Dialogic®
Making Innovation Thrive™

Application Note

Using Dialogic® Distributed Signaling Interface Components for IS-41 Short Message Service Center Applications

Using Dialogic® Distributed Signaling Interface Components for IS-41 Short Message Service Center Applications

Application Note

Executive Summary

Developers adding Short Message Service (SMS) messaging to their solutions may chose to develop an application that combines the functions of SMS Center (SMSC) for receiving and storing messages, and Short Message Entity (SME) for creating and/or displaying the content of SMS messages.

This application note provides a demonstration and information about developing an IS-41 SMSC application that combines the functions of SMSC and SME using Dialogic® Distributed Signaling Interface Components. Dialogic has developed a reference application that provides both a starting point and a reference for an SS7-based application that performs SMSC functions. This reference application, which includes sample code and supporting files, is available for download with the application note.

Using Dialogic® Distributed Signaling Interface Components for IS-41 Short Message Service Center Applications

Application Note

Table of Contents

Using Dialogic® Distributed Signaling Interface Components for IS-41 Short Message Service Center Applications

Application Note

Introduction

A Short Message Service Center (SMSC) is the Signaling System 7 (SS7) equivalent of a post office for mail. Like a post office, SMSCs are responsible for receiving Short Message Service (SMS) messages and then forwarding them on to their end destination. The SMSC is also responsible for storing messages it receives for the end points it services when they are powered off, and then retrying delivery, as appropriate. The PC-based SME provides the functionality of creating and/or displaying the content of SMS messages. Developers adding SMS messaging to their solutions may chose to develop an application that combines SMSC and a Short Message Entity (SME).

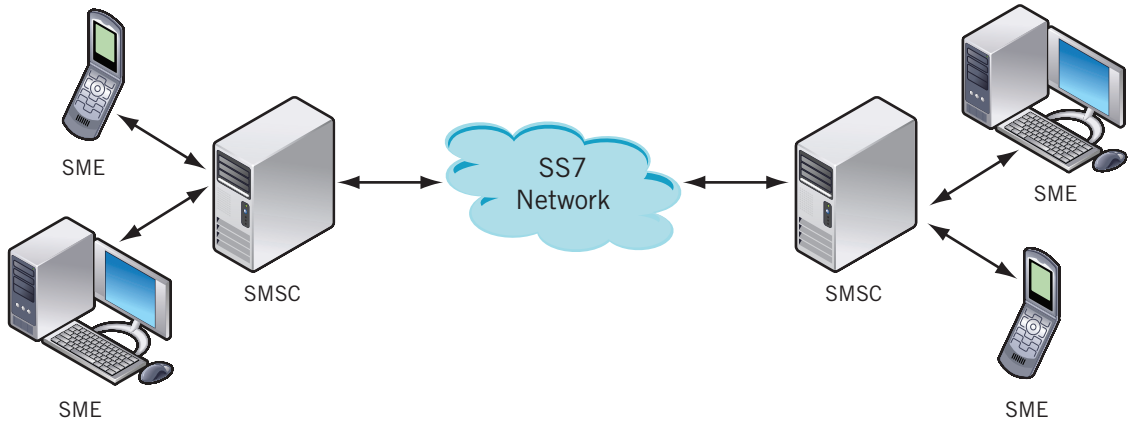

Figure 1. Simplified SMS Deployment

Figure 1 shows a simplified SMS environment. In reality, there are numerous other signaling components in use in the SS7 network that are not shown here. Note that it is possible to collapse the SMSC and SME functions into a single server running a single application. The sample code supplied with this application note functions in this way.

This application note provides information for developers who choose to build an SMSC application using Dialogic® Distributed Signaling Interface Components to provide a seamless and robust solution. The accompanying reference smsmc application demonstrates the use of the Dialogic® DSI signaling API to implement a basic SMSC application. Configurations include a conventional SS7 (board-based) example and a SIGTRAN example. The SIGTRAN example allows a developer to get started without purchasing SS7 hardware by using a demonstration license for the Dialogic® DSI Protocol Stacks. The architecture of both the development environment and the production environment is presented. In addition, a walkthrough of the accompanying smsmc code is provided.

A Zip file containing the sample code for the reference smsmc application and additional supporting applications is available for download (see the *For More Information* section).

Architecture

The PSTN, SS7, and SMS

The PSTN is comprised of multiple network elements that must communicate with each other for a calling party to reach a called party, and perform other functions associated with the call. Some of these network elements include Class 5 central office switches, Class 4 tandem switches, databases containing subscriber information, and servers that provide additional call control.

