consequential damages in connection with, or arising from use of any LeibICT product.

Table of Contents

Introduction

LeibICT USSD S-Gateway supports applications over MAP (Mobile Application Part) (phase 2 and 3), either mobile or USSD application originated, using the message set defined in the protocol. The mobile originated data are received by the USSD application and forwarded to the HLR (Home Location Register), where as the network originated messages are forwarded to the HLR at once.

This document presents a description of the USSDJNI JavaAPI supported and managed by the LeibICT's USSD S-Gateway.

The USSDJNI JavaAPI offers a high abstraction layer for java Applications witch:

1) Controls the TCP/IP connection, including reconnection.
2) Implements the RSA encryption and decryption.
3) Implements the XML creation and parsing.
4) Controls the USSD/MAP protocol details.

Capacity

The current USSD S-Gateway version supports 16384 incoming and outgoing messages by stack, with capability of two stacks running independently, so a total capacity of 32678 incoming and outgoing simultaneous messages

Each USSD transaction is identified by a dialog identifier (field named: *DialogueId*), represented by 16 bits. This parameter must be included in each dialogue message so the USSD application must be capable of controlling such parameter. For each outgoing message coming from the application, the latter must include a *Dialogue Id* between 0 and 16383, where as when it receives a message, it must answer using the same *Dialogue Id* of the incoming message, which will be a digit between 16384 and 32768.

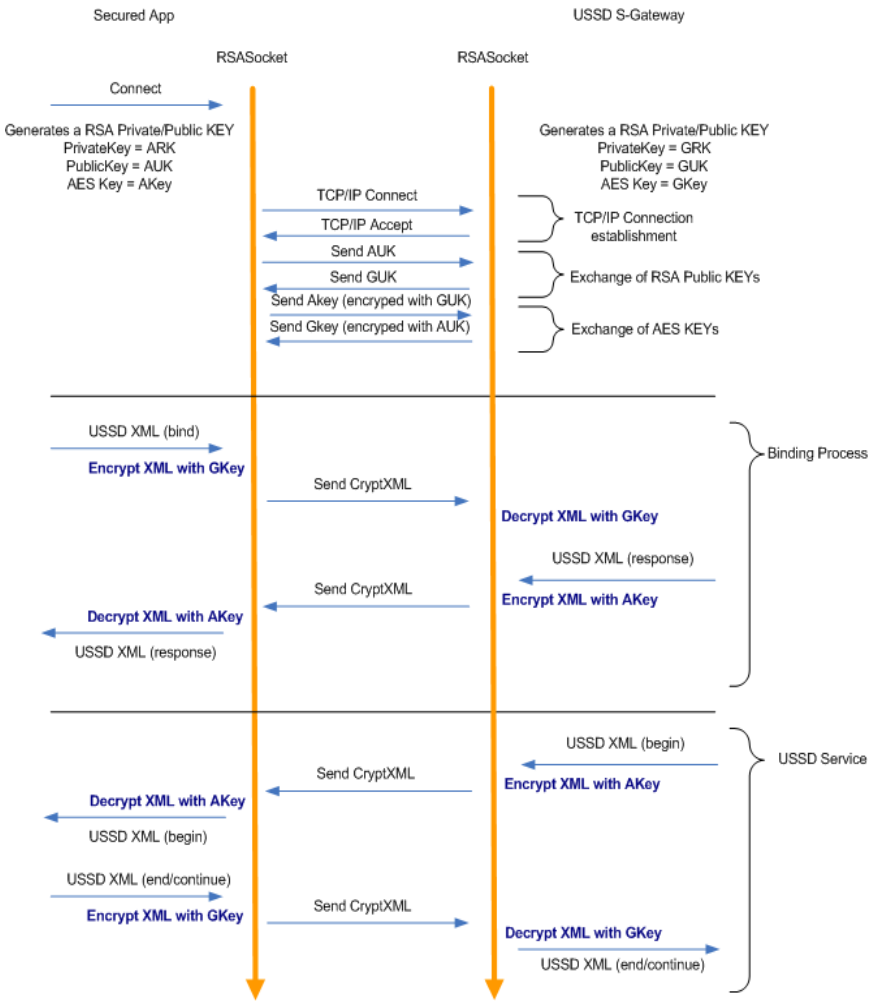
The RSA encryption and decryption process is very high cpu consuming, setting up the right key length is needed to depending of the traffic estimated.

RSA/AES Secured mode

The USSD S-Gateway may be configured to use secure AES and RSA PKI Encryption. This mode assures a totally secure communication, valid for banking applications over non-secured lans.

