The SOAR Communications Library New (SCLN)

Introduction.

The SOAR Communications Library New (SCLN) provides a set of LabVIEW VIs for implementing TCP/IP connections in a client-server applications environment. Multiple client-server connections are accepted , each acting in parallel and in a non-blocking mode.

Background.

Client-server describes an application architecture in which the “client” requests an action or service from the provider of service, the “server”. Client programs request service from a server by sending it a message (or command message). Server programs process client requests by performing the tasks requested by clients, and informs the client about the result of the action by sending a response message.

Command messages are the responsibility of the users to define. A command message consist of one or more bytes of information. Null commands are not accepted. The SCLN can transport any type of message, being it ASCII or binary information. Internally this is done by preceding the message with a 4 bytes header that contains the length of message information. For LabVIEW applications, this is transparent to the user, but for applications written in other languages, a more detailed description is given in Appendix 1.

The SCLN establishes one to one client-server connections. A server handles only one client, but every application can have several clients and/or servers, depending on their needs. The present implementation of the SCLN imposes a limit of 7 servers and 10 clients per program.

At any given time, there is only one command pending for every client-server connection. The SCLN will not accept another command for that connection until a response to the previous one has been received. The SCLN implementation enforces the discipline of the command-response loop within certain timing constraints: to every command there must be a response within 1500 ms or a timeout condition will be generated. Under this condition, the connection is closed and the client must reconnect to the server again (see the examples below).

Configuration file.

Before using the software, a “configuration” file needs to be prepared with the IP addresses and port numbers of the computers that will run the applications to be connected (a “soar_comms.cfg” file is included in the config folder of the SCLN package). The configuration file is similar to a Windows INI file, consisting of section headers names, enclosed in square brackets ([]), and section entries with parameter information.

The section headers are the name of the "servers" to be utilized in the applications. The entries consist of the server IP address, the client IP address and the port number. Note that a server and a client IP must be given, since the SCLN performs authentication when a client tries to establish a connection. If the IP numbers don't match the ones in the configuration file, the connection is rejected.

Example configuration file:

```
[CommLib]
Version=1.0

[SRV1]
IP_Server=139.229.3.21
IP_Client=139.229.3.20
IP_Port=30040

[SRV2]
IP_Server=139.229.3.21
IP_Client=139.229.3.20
IP_Port=30050

[TCS_OPR]
IP_Server=139.229.15.2
IP_Client=139.229.15.4
IP_Port=5684

[TCS_INS]
IP_Server=139.229.15.2
IP_Client=139.229.15.4
IP_Port=5687

[KERNEL_CMDS]
IP_Server=139.229.15.2
IP_Client=139.229.15.2
IP_Port=8008

[KERNEL_SVC]
IP_Server=139.229.15.2
IP_Client=139.229.15.2
IP_Port=8025

 [DOME]
IP_Server=139.229.15.7
IP_Client=139.229.15.2
IP_Port=5679

[ECS]
IP_Server=139.229.15.5
IP_Client=139.229.15.2
IP_Port=5681
```

Example Use: The Server.

Figure 1 shows the diagram of a server application. Starting on the left, there is a sequence with two frames. Frame 0 contains the SCLN_InitGlobals.vi that initializes the