Each of these network elements uses SS7 messages to communicate with one another. Additionally, Signal Transfer Points (STPs) act as routers for SS7 messages to allow this communication. Bearer paths, made up of T1/E1 spans, DS3 spans, or optical links, provide the conduits for the voice traffic. All of these network elements use SS7 messaging to facilitate call delivery over the PSTN, so that the voice traffic can be effectively routed through the network. When a call is made in the PSTN network, SS7 messages are generated and routed through it from the local switch the caller is using to the switch the called party is connected to.

Since short messages consist of strings of text, a voice circuit does not need to be established to transmit them. Instead, the messages are transmitted and received in the context of a data transaction. The SS7 layer that handles this is known as the Transaction Capabilities Application Part (TCAP). The TCAP layer is used as the foundation for sending and acknowledging short message data.

The format of short messages carried by the TCAP layer is described by the ANSI/TIA/EIA-41 standard (IS-41), the North American standard for wireless telecommunications network signaling. Thus, this application note is oriented toward the North American SMS market. SMS in Europe is governed by the Mobile Applications Part (MAP) layer. While MAP procedures for sending and receiving short messages are similar to IS-41, they will not be covered here.

Simulating a Signaling Environment

During the development phase, an SMSC simulated environment can be established, eliminating the need for connection to the PSTN signaling environment (see Figure 2).

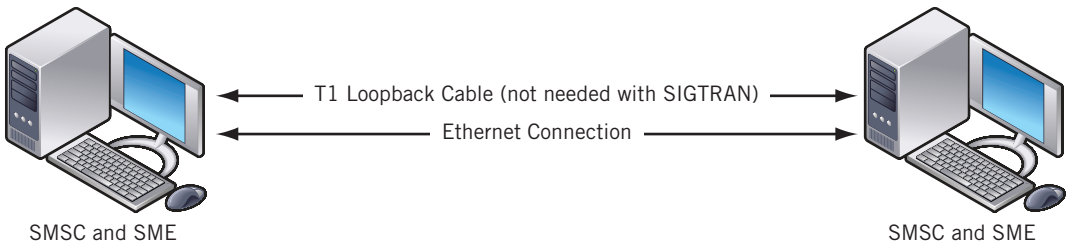

Figure 2. Simulated SMSC Environment

The simulated signaling environment consists of two servers, using either conventional SS7 or SIGTRAN protocol stack sets:

- **Conventional SS7** — Each server contains a Dialogic® DSI SS7 Board, Dialogic® DSI Protocol Stacks, and the reference smsmc application. Systems have one T1/E1 span on each system connected via a T1/E1 loopback cable.
- **SIGTRAN** — Each server contains DSI Protocol Stacks and the reference smsmc application. Systems communicate via the Dialogic® DSI M3UA Layers over IP.

SMS Text Encoding in the SMSMC Application

IS-41 encoding schemes used to convert the short message text into a series of Binary Coded Decimal (BCD) 4-bit hex values are specific to each carrier that offers SMS messaging. For a production system that will interact with the SS7 world at large, consult Teleservice specifications in order to determine the SMS text format to use.

A workable, though non-standard, scheme has been invented for use with this reference smsmc application. Text messages are strings of 7-bit ASCII alphanumeric values, converted to the BCD hex needed for the IS-41 message. Character conversion works as follows:

1. An ASCII character's hex value is converted to two single character representations of each hex digit.

2. Each hex digit is converted to a 4-bit BCD value and packed into the IS41 message.

For example:

A (ascii character) ➔ 0x41 (ascii hex value) ➔ '4' and '1' (two ascii characters) ➔ 0100 and 0001 (two BCD values in a single byte)

Reference SMSMC Application Architecture

This section is intended to provide a walkthrough of the reference smsmc application that accompanies this application note and contains information to assist developers in gaining an understanding of the sample code provided.

The reference smsmc application showcases the DSI signaling API used for the SMSC. This application was developed in C++ with a basic object orientated design. The design complexity was reduced in order to provide a clear example of the API commands needed.

Overview

The application may be compiled and run on either the Windows® or Linux operating systems. A Microsoft® Visual Studio® 2005 project and a set of Linux makefiles are both included. The code does not use libraries or APIs specific to either platform.

Two user interface options are available for the demonstration. The first is a simple text console, where short single-letter commands and text messages are entered. Application events are logged to this same console. The second option is to use the GUI provided by the QT cross-platform (Windows® or Linux) application framework from Trolltech.com. The open source version of QT, distributed under the Gnu Public License (GPL), was used.