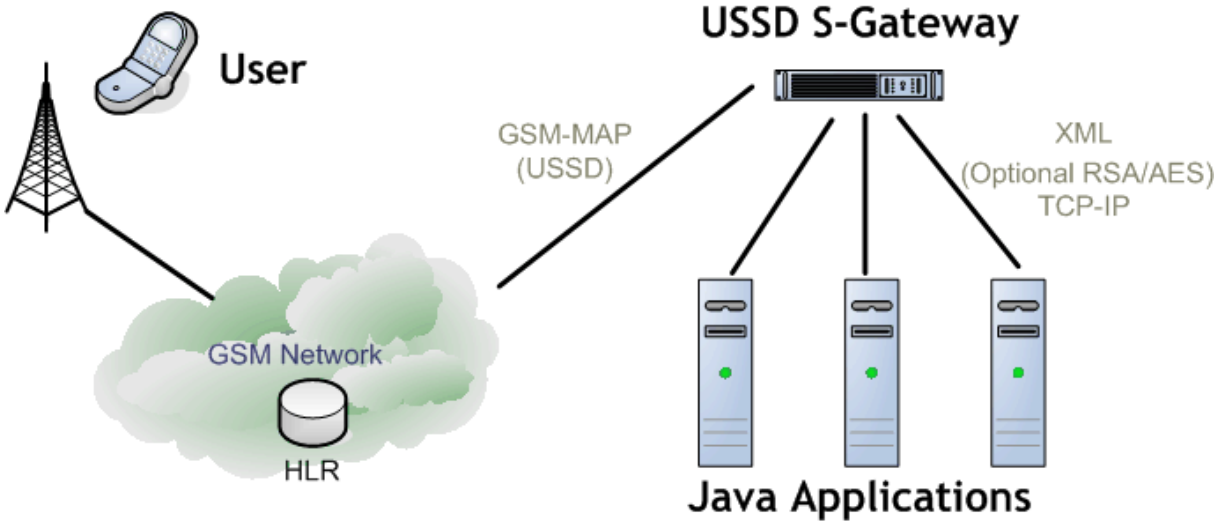
Due to the asymmetric load of the operations of encryption and decryption, the connection transmission and reception can be secured using different key length, or even without it.

For example, if your application vulnerable point is information presentation, you can encrypt the reception with 4096 RSA and 256 AES bits keys, and maintain the transmission unsecured. Otherwise if your application critical information is user input, you can encrypt the transmission.



Typical system structure

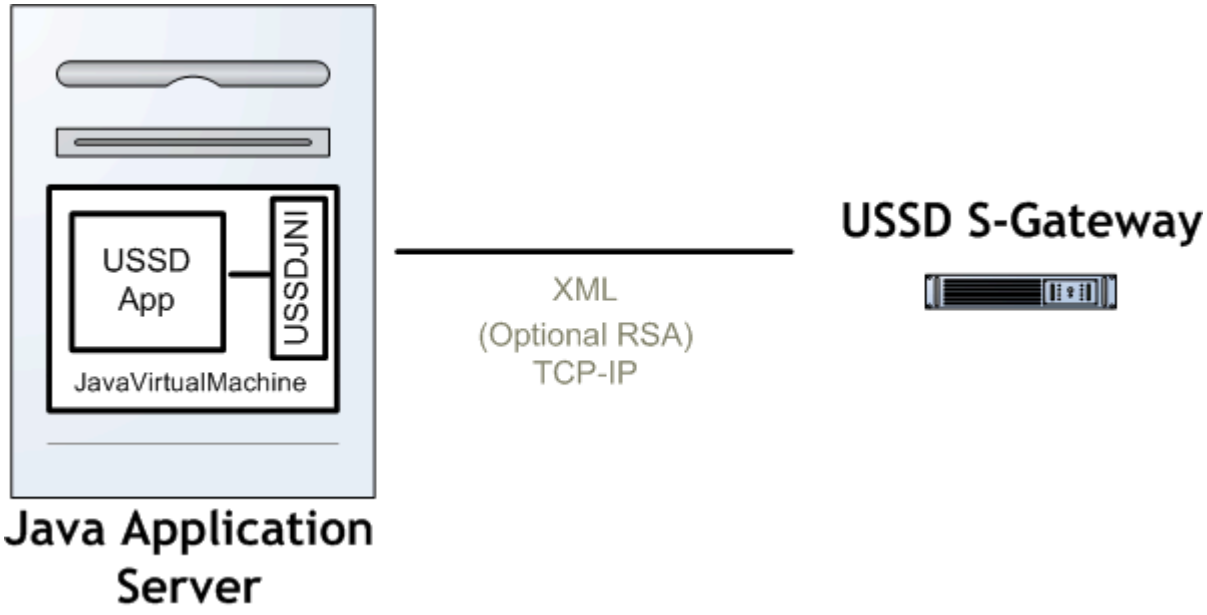
The LeibICT USSD S-Gateway provides Load Balancing and Fault-Tolerance of applications:



Two USSD S-Gateways can be paired to provide Fault-Tolerance on the gateway and GSM Network level.

Software Framework

The USSDJNI JavaAPI Library consists in a group of dynamically linked libraries developed in C++ that are loaded into de Java VirtualMachine:



The USSDJNI JavaAPI Library is implemented over JNI (Java Native Interface) to provide a framework for USSD Applications.

High Availability

The USSD GW support a dual node High Availability (HA) operation mode.

Dual node HA enables load sharing and automatic fail-over without human intervention.

The application logic and business sequences needs no modifications.

In order to work within the HA mode, the application must not call "connect" but "connect_ha" instead (described in the next chapters).

Calling the API "connect" on a HA enabled USSD GW will not fail, but the application will not receive traffic.

Calling the API "connect_ha" on a non-HA enabled USSD GW will not fail, but the application will not receive traffic.

API Architecture

The USSDJNI JavaAPI library enables programmers to access the services provided by USSD S-Gateway and protocol using a Java function and callbacks based interface.

