CHAPTER

31

Using Indirect 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.

Often the programs that make up an application need to run on their own schedules, independent of the other programs. Also, as applications become more distributed, businesses will strive to simplify their networks and to minimize the number of direct connections that must be maintained and re-started in the event of a network failure.

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 and queuing facilities that are available in SAS software can address all of these needs by using a flexible method of data exchange through messages.

This chapter describes the indirect-messaging (message queuing) 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.

You can even implement applications that communicate asynchronously with each other. That is, one application could send messages to one or more target applications that may not be currently running and that may not run for several more hours or days. These services are extremely adaptable which can minimize the cost of restructuring your applications to meet your ever-changing business needs.

The Indirect-Messaging (Queuing) Concept

In some instances, you do not want the programs that make up your application to run at the same time or to be synchronized so that one side sends a message and waits for a reply before it can send another message. These restrictions disappear with indirect-messaging (queuing).

SAS *message queuing* enables programs to communicate indirectly by placing messages on queues in storage. Therefore, the pieces of your application can run independently of each other, can run at different speeds and times, and can run without a direct connection between them.

SAS message queues provide a basic and logical means of communication. Programs communicate indirectly by delivering messages to queues and by fetching from or browsing messages in queues. The programs are commonly referred to as store-and-forward applications. The message is stored on a queue by a program and forwarded to one or more programs at a later time. The message queues are administered by a collection manager and a queue manager.

The *collection manager* is responsible for managing "collections" of queues and starting the queue manager that processes the individual messages for each queue. A "collection" allows you to group queues together so they are managed by the same collection manager. Each queue must be uniquely associated with a collection, and your application may contain multiple collections.

The *queue manager* is a server process that is responsible for

- allocating the queues.
- maintaining access information for each of the queues.
- administering the messages that belong to each queue.

For more information on the collection manager and queue manager, see Chapter 38, “The DOMAIN Server,” on page 411.

Queues can be predefined or dynamically created during an open operation. Predefined queues are permanent queues and will remain available until they are explicitly deleted by an administrator using PROC ADMIN. Permanent queues may also be defined to maintain *message persistency*. That is, a queue's messages will be stored so that they are available even if the queue manager re-starts. Dynamic queues can be opened as either permanent or temporary. A temporary queue is deleted automatically when the queue is closed.

Figure 31.1 on page 283 illustrates the basic structure of the SAS indirect messaging communication. In this figure,

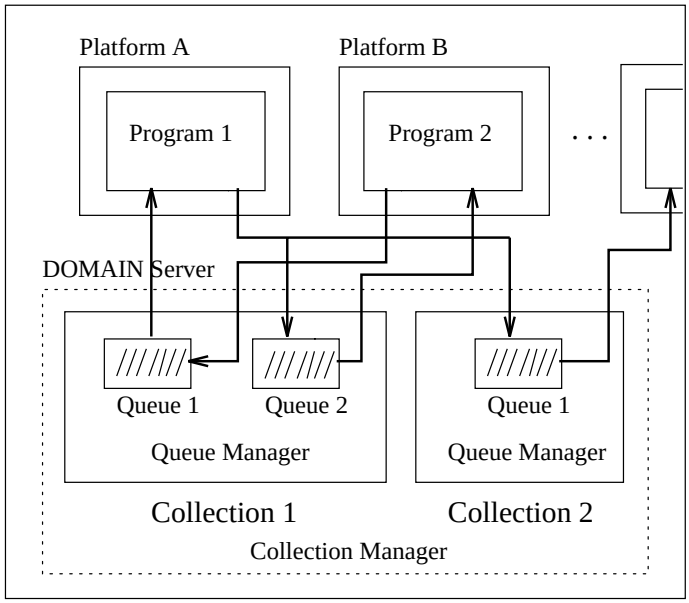
- Program1 receives messages from Queue1 and writes messages to Queue2 in Collection1 and Queue1 in Collection2.
- Program2 receives messages from Queue2 and writes messages to Queue1 in Collection1.
- Queue1 in Collection2 is used by other programs in the application.

Note: Collection1 and Collection2 each have a queue called Queue1; however, these are *not* the same queue. Each Queue1 represents a queue that is unique within its collection. Collection names must be unique within your SAS session. △

The ellipses (...) in the figure indicate the ability to have multiple programs communicating by using multiple queues. Also, this figure shows how the programs and the DOMAIN server can each be executing on a different platform. This is only one possibility; they can also execute each on a different platform, all on the same platform, or any combination in between.

It should also be noted that multiple programs can read or write from the same queue. You do not need a separate queue for each program.

Figure 31.1 Basic Structure of Indirect Messaging



Programs can be developed to communicate in either of two modes: one-way (datagram) or two-way (reply). In a datagram mode of operation, Program1 puts a message on a queue but does not expect a reply response. This is illustrated in the figure above by Program1 and Queue1 in Collection2.