The application is multithreaded. There is one thread to run the user interface, and a second to process incoming and send outgoing SS7 messages. Threading portability was achieved by using the Boost libraries. Boost is an open source project providing an extensive set of C++ library functions. One of these is the Boost Threads library.

The sample code available with this application note contains the signaling functions needed to run a basic SMSC environment shown in Figure 2. The smsmc executable runs on each of the two servers.

In a production environment, some of the messages sent and received may come from sources other than an SMSC. Home Locator Registrars (HLR), Visitor Locator Registrars (VLR), and SMS endpoints (cell phones, PCs, smart phones) are likely to generate and consume SMS messages.

SMSC Server

Both servers run the same reference smsmc application. The application is symmetrical; that is, either side can send or receive a message. Since both servers contain the same state machine for sending and receiving IS-41 messages, the server receiving the SMS message must not send an SMS message until the "receive" processing completes.

Reference SMSMC Application Description

SMS Messages and Application State Machines

The application uses several state machines to control the sending and receiving of the IS-41 messages that are used for SMS. Two message exchanges are demonstrated:

- **SMS Request and Request Response** — Queries the other SMSC for the SMS address of the handset to which the SMS message is to be delivered. Note that the queried SMSC is playing the role of a Home Location register (HLR). It sends back the SMS address of the handset to which the short message will be delivered, or an indication that the handset is not currently reachable.
- **SMS Delivery Point-To-Point (SMS DPTP)** — When the SMS address is known, the SMSC is able to encode and deliver a short message. Here, the receiving SMSC plays the role of a Mobile Switching Center (MSC). The IS-41 message exchange also includes a "successful" delivery response.

Three state machines are used to control these two message exchanges. The first is an "Initial" state machine, which sets up the receive side to expect either of the two messages. As the message exchange progresses, the message type becomes known. At that point, either the "SMS Request" or "SMS DPTP" state machine is invoked. On the sending side, the type of message is known when it is sent, so the appropriate state machine is immediately used.

Figures 3 and 4 outline the full IS-41 message exchanges and state transitions between the two applications for the SMS Request and SMS DPTP messages.