The USSDJNI JavaAPI – USSD S-Gateway communication mechanism used is transparent to the programmer, as all the socket management is performed by the USSDJNI library.

When the USSDJNI library is initialized by calling **initialize** and **connect**, a worker thread is started which is responsible for socket management.

Events received from USSD S-Gateway are reported to the user application from this thread using a callback mechanism.

The USSDJNI library is thread-safe from the user's point of view. However, the user is responsible for protecting any of his own data structures accessible from multiple threads.

The USSDJNI worker thread only delivers one event at a time. That is, it does not notify the user application of a new event until the user returns control from the current event handler. However this means that the user needs to avoid using blocking or lengthy operations within callback functions.

API Functions

These are the API functions :

.1 initialize

private native int initialize();

Description

The initialize function initializes the JNI library within the Java Virtual Machine.

Output

Return values:

0 – An error occurred during JVM/JNI initialization

1 – Initialization successfully

.2 ussd_requestShortCode

public static native int ussd_requestShortCode(String shortCode);

Input

Parameter	Description	Example
shortCode	Interest USSD ShortCode	*123#

Description

The ussd_ requestShortCode should be called before the connect function to indicate the USSD S-Gateway of a traffic filter.

Output

Return values:

0 – An error occurred

1 – The command was successfully received by the USSD S-Gateway

.3 connect

private native int connect(String addr,int port);

Input

Parameter	Description	Example
addr	IP or Domain Name of the USSD S-Gateway	10.0.0.1
port	TCP port of the USSD S-Getway	5454

Description

The connect function tries to connect the USSD S-Gateway. If it success and there is a network failure, there is no need to call this function again, the USSDJNI will try to reconnect automatically.

Output

Return values:

0 – An error occurred during socket connection establishment or while securing it.

1 – Connection and security established successfully

.4 connect_ha

private native int connect(String addr1,String addr2, int port);

Input

Parameter	Description	Example
addr1	IP or Domain Name of the USSD S-Gateway 0	10.0.0.1
port1	TCP port of the USSD S-Getway 0	5454
addr2	IP or Domain Name of the USSD S-Gateway 1	
port2	TCP port of the USSD S-Getway 1	5454

Description

The connect function tries to connect the USSD S-Gateway. If it success and there is a network failure, there is no need to call this function again, the USSDJNI will try to reconnect automatically.

Output

Return values:

0 – An error occurred during socket connection establishment or while securing it.

1 – Connection and security established successfully

.5 ussd_request

private native int ussd_request(int dialogId,String prompt);

Input

Parameter	Description	Example
dialogId	Dialog of the current USSD Service	1
prompt	String containing a prompo for the user	"Enter the PIN"

Description

The ussd_request function sends a message to the user trough the USSD S-Gateway requesting some input.

Output

Return values:

0 – An error occurred

1 – The command was successfully received by the USSD S-Gateway

.6 ussd_end

private native int ussd_end(int dialogId,String info);

Input

Parameter	Description	Example
dialogId	Dialog of the current USSD Service	1
info	String containing the last message to the user	"Have a nice day"

Description

The ussd_end function sends a message to the user trough the USSD S-Gateway finalizing the dialog.

Output

Return values:

0 – An error occurred

1 – The command was successfully received by the USSD S-Gateway

.7 pong

private native int pong();

Description

The pong function sends a pong to the USSD S-Gateway. The pong function must be the response to the ping call-back. If the application does not respond the ping with a pong, the USSD S-Gateway may disconnect the application.

The intervals between pings are defined in the USSD S-Gateways and cannot be determined in the Application side.

Output

Return values:

0 – An error occurred

1 – The command was successfully received by the USSD S-Gateway

.8 ussd_begin

private static native int ussd_begin(int dialogId,String hlrGt,String phoneNumber,String text){}

Input

Parameter	Description	Example
dialogId	Dialog of the current USSD Service (1- 32768)	1
hlrGt	HLR Global Title	59899999901
phoneNumber	MSISDN	59899256038
text	Text to send	"hello"

Description

The ussd_begin function sends a message to the user trough the USSD S-Gateway while initiating a dialog: the subscriber is capable of sending a response.

Output

Return values:

0 – An error occurred

1 – The command was successfully received by the USSD S-Gateway

.9 ussd_notify

private static native int ussd_notify(int dialogId,String hlrGt,String phoneNumber,String text){}

Input

Parameter	Description	Example
dialogId	Dialog of the current USSD Service (1- 32768)	1
hlrGt	HLR Global Title	59899999901
phoneNumber	MSISDN	59899256038
text	Text to send	"hello"

Description

The ussd_notify function sends a message to the user trough the USSD S-Gateway without initiating a dialog: the subscriber is only capable of ending the dialog.

Output

Return values:

0 – An error occurred

1 – The command was successfully received by the USSD S-Gateway

API Callbacks

These are the API Callbacks:

.1 cb_ussd_service

private void cb_ussd_service(int dialogId,String shortCode,String phoneNumber) {}

Output

Parameter	Description	Example
dialogId	Dialog of the current USSD Service (32768 – 65536)	32768
shortCode	String containing the shortcode entered by the user	"*123#"
phoneNumber	String containing the phone number of the user	"888-123-123"

Description

The *cb_ussd_service* call-back is called when a new ussd service is started by the User.