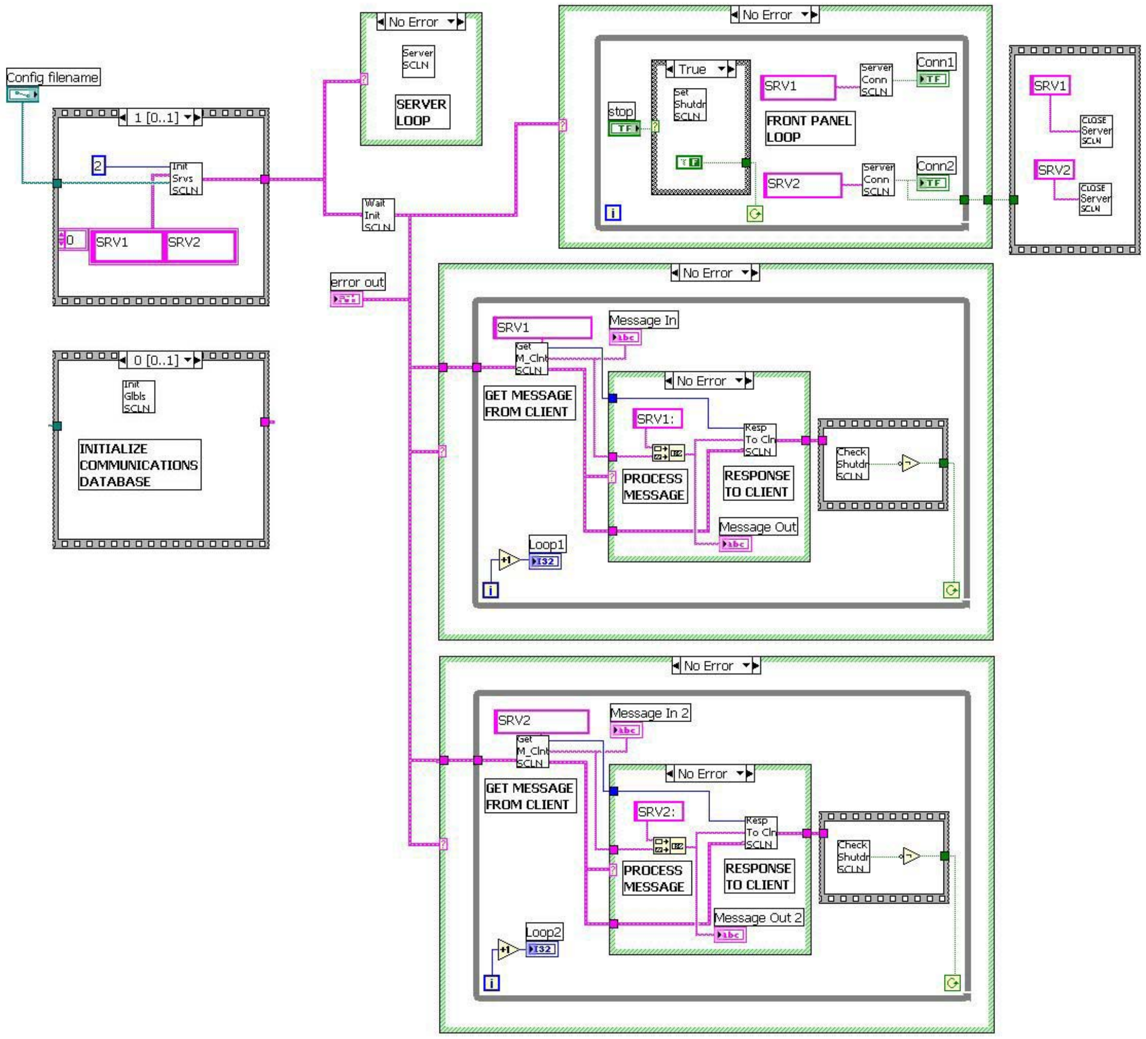

Figure 1. Example Server Diagram

internal communications database. This must be the first VI to be executed in the application.

Frame 1 contains the SCLN_InitServers.vi, that performs the initialization of the servers in this program. It needs a wire connection to the number of servers (2 in this case), to an array with the names of the servers and to the configuration file path. The names of the servers must correspond to the names in the configuration file.

From the sequence you wire the error output to case structures to enable or prevent execution of other parts of the diagram. The top middle case wraps SCLN_Servers.vi that contains the logic to perform the communication with the clients.

The error wire then enters SCLN_WaitInitReady.vi. This VI must be called before any other user code is executed. Output of the VI is an error wire that contains the status of the initialization process.

The top right loop is a typical front panel loop, servicing the stop button and informing the status of the connections. When the stop button is pressed, SCLN_SetShutdownFlag.vi is invoked. This action sets the shutdown flag in the communications database allowing an orderly stop of the application. The status of the connection is extracted with the SCLN_ServerConnected.vi. The name of the server is wired as an input, and the response is a Boolean with true indicating connected. To the right of the loop there is a sequence that is executed when the front panel loop exits. The sequence contains the invocation to the SCLN_CloseServers.vi, that is needed to escape the server loop and to finish the application.

The other two loops corresponds to the user code servicing the commands received from a client. The first VI called is SCLN_GetMessageFromClient.vi where you wire the name of the server you are implementing. The VI returns an index wire that you must connect to SCLN_ResponseToClient.vi along with the actual response to send to the client. In between there is a process message section with the code appropriate for each command. In this example, the code consists of echoing the message concatenated with the name of the server. Upon return from sending the response to the client, the control goes to SCLN_CheckShutdown.vi, that examines the database for the presence of the shutdown flag. If the flag is true, the loop is exited.

Example Use: The Client.

Figure 2 shows the diagram of an example client application. As with the server example, the first vi invoked is SCLN_InitGlobals.vi, that is shown inside the top left sequence frame. From that frame a boolean is wired to the other loops in the application.

The top loop is a typical front panel loop, servicing the stop button and informing the status of the connections. When the stop button is pressed, SCLN_SetShutdownFlag.vi is invoked. This action sets the shutdown flag in the communications database allowing an orderly stop of the application. The status of the connection is extracted with the SCLN_ClientConnected.vi. The name of the server is wired as an input, and the response is a Boolean with true indicating connected.

The other two loops corresponds to the client application logic. The outer loop starts with the client initialization VI: SCLN_InitClient.vi. The server name is wired as an input and the return is an error cluster showing the status of the initialization and connection procedures. If there is no error, the second loop is entered where the exchange of commands is performed. If there is an error in the connection, the client keeps attempting to connect until the shutdown flag is set.

At the center of the internal loop in the example, are shown the two VIs used to exchange messages with a server. To SCLN_MessageToServer.vi you wire the server name and the message to send. The output is an index value that you wire to SCLN_GetResponseFromServer.vi, to receive the response message. After the response, the shutdown flag is checked.