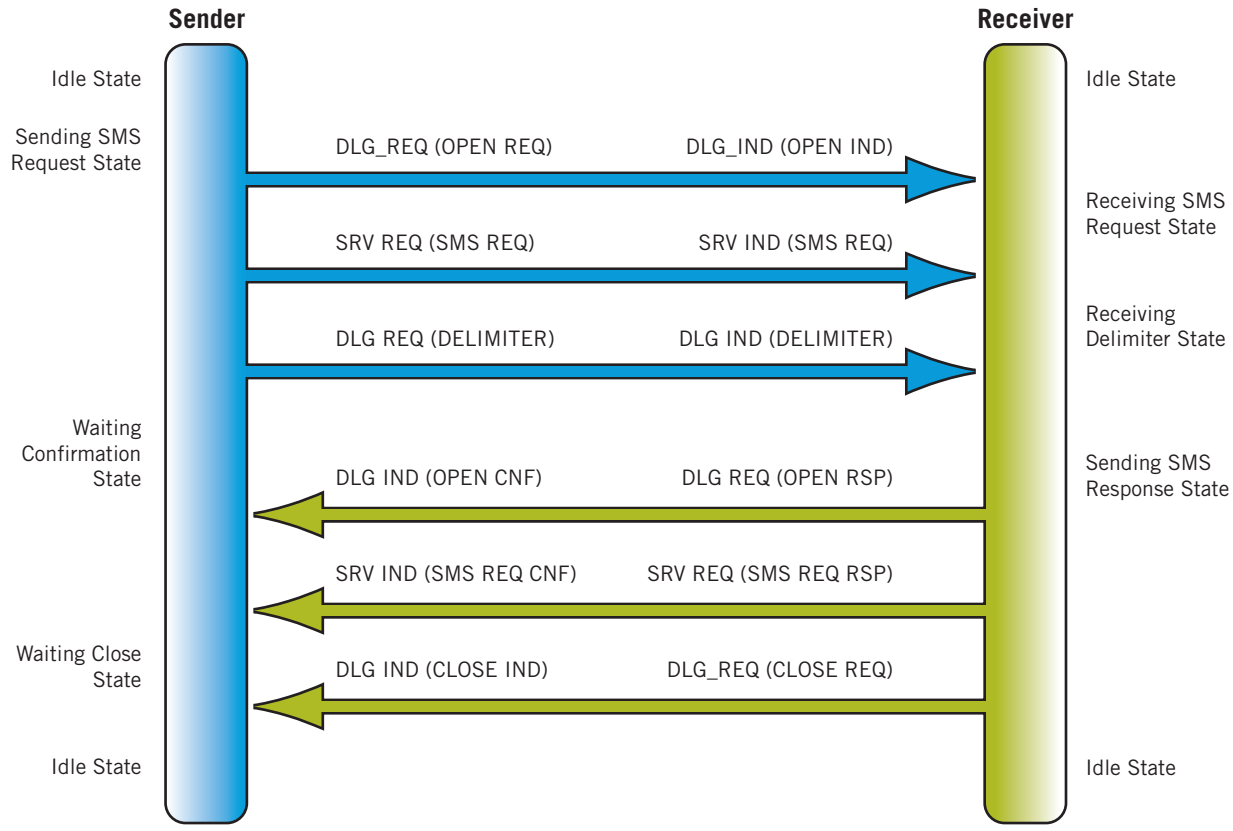

Figure 3. SMS Request Messages and State Machine

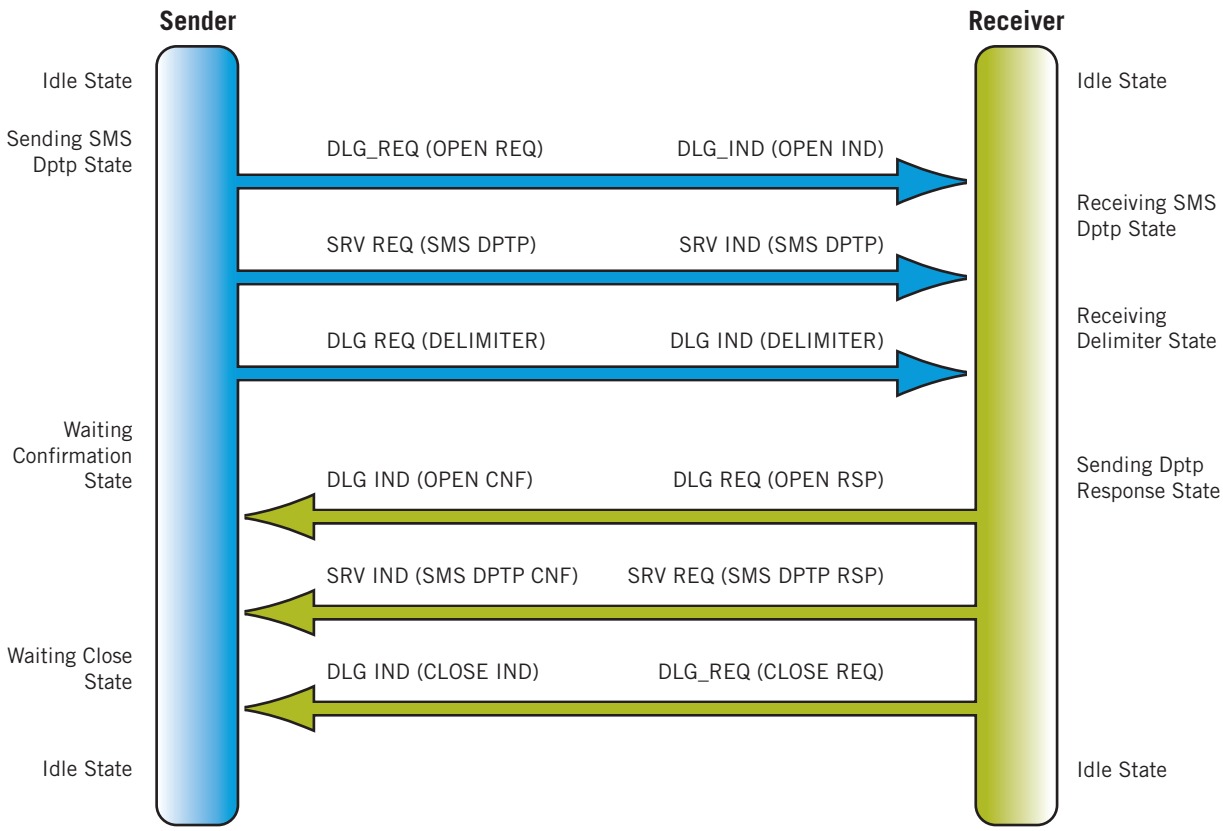

Figure 4. SMS DPTP Messages and State Machine

Functionality by Class

The following sections provide a breakdown of the functionality provided by class in the reference smsmc application.

SmsMcMain

This module is the entry point for the reference smsmc application.