.2 cb_ussd_serviceEx

private void cb_ussd_service(int dialogId,String shortCode,
String destinationReference,
String originationReference,
String destinationNumber,
String originationNumber,
String msisdn) {}

Output

Parameter	Description	Example
dialogId	Dialog of the current USSD Service (32768 – 65536)	32768
shortCode	String containing the shortcode entered by the user	"*123#"
destinationReference	MAP 29.002 Destination Reference	"8881231230"
originationReference	MAP 29.002 Origination Reference	"8881231231"
destinationNumber	MAP 29.002 Destination Number	"8881231232"
originationNumber	MAP 29.002 Origination Number	"8881231233"
phoneNumber	String containing the phone number of the user	"8881231234"

Description

The *cb_ussd_serviceEx* call-back is called when a new ussd service is started by the User.

.3 cb_ussd_response

private void cb_ussd_response(int dialogId,String str) {}

Output

Parameter	Description	Example
dialogId	Dialog of the current USSD Service (32768 – 65536)	32768
str	String containing the user response of a request.	"1"

Description

The *cb_ussd_response* call-back is called when a user responses a request.

.4 cb_ussd_end

private void cb_ussd_end(int dialogId) {}

Output

Parameter	Description	Example
dialogId	Dialog of the current USSD Service (1 – 65536)	1

Description

The *cb_ussd_end* call-back is called when the user, the GSM network or the USSD S-Gateway ends the dialog.

.5 cb_disconnected

private void cb_disconnected() {}

Description

The *cb_disconnected* call-back is called when USSDJNI library gets disconnected from the USSD S-Gateway.

.6 cb_connected

private void cb_connected() {}

Description

The *cb_connected* call-back is called when USSDJNI library gets connected again to the USSD S-Gateway.

.7 cb_ping

private void cb_ping() {}

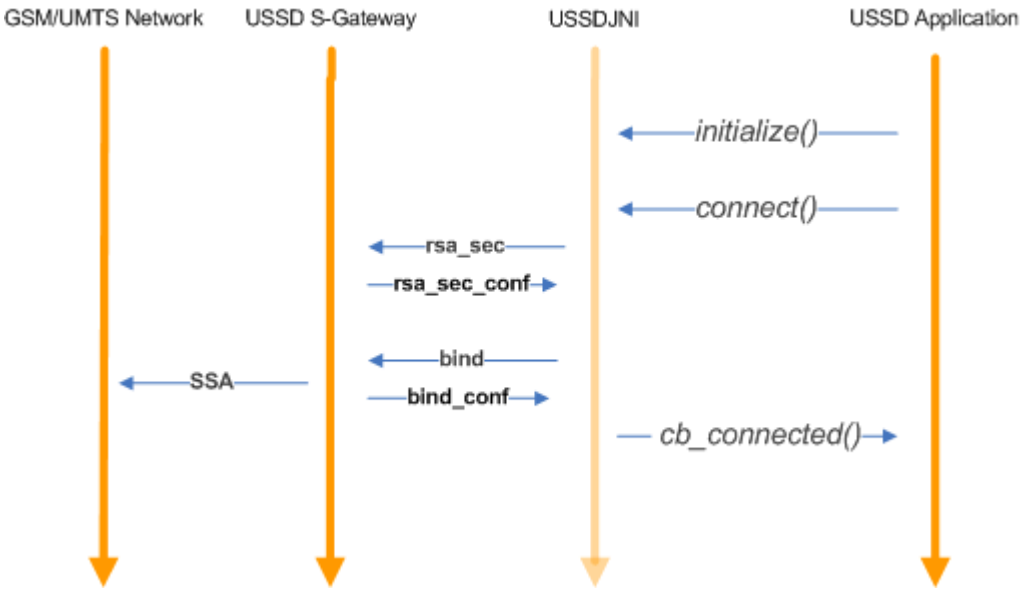
Description

The *cb_ping* call-back is called when USSD S-Gateway sends a ping to the Application.

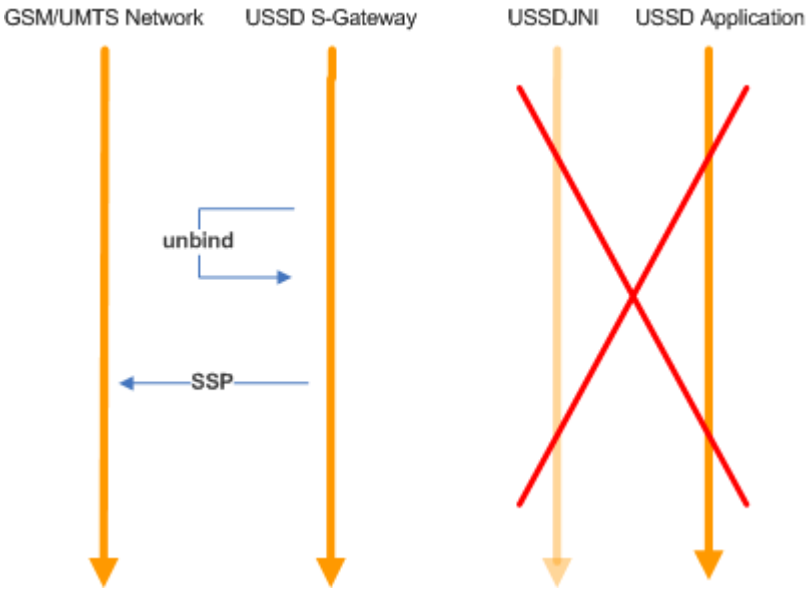
Example Message Sequences

The fallowing chapter gives example message sequences for interacting with USSDJNI Java API.

