

Project54 client/server application messaging

W. Thomas Miller, III

This document describes standard procedures for performing remote control of applications, and for getting remote feedback from applications, using Project54 version 2 standard messaging.

Within the document, applications will be classified as either servers or clients. In this context, a server application is an application which has access to information or has control capabilities, either directly by communicating over the IDB bus with a hardware device or indirectly by communications with another server application. A client application gets information or performs control actions by communicating with a server application. For example, the WhelenSerial application (lights) is a server relative to the lights and siren, while the PatrolScreen application (pscreen) is a client relative to the lights and siren. Note that it is quite possible for a single application to be a server relative to one type of information or action and at the same time to be a client relative to some other information or actions.

This document describes how things should be done, but is not intended to imply that all current version 2.x applications support the functionality described. The functionality described was developed in order to support the PatrolScreen. As a result, the PatrolScreen application is currently the best example of a functional client application, while the WhelenSerial, Radio, and various radar applications are fully functional server applications. Clearly, only applications that need to perform client operations need have client functionality. Ideally, however, *every application which communicates directly with a device or which responds to voice commands should support basic server functionality*. It should then be possible to implement new client applications in the future, without having to also modify the server applications.

For reference, a Project54 version 2 application message has a source application name, a destination application name, a message identifier string, and a message body string.

Remote Control

In the simplest terms, client applications implement remote control by sending command messages to server applications, and server applications support remote control by fielding command messages and performing the desired actions. *Every application which responds directly to voice commands should as a minimum support a set of remote commands identical to the commands supported by that application's speech input grammar*. A programmer implementing a client application should be able to examine a server application's grammar file and see much of the remote command functionality without having to look at the server application's code or other documentation.

For example, the grammar for the WhelenSerial application supports voice commands such as “FRONT STROBES”, “REAR STROBES”, “STROBES”, and “STROBES OFF”. A client application should be able to (and, in the case of WhelenSerial, is able to) control the lights and sirens by sending the same command strings in the body of messages sent to the WhelenSerial application. Note that actual speech input messages have the prefix “SPEECHIN”. Thus, the WhelenSerial application will receive a “SPEECHIN STROBES OFF” message if the command comes directly from speech input, but will receive a “STROBES OFF” message if the command comes from another application. Server applications will typically behave somewhat differently in response to speech than in response to remote commands. For the example above, in response to speech input the server application should turn off the strobes and repeat “STROBES OFF” verbally to confirm that the speech input was understood correctly. For a remote command from a client application, it should just turn off the strobes for the client. Any additional visual or verbal communication with the user is the responsibility of the client application.

The speech grammar represents the minimum set of remote commands that a server application should support. It is quite reasonable to support additional remote commands which have no equivalent in speech input.

Remote Feedback

A server application (any application that knows the state of something important) should also provide feedback of state to client applications at their request. A simple model for this interaction has been developed. A client application should send the “FEEDBACK ON” message to a server application to begin receiving feedback information from that server, and should send the “FEEDBACK OFF” message to stop receiving feedback.

A server application should be prepared to send feedback of its complete state information to a client. Feedback messages should start with the prefix “STATUS” and then should contain the information. In the case of on/off type functionality, the status messages should mirror the command messages for simplicity. For example, the messages “STROBES” and “STROBES OFF” are commands to turn the strobes on or off. Similarly the messages “STATUS STROBES” and “STATUS STROBES OFF” reflect the current on/off state of the strobes when providing feedback.

A server can also provide more detailed information in a status message. For example, the message “STATUS TARGET SPEED 102” indicates that the target of the radar reading is going 102 MPH (!!!). The message “STATUS CHANNEL A1 TRP A” indicates that the radio channel display currently shows “A1 TRP A”.

When a server application receives a “FEEDBACK ON” message, it should save the name of the application that sent the message, along with the message identifier string. All subsequent feedback should be sent to that application using the same identifier string (in case the client has placed some significance on the identifier). The server should immediately send a series of status messages reflecting its entire current state.

Subsequently, the server should send unsolicited status messages to the feedback client whenever the state of something changes.

A server should maintain a list of feedback clients. Every server should allow for at least 8 simultaneous feedback clients (this requires storage for at least 8 client names and 8 message identifiers). When a server application receives a “FEEDBACK OFF” message it should remove the application from its list of feedback clients and should thus stop sending feedback to the client.

Under this feedback model, a client application gets all feedback information once feedback is enabled. This may well include STATUS messages that the client is not interested in. The arrival of unexpected and unknown messages to an application should not be treated as an error condition. All applications, clients or servers, should always be written to simply ignore incoming messages that the application doesn’t understand.

Client/Server Startup Synchronization

In a PC installation of the Project54 software system (in possible contrast to PDA based applications) all applications are launched almost simultaneously and immediately start processing on their own threads. Thus, there is no guarantee that when a client application first becomes ready at startup the server application it talks to will also be ready to receive and handle messages.

The application manager supports a simple protocol to assist in application startup synchronization. When applications are initially launched, application manager support for inter-application message passing is disabled. When an application finishes initialization and becomes ready for normal operation, it should send a “STARTUP” message to the application manager. As soon as the application manager has received the “STARTUP” message from all applications (or after a relatively long timeout period) the application manager enables inter-application message passing and sends the “BROADCAST STARTUP” message to all applications. Applications should refrain from sending messages to other applications until after they receive the “BROADCAST STARTUP” message signaling that all applications are ready.

Incoming messages are generally queued by the application manager and by individual applications. After startup, all applications should therefore always be ready to receive messages without message loss.

Clients Should Be Servers Too!

If a client application responds to voice commands or receives feedback information from server applications, it should also implement server functionality, making its own command set and its status information available to other clients. For example, the PatrolScreen application should (but currently does not) support its own grammar as remote commands available to other clients, and it should be prepared to pass on the status feedback it receives to other clients. This maximizes the flexibility of the system architecture at very little cost in coding overhead.