The basic functions are:

- Initialize the application by reading and parsing the configuration file
- Set the interrupt handler, which catches a SIG_INT and "gracefully" shuts down the application by releasing Dialogic® resources and deleting objects. A graceful shutdown includes closing all open Dialogic® signaling devices, deleting all created objects, and closing all open threads.
- Initialize signaling state machines
- Create a signaling object and message receiving and sending objects

Other functions depend on the user interface's application:

- **Console application** — Loop forever waiting for console input to send messages or quit
- **QT GUI application** — Create a QT application and dialog. Display the main window and execute the application. All further control is done by the QT GUI that appears. All logging to stdout is done in the console where the application is started.

The reference smsmc application uses a configuration file to control its operation. The application will operate with default parameters if the configuration file is missing or not completely configured. SmsMcMain calls the accessor for the configuration object, and then calls the initialization module to set operating parameters. The application log and the signaling objects are then instantiated and initialized.

CConfigurationProc

The "CConfigurationProc" class is the class responsible for reading, parsing, and processing the configuration for the application. This class is a singleton, and as such, other classes can globally access it to get configuration information, reducing parameters passed between classes. The CConfigurationProc opens and reads parameters stored in a configuration file named "SmsMsgCtrApp.cfg".

All of the information contained in the configuration file is read in a single pass. The parameter values are parsed from the comments, spaces and parameter names, and stored as member variables. Public accessors provide the means for all classes to access parameter values.

Parameters can easily be added to the configuration file with modifications to the "setParametersFromFile" method. Developers would add parameter names to the "CommonParameters" array, set an enum value for the parameter, and add an accessor in order to add parameters to the configuration file.

CSignaling

The "CSignaling" class is the class responsible for receiving signaling messages and processing those messages according to internal state machines. As part of the signaling environment configuration, the SS7 configuration file "system.txt" was modified to include the entry 0x2d, which represents the reference smsmc application. This 0x2d value is used in the GCT_receive API, which, combined with the system.txt entry, delivers all signaling messages to the application (see Figure 5).

```
*
* Optional modules running on the host:
*
LOCAL 0x2d * SmsMcApp
```

Figure 5. system.txt Entry for SMSMC Application

The GCT_receive API runs in a separate thread of execution created using the Boost libraries. Code comprising this thread starts in the processMessagesReceived method of the CSignaling class.

GCT_receive is a blocking API call, returning only when a message for the application is placed into its signal processing queue. As a result, the CSignaling destructor utilizes the Boost thread's yield method in order to shut down the application. In a production

environment, taking the signaling link out of service would generate an application event that could be used to terminate the processing thread.

CSmsSenderMsg

The "CSmsSenderMsg" class is responsible for generating all signaling messages sent to the adjacent SMS server. Messages generated are based on the application state.

CSmsReceiveMsg

The "CSmsReceiveMsg" class is responsible for processing the SMS server and dialog messages received from the second server and building and sending reply messages when appropriate.

CAppLogger

The "CAppLogger" is responsible for logging information while the application is running. Log files are written to the same directory in which the application resides and are overwritten each time the application runs. This singleton class ensures that a single logging object exists for the entire application.

In addition to logging to a file, information is also written to stdout. Sample log entries are shown in Figure 6.

```
10:04:28.859 Monitor Signaling App Started
10:04:28.859 Received a IS41ST _ SMS _ IND message
10:04:28.875 Message type received = 8744
10:04:28.875 application ready for sending sms messages
10:04:42.312 TERMINATING PROGRAM
10:04:42.312 Exiting Signaling
```

Figure 6. Sample SMS Reference Log

Utilities

The "Utilities" module is not itself a class. Rather, it contains common methods used in other classes within the application.

A timestamp method is provided, to be used by the logging function to provide timestamps on the log file entries.

Dialogic® DSI Signaling API Usage

A number of Dialogic DSI signaling APIs are used within the reference smsmc application. These APIs are documented in the *Software Environment Programmer's Manual* (see the *For More Information* section). The first API call used is GCT_receive, which has a parameter of the application's process ID defined within the system.txt file. This call is a blocking call, returning only when there is a signaling message in the queue.

The getm API call is used to allocate an SS7 message and to initialize the message header fields to specific values. Additional parameters are initialized by advancing the pointer returned from getm for each parameter to be set and then setting the value.

Building, Configuring, and Running the Demonstration

Installing Dialogic® DSI Software

A trial license is built into several of the Dialogic® DSI Components. When they are started in trial mode, the processes will run for ten consecutive hours, then terminate and require a restart. This should be an adequate amount of time for running the demonstration and testing the products.