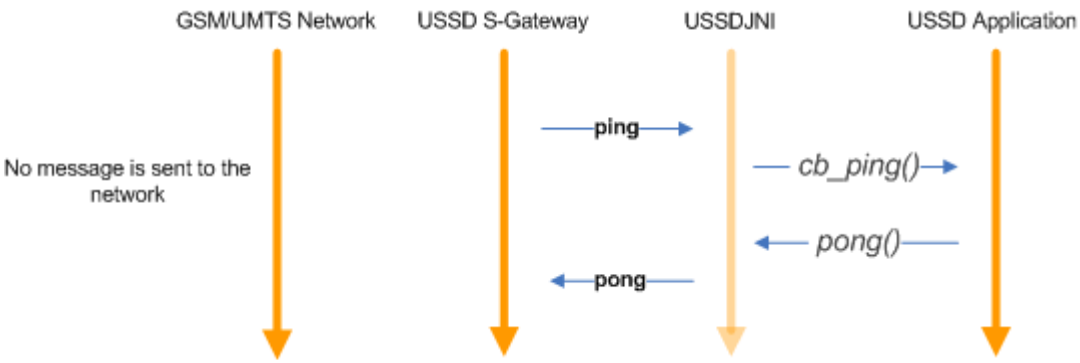
.1 Succesfull Application Connection



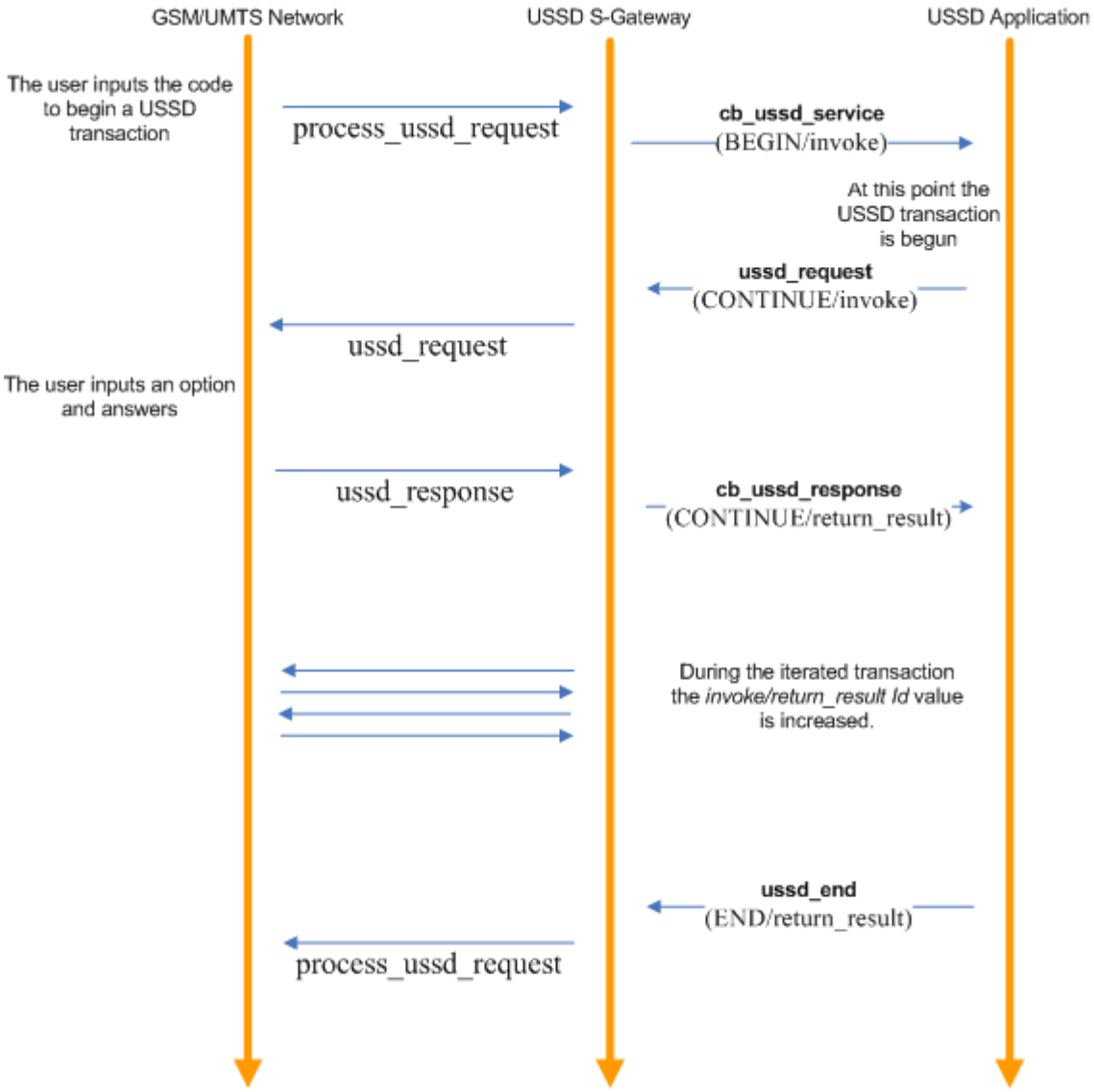
.2 Application Disconnection