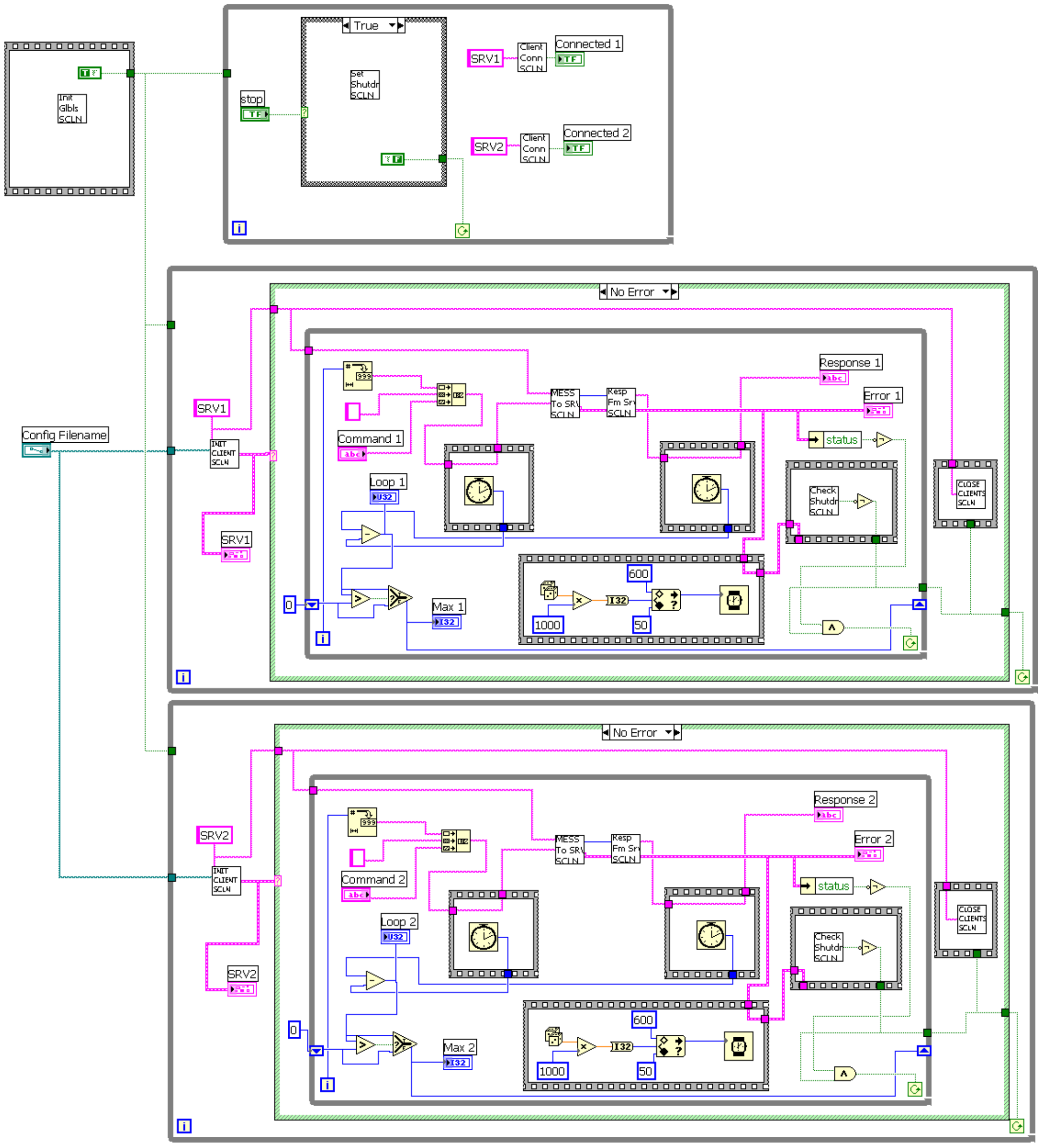

Figure 2. Example Client Diagram

The other sections of the loop deals with message generation and display activities. The message consists of the sequential loop index concatenated to whatever characters where typed in a string control input. Before sending the message, the present tick count is taken, and the same action is repeated after receiving the response. The tick counts are subtracted and the result displayed on a chart. Finally, a random number between 50 and 600 is generated, that serves as the wait time in milliseconds before sending the next command.

Appendix 1: Connecting from non LabVIEW Applications.

The SCLN transport messages between clients and servers, by preceding each message with 4 bytes indicating the length of the message. For non LabVIEW applications it's necessary to extract those 4 bytes when receiving a message and to insert 4 bytes when sending a message. Care must be taken though, because even though the 4 bytes represent an integer number of characters corresponding to the message length, those bytes are rotated due to the way LabVIEW "casts" integers to strings and vice versa. To help illustrate the solution, a C code program (tsTCSconn.c) is included in the SCLN C folder that can be utilized as a template for communicating non LabVIEW applications with the ones using the SCLN.