Two configurations are supported — conventional SS7 and SIGTRAN (SS7 over IP) on both the Linux and Windows® operating systems — so a variety of executables and configuration files are needed. The following components must be downloaded to build a demonstration system:

Conventional SS7

- SS7 Development Package (Linux or Windows®)
- Binary for MTP3 (Linux or Windows)
- Binary for SCCP (Linux or Windows)
- Binary for TCAP (Linux or Windows)
- Binary for IS-41 (Linux or Windows)
- Configuration files that accompany this application note as a download:
  - ss7boards/windows/system.txt
  - ss7boards/linux/system.txt
  - ss7boards/windows/config.txt
  - ss7boards/linux/config.txt

SIGTRAN SS7

- SS7 Development Package (Linux or Windows®)
- Binary for MTP3 (Linux or Windows)
- Binary for M2PA (Linux or Windows)
- Binary for SCCP (Linux or Windows)
- Binary for TCAP (Linux or Windows)
- Binary for IS-41 (Linux or Windows)
- Configuration files that accompany this application note as a download:
  - sigtran/windows/system.txt
  - sigtran/linux/system.txt
  - sigtran/windows/config.txt
  - sigtran/linux/config.txt

Installation is as follows:

- On Linux platforms, uncompress the downloaded SS7 Development Package into an SS7 system directory (default is /opt/dpklnx).
- On Windows® platforms, run the SS7 Development Package file dpkwin.exe. When prompted, select SS7 board drivers required, and proceed with the installation.

The directory "Septel" is created containing SS7 Development Package files.

Copy the appropriate system.txt and config.txt files into the SS7 directory. Make necessary edits (see the *SS7 Signaling Configuration* section) to the two files before starting the SS7 or SIGTRAN subsystems.

Installing Other Prerequisite Software

If the QT GUI is to be used, install the QT development packages. QT version 3.3.3 is used for the demonstration. Under RedHat Enterprise Linux version 4, use the GUI-based Package Management utility. KDE Software Development, Standard Package set should be added. This will install the QT libraries and QT Developer. Developer, QT's graphical development environment, is used only if modifications are to be made to the demonstration itself.

The Boost thread libraries must be installed before the application can be built. Much of Boost's functionality is available by including Boost header files. Macros and inline functions are used when possible. Other packages require libraries that must either be downloaded or built from source.

On Windows® platforms, the easiest route is to install pre-built Boost binaries, available from Boost Consulting. First, download a download manager executable. When that is run, select a mirror for the download. Next, select the version of Microsoft® Visual Studio® (2003/2005) being used for application development, and then select the type of library. Choose Multithreaded DLL. Boost header files and all libraries are selected by default. To avoid excessive download time, unselect all choices except the Headers and the Thread library.

On Linux platforms, it is necessary to build the single library needed (Boost thread):

```
tar xvfz boost_1_35_0.tar.gz
cd boost_1_35_0
./configure --with-libraries=thread
make install
```

Building the Demonstration Application

Locations for Dialogic, Boost, and QT (if used) C++ header and library files must be set before compilation and linking on either operating system.

On Linux — The demonstration comes with two Linux makefiles: consolemake, which builds a simple console user interface, and qtmake, which uses the QT GUI. Adjust C++ header and library locations before building.

On Windows® — A Microsoft® Visual Studio® 2005 project accompanies the demonstration. Adjust C++ header and library locations, build and run the application from Visual Studio. Note that a QT GUI application is not provided for Windows.

SS7 Signaling Configuration

Two sets of configuration files are provided with the demonstration for two different configurations:

- **Conventional SS7** — Each system requires a Dialogic® DSI SS7 Board, and a loopback cable is run between one of the spans on each board. Configurations for two DSI SS7 Boards — the higher density Dialogic® DSI SS7HD Network Interface Boards and the lower density Dialogic® DSI SPCI4 Network Interface Boards — are provided.
- **SIGTRAN** — Systems may be run in an IP-only configuration by using SIGTRAN as the transport rather than the lower-level DSI Protocol Stacks communicating over a TDM link. Hence, no Dialogic® DSI hardware is needed to run the demonstration.

The configuration needed to define the operation and functionality of the Dialogic DSI Components is contained in two files, config.txt and system.txt. Information for configuring the config.txt and system.txt file is found in the *Dialogic® SS7 Boards - SS7HD Programmer's Manual, Dialogic® SS7 Boards Programmer's Manual for SPCI2S and SPCI4*, or Programmer's Manuals for SIGTRAN Host Software.

system.txt

system.txt contains information related to which signaling protocol modules are loaded and includes commands to start required tasks. In addition, for conventional SS7, where they run — whether on the DSI SS7 Board or on the server hosting the DSI SS7 Board — is also set.

config.txt

config.txt is used to set up the configuration of the T1/E1 interfaces and the operating parameters for the DSI SS7 Board(s) or SIGTRAN links used.