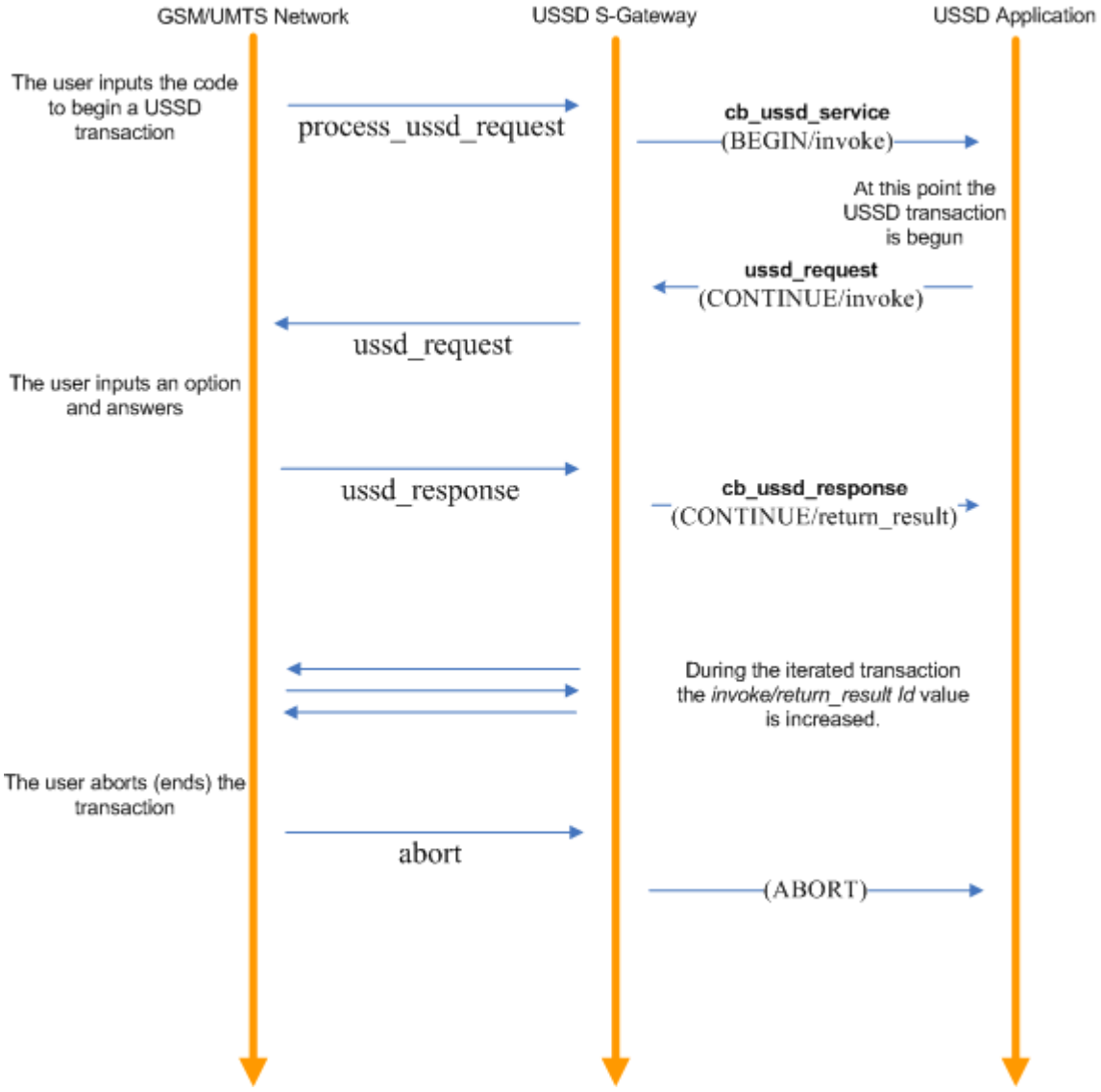
.3 Ping-Pong


GSM/UMTS Network
USSD S-Gateway
USSDJNI
USSD Application
No message is sent to the network
ping
cb_ping()
pong()
pong

