CHAPTER 29

Using Direct Messaging

Introduction

The benefits of client/server applications are proven and many. The primary benefits include providing access to all of the resources on your network as well as maximizing the use of these resources. However, as client/server processing has been adopted and implemented by the business community, additional requirements have emerged.

In today's complex business world, tasks are best accomplished by a series of programs that work together as an application to produce a result. These programs can be spread across multiple computing environments that may or may not be homogeneous.

However, one requirement remains the same whether all of the programs that comprise an application run on a single processor, or each program runs on a separate heterogeneous processor: programs must communicate with each other in order to accomplish the goal of the application. The message facility that is available in SAS software can address all of these needs by using a flexible method of data exchange through messages.

This chapter describes the direct-messaging concept and introduces the messaging services that have been added to SAS to allow you to easily write applications that can communicate with each other on a single processor or across a network. You can develop "thin" client applications that talk to "fat" servers. You can implement applications that perform parallel processing and load balancing.

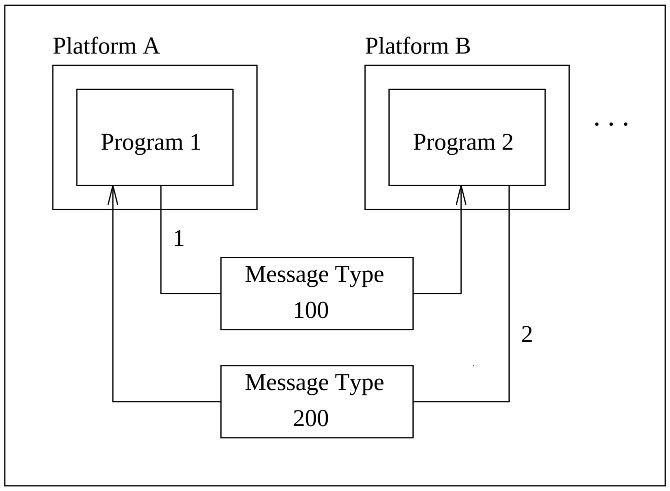
The Direct-Messaging Concept

Typically, one program communicates with another program by directly calling it. This can put unnatural restrictions on your applications that add complexity and hinder the flow of information. Messaging allows applications to communicate by sending each other data in messages. Any action can be taken upon receipt of a message, and acknowledgments can be returned to the sender if and when appropriate.

In its simplest form, messaging requires that both the client and the server portions of the application be active at the same time. This is called *direct messaging*. In other words, the client cannot send a message unless a server is listening for a message.

Figure 29.1 on page 274 illustrates the basic structure of direct-messaging. In this figure, Program 1 sends Program 2 a message with a message type of 100 as shown by path 1. Program 2 receives message type 100 and generates a response with message type 200 that is sent back to Program 1 as shown by path 2. Programs 1 and 2 are shown running on separate platforms. These programs could run on a single platform or separate platforms of the same or of a unique type. Also, any number of programs can communicate simultaneously by using direct-messaging.

Figure 29.1 Basic Structure of Direct-Messaging



The direct-messaging facility allows basic and flexible message construction, transmission, and notification services that span operating system and hardware boundaries across the enterprise. Messages are free-form. Their structure, which is defined by the application developer, may range from a simple collection of variables to complex hierarchies of SCL lists. Additionally, messages may include one or more attachments in the form of SAS data sets or filtered subsets, catalogs or catalog entries, and external files.

Each message contains a message type field. This field is used to define the set of message types that are meaningful to a particular program. When a program receives a message that has a known message type, it knows the layout of the data that is contained in the message body, and it can take the appropriate action based on the values of the data.

Direct-Messaging Benefits

Messaging enables application developers to deploy multi-tiered distributed applications. This multi-tiered design allows you to separate and centralize business and data access to the server portion of the application. You can then implement a thin client application that requires little or no maintenance. Not only is it easy to segment your logic into individual programs, but these programs can execute on the host that best meets your data and resource requirements.

To illustrate these benefits consider a three-tiered implementation of a business application. The first tier could be the thin client piece which is a graphical user interface. The middle tier would then contain the business logic that is needed to manipulate data and to produce information. The third tier would perform the data access logic that is necessary to read or to write the data source.

Any piece of this application could be modified without changing the other tiers of the application. For example, the data source could change from a DB2 database to an ORACLE database and only the third tier (the data access logic) would need to be changed.

The SAS System now provides an SCL interface to direct-messaging that allows you to develop integrated SAS/AF and FRAME applications that can communicate through a basic yet flexible interface. In addition, the TCP/IP access method is the only access method that supports direct-messaging.

Application Design

When designing a direct-messaging application, the client and server section must be choreographed so that they do not block one another. That is, you want to make sure the client and server applications are completely clear as to what to expect from each other.

A typical *server* application using direct-messaging would execute the following steps:

1. Initialize the messaging environment.
2. Listen for any messages.
3. Respond to messages when required.
4. Repeat steps 2 and 3 as needed.
5. Shut down the messaging environment.

A typical *client* application using direct-messaging would execute the following steps:

1. Initialize the messaging environment.
2. Establish communication with a specific server.
3. Send messages.
4. Process any responses from the server.
5. Repeat steps 3 and 4 as needed.
6. Disconnect from the server.
7. Shut down the messaging environment.

To accomplish the above steps, direct messaging provides two SCL classes (STATION and CNCTION) and a set of SCL methods for each class. When designing a direct messaging application, the application programmer must choreograph the client and server portions carefully. Using direct messaging, the SCL _query method will block until an incoming event is received. Also, the _send method will block if the receiving side is not listening for any messages.

You must specify a nickname (moniker) in any SAS server session (the session to which applications will connect). This value is then used by any client session in order to locate the server when connecting by using the CONNECT method.

The syntax for specifying the moniker is:

```
%let _moniker=protocol//network_node/service_name;
```

If you omit *protocol*, the moniker uses the default access method for that host. If you omit *network_node*, the moniker defaults to the local node. For example, by specifying the following in a SAS session that runs on a node named MYNODE

```
%let _moniker=/shr1;
```

you would default to the TCP/IP access method on the network node MYNODE, which uses the service name SHR1. The moniker can be initialized anywhere in the SAS application, prior to calling the direct-messaging methods.

Note: TCP/IP is the only access method that supports distributed messaging. △

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/ CONNECT User's Guide, Version 8*, Cary, NC: SAS Institute Inc., 1999. pp. 537.

SAS/CONNECT User's Guide, Version 8

ISBN 1–58025–477–2

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, September 1999