Note: IP addresses in config.txt for the SIGTRAN configurations must be changed to accommodate the address of the actual systems being used.

Application Configuration File

Configuration parameters for the application are in the file SmsMsgCtrApp.cfg. Their names and descriptions are listed below:

HomeMsgCtrAddr — Identifies the message center address of the local SMS reference application server. This is an SCCP message address, which consists of the following parts:

- **Address Indicator** — Set according to the Q.713 definition of "Called Party Address" (0x43 in the example below)
- **SCCP Subsystem Number** — Identifies the local subsystem number (or application) within the node (0x01 or 0x02 in the example below)
- **Destination Point Code** — SS7 network address

The value entered is linked to several defined in config.txt:

- **SCCP** — Local sub-system number (0x0a or 0x0c in the example below)

Each server's HomeMsgCtrAddr must match the ServicingMsgCtrAddr of the other server, as seen in the following example:.

Example

Node A	Node B
HomeMsgCtrAddr = 430a020000	ServicingMsgCtrAddr = 430c01000
HomeMsgCtrAddr = 430c010000	ServicingMsgCtrAddr = 430a02000

ServicingMsgCtrAddr — Identifies the message center address of the remote SMS reference application server. See HomeMsgCtrAddr for an explanation.

MobileIdNum — This is an arbitrary value that would normally be the Mobile ID Number of the user sending the SMS message. The MobileIdentificationNumber (MIN) is a 10-digit representation of the MS's MIN, coded in BCD form (TIA/EIA SP-3588)

InterMsgCtrId — This is an arbitrary value that would normally be filled in as a result of an SMSRequest query made into the signaling network.

TeleserviceId — This is an arbitrary value that would normally be filled in as a result of an SMSRequest query made into the signaling network.

ElectronicSerialNumber — This is an arbitrary value which represents the electronic serial number assigned to the SMSC.

BearerData — This entry contains default SMS message data to be sent if none is supplied by the user when running the application.

Starting SS7 Services

Once the configurations files (config.txt and system.txt) are in place in the SS7 system directory on each server, each side is then started with "gctload –d" to start the SS7 services and then "mtpsl act 0 0" to bring the SS7 links into service.

Running the Console Application

Running the application will result in output to the console showing the operation of the demonstration. The following information is seen:

- Configuration information that is read in
- Messages sent and received
- State transitions during each message exchange
- Results of sending and receiving SMS requests and short messages
- Any errors encountered

The keyboard is also monitored for the following user input:

- **h** — Print a summary of the commands that can be entered to exercise the demonstration.
- **r** — Send an SMS Request message to obtain the SMS address from the other server. The address is saved for use by the SMS short message.
- **n** — Set the application so that subsequent SMS Request Response messages contain an indication that there is no SMS Address available for the subscriber. **Note:** This command must be done on the receiving side.
- **g** — Set the application so that subsequent SMS Request Response messages contain an SMS Address. **Note:** this command must be done on the receiving side.
- **s** — Send a short text message to the SMS address obtained by the SMS request message. **Note:** If no SMS Address has been received, the HomeMsgCtrAddr set in the config.txt file will be used as a default. Enter "*s short_text_message*", where the text message is any combination of 7-bit ASCII characters. The maximum length of the message is 160 characters.
- **q** — Terminate the application and exit cleanly. Selecting Ctrl-C is also an option.

Running the GUI Application

Running the GUI version of the application will open up a new window with the GUI, as shown in the example in Figure 7.