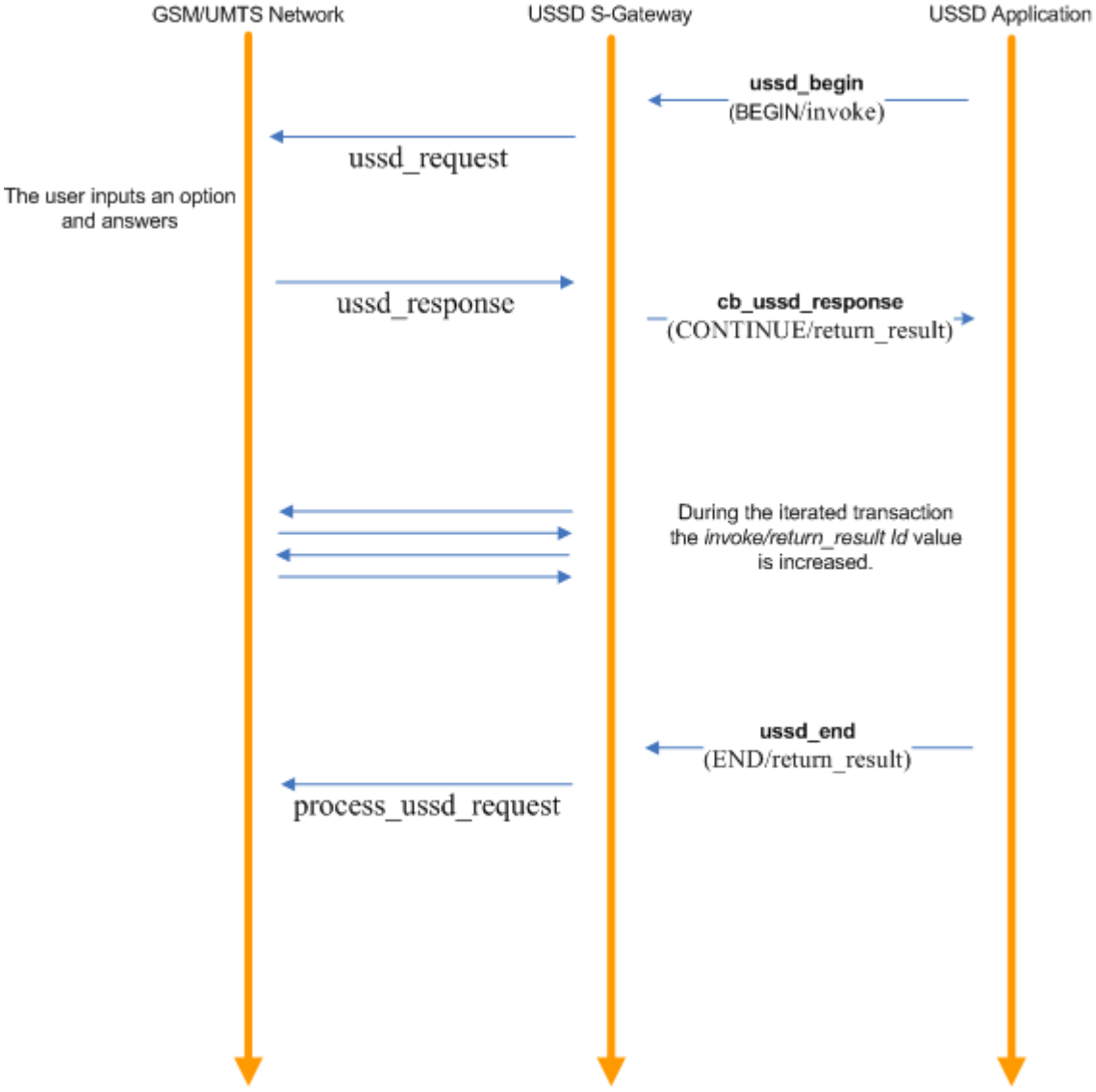
.4 Mobile Initiated USSD Transaction Ended by Application