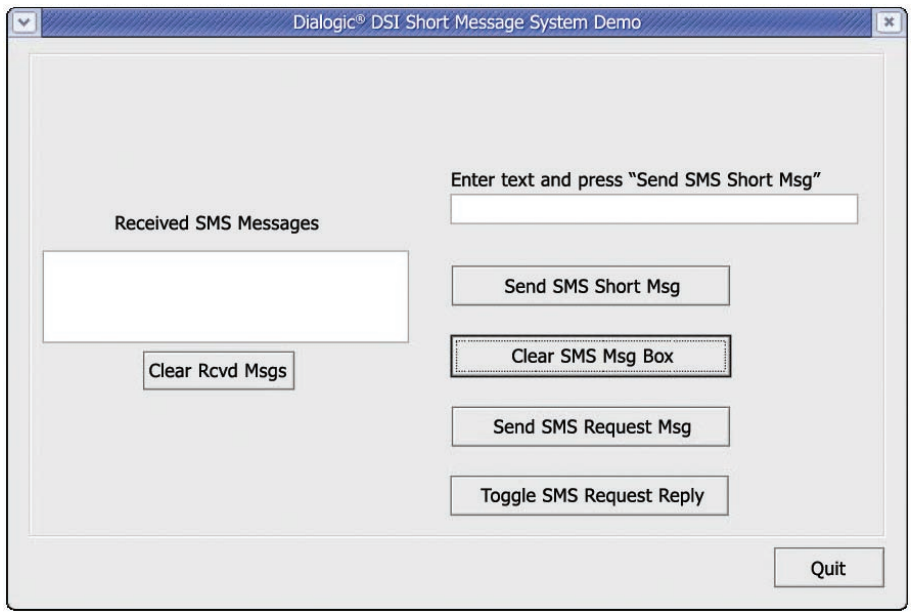

Figure 7. DSI Short Message System Demonstration using QT GUI

The console where the application was started will contain operational logging, but control of the application is done from the GUI.

The GUI buttons and text windows function are as follows:

- **Send SMS Request Msg button** — Sends an SMS Request message to obtain the SMS address from the other server. The address is saved for use by the SMS short message.
- **SMS Short Msg Text Entry window** — Enters short message to be sent. The text message is any combination of 7-bit ASCII characters. The maximum length of the message is 160 characters.
- **Send SMS Short Msg button** — Sends a short text message entered in the text entry window to the SMS address obtained by the SMS request message. **Note**: If no SMS Address has been received, the HomeMsgCtrAddr set in the config file will be used as a default.
- **Received SMS Messages window** — Displays all short messages received by the application. The most recently received message is displayed at the bottom of the window.
- **Clear Rcvd Msgs button** — Clears messages from the Received SMS Messages window.
- **Clear SMS Msg Box button** — Erases text in the SMS Short Msg Text Entry window.
- **Toggle SMS Request Reply button** — Toggles between these two conditions:

  - The application sets subsequent SMS Request Response messages to contain an indication that there is no SMS Address available for the subscriber

  - The application sets subsequent SMS Request Response messages to contain an SMS Address

Note: This command must be done on the receiving side.

- **Quit button** — Terminates the application and exits cleanly. Selecting Ctrl-C is also an option.

For More Information

A Zip file containing the source code and supporting applications can be downloaded at http://www.dialogic.com/goto/?11337

Materials List for Reference SMSMC Application

Software

Dialogic® Distributed Signaling Interface (DSI) Software and Documentation — http://www.Dialogic®.com/support/helpweb/signaling/software3.htm

Dialogic® SS7 Boards - SS7HD Programmer's Manual — http://resource.dialogic.com/telecom/support/ss7/cd/ProductSpecific/SS7HD/Documentation/SS7HD-PM-Iss008.pdf

Dialogic® SS7 Boards Programmer's Manual for SPCI2S and SPCI4 — http://resource.dialogic.com/telecom/support/ss7/cd/ProductSpecific/SPCI-CPM8/ProgrammersManual/u03hsp02.pdf

Dialogic® Programmer's Manuals for SIGTRAN Host Software — http://www.dialogic.com/products/signalingip_ss7components/download/dsi-interface-protocol-stacks.htm

Dialogic® SS7 Protocols Software Environment Programmer's Manual — http://resource.dialogic.com/telecom/support/ss7/cd/GenericInfo/GeneralDocumentation/U10SSS05-SwEnv-PM.pdf

Trolltech open source QT downloads — ftp://ftp.trolltech.com/qt/source/

Boost Library open source downloads — http://www.boost.org/users/download/

Boost Consulting Windows® download — http://www.boostpro.com/products/free

Microsoft® Visual Studio® 2005 — Windows® Development Environment

IS-41 Reference Specification — TAI/EIA IS-41-D

SMSC Introduction/Tutorial — http://www.developershome.com/sms/smsIntro.asp

Hardware

SMS Message Center Server 1

Dialogic® SPCI4S Network Interface Board — http://www.Dialogic®.com/products/signalingip_ss7components/Signaling_Boards_SPC.htm

SMS Message Center Server 2

Dialogic® SS7HDP Network Interface Board — http://www.Dialogic®.com/products/signalingip_ss7components/Signaling_Boards_SS7HDP.htm

Dialogic®
Making Innovation Thrive™

www.dialogic.com

Dialogic Corporation
9800 Cavendish Blvd., 5th floor
Montreal, Quebec
CANADA H4M 2V9

01/09 11336-01