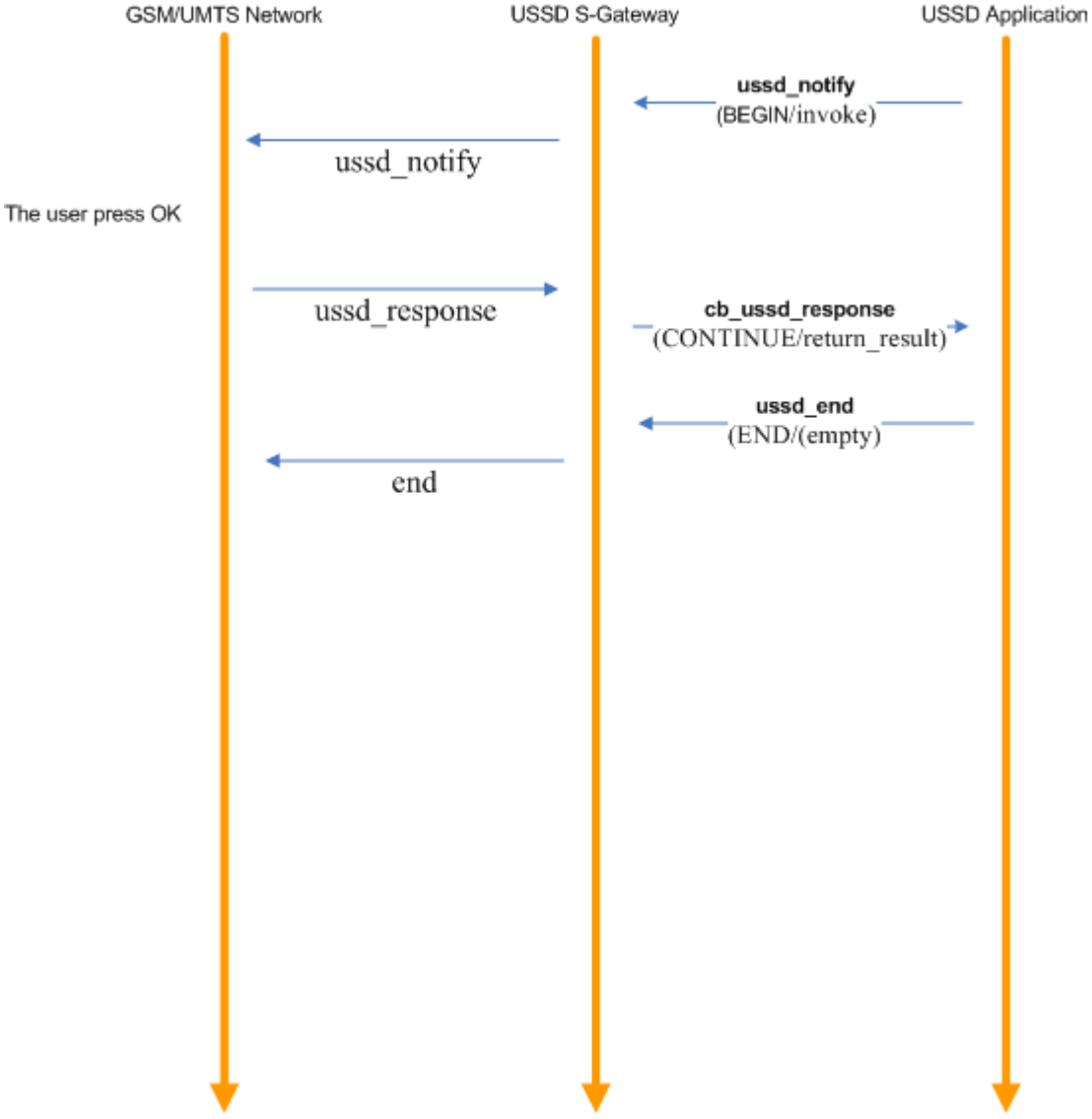
.5 Mobile Initiated USSD Transaction Aborted by User



.6 Mobile Terminated USSD Transaction Ended by Application



.7 Mobile Terminated USSD Notify Ended by Application



Length of the USSD String

In GSM 0902 160 octets is stated as the maximum length for the USSD string. Due to underlying signalling layers the maximum length of the USSD string depending on the message is:

USSD operation	Max length
Begin, Invoke ProcessUSSDRequest	133
End, Result ProcessUSSDRequest	160
First Continue, Invoke USSDRequest in mobile initiated dialogue	154
Begin, Invoke USSDRequest	144
First Continue, Result USSDRequest in network initiated dialogue	154
Other messages	160

Dialog IDs

Dialog IDs are generated by the GW for incoming (Mobile Originated) dialogs, and by the application for outgoing (Mobile Terminated) dialogs.

To avoid collisions two ranges are defined to work:

Range	Purpose
1 - 32767	Mobile Terminated
32768 - 65536	Mobile Originated

Its recommended to use a cyclical dialog selection pattern for MT applications, starting from 1 and increasing Dialog ID value by 1.

Once the value 32767 is reached, starting again from 1.

USSDJNI Configuration

The USSDJNI Library has on Windows a registry based configuration and on Linux a file based configuration.

Windows USSDJNI:

HKEY_LOCAL_MACHINE\SOFTWARE\LeibICT\USSDJNI

Linux USSDJNI:

./USSDJNI.conf

Field	Description	Default Value
log	Enable or Disable standard logs	1
logDebug	Enable or Disable debug logs	0
logDir	Directory where the logs are placed	c:\logs\
logErrorsFileName	Name of the file for errors logs	USSDJNIErrors.log
logFileName	Name of the file for standard and debug logs	USSDJNI.log
logTraffic	Enable or Diable traffic logs	0
logTrafficFileName	Name of the file for traffic logs	USSDJNITraffic.log

Library handling

The Linux and Windows JNI Java API is implemented through a dynamically linked library.

.1 Windows

In Windows its called: *ussdjni.dll*

The library is found by the Java Virtual Machine if its located in the "PATH" environment variable.

You can check the PATH variable while typing "PATH" in the command line, example:

C:\>PATH

PATH=C:\Program Files (x86)\CollabNet\Subversion Client;C:\Program Files\Common Files\Microsoft Shared\Windows Live;C:\Program Files (x86)\Common Files\Microsoft Shared\Windows Live;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem

Before using the library check the Window OS version as 32 and 64 versions require different libraries.

.2 Linux

In Linux its called: *libussdjni.so.1.0 / libussdjni.so*

The library is found by the Java Virtual Machine if its located in the "LD_LIBRARY_PATH" environment variable.

You need to setup a LD_LIBRARY_PATH for your libraries in the next way:

LD_LIBRARY_PATH=<my lib path>:$LD_LIBRARY_PATH

export LD_LIBRARY_PATH

Example:

LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH

export LD_LIBRARY_PATH

Before using the library check the Linux OS version as 32 and 64 versions require different libraries.

Important note: the distributed file is named *libussdjni.so.1.0* but the Java Virtual Machine loads the file *libussdjni.so* so keep the .so file as a symbolic link of the distributed file.

ln -s libussdjni.so.1.0 libussdjni.so