In a reply mode, Program1 puts a message on a queue and expects a reply message to be sent to a designated reply-to-queue by Program2, after Program2 receives the original message. This is illustrated in the indirect-messaging figure using Collection1 by

1. Program1 sending an initial message to Queue2.
2. Program2 fetching this message from Queue2.
3. Program2 sending a reply back to Queue1.
4. Program1 fetching the reply from Queue1.

It is important to note that Program1 and Program2 are communicating without a direct connection between them. Therefore, they are not required to run at the same time or at the same speed. The target program could be busy when a message is put in its queue. In fact, the target program may not run for hours or days after messages for it have been put on its queue. You have complete freedom to schedule the pieces of your application based on your business requirements.

The communication between programs that use the SAS message-queuing facility can be one-to-one, one-to-many, many-to-one, or any combination of these to provide you with complete flexibility in the structure of your application. These structural combinations can be used with the datagram and reply message-flow modes, as previously discussed.

- In a one-to-one relationship, a client sends messages to a queue, and the receiving program retrieves the messages in a time frame that is dictated by your business needs.
- In a one-to-many relationship, a single client could send messages to a queue that is serviced by the same program that runs on multiple platforms to provide load

balancing. Another scenario would be for a single client to send messages to a queue that is serviced by multiple subtasks of a program that can run concurrently to provide parallel processing.

- In a many-to-one relationship, you could have multiple clients that send messages to a queue that is being serviced by a single server. The clients could run independently of the server's speed and would never need a direct connection between any of the clients and the server.

With all of these relationships, the receiving program can optionally generate replies that are based on the messages that it retrieves.

A queue can be defined as permanent or temporary. Permanent queues remain until they are explicity deleted, while temporary queues are implicitly deleted when closed. Permanent queues may either contain persistent or non-persistent messages. Persistent messages are stored on disk, and therefore guaranteed to persist through queue open and close boundaries, as well as a queue server process shutdown (DOMAIN Server stop and re-start).

A message is deleted from a temporary queue after it is fetched by an application, after the queue is closed, or upon a queue server process shutdown. A message sent to a permanent, persistent queue is stored on disk for retrieval by any number of applications and remains intact in the event of a queue server re-start.

Messages in a permanent, persistent queue are deleted only when fetched from the queue. This guarantees that a message in a permanent queue will remain there for any number of applications to browse, will persist through queue open/close boundaries, and will be removed from the queue only when it is fetched from the queue.

Indirect-Messaging (Queuing) Benefits

There are many benefits to using the SAS message queuing facility for implementing your distributed applications. The following paragraphs present several benefits and you may think of others as you visualize your distributed applications implemented with SAS message queues.

As is the case with applications that are currently developed with SAS, an application developed with the message-queuing facility is completely portable. There are two interfaces available for using SAS message queues:

- an SCL interface
- a functional interface for use through a SAS DATA step or a SAS macro.

In addition, the TCP/IP access method is the only access method that supports indirect messaging.

Because the message-queuing facility is completely integrated with the SAS System, you continue to have the same portability that you expect from your SAS applications.

A significant benefit of using SAS message-queuing for your distributed applications is that the communicating programs run independently of each other in respect to time. This indirect mode of communication has several positive results. Because the programs communicate indirectly by using message queues, each program is completely removed from the interface of any other program. Therefore, an individual program could be modified to execute different logic that is based on message receipt. It could be moved to execute on a different platform, or it could be rescheduled to run at a different time and absolutely none of these things would require any changes to the other programs that make up the application. Also, individual programs could be added or deleted without any disruption to the overall application.

Another benefit of the ability to run communicating programs independently is that there is never a direct link between them. As an illustration, a program that needs to

send a message to a queue connects to the queue, sends one or more messages to the queue, retrieves responses if appropriate, and then disconnects. Connections are not left idle while one program waits for a response from another. This helps to minimize the number of active connections that need to be maintained in the network and it facilitates network re-start in case of failure.

The structure of the SAS message-queuing facility insulates application developers from the details of the network. The queue manager is solely responsible for maintaining the queues and for ensuring that the messages in the queues reach their destination when requested and are not lost.

The queue manager is also responsible for establishing the information that is needed by the network protocols that are used to transmit the messages to and from the queues. Because the applications programmer is not distracted by the networking details, attention can be focused solely on the business needs and the application logic that is necessary to meet these needs. The more time and thought that can be given to the business algorithm and the flow of data, the faster and better the application becomes at delivering the necessary information.

A general rule of thumb for writing distributed applications is to "keep the logic as close to the data source as possible in order to minimize network traffic and the cost of client/server computing". Because the SAS message-queuing facility allows all combinations of one-to-one, one-to-many, and many-to-one application structure as well as datagram and reply modes of communication, you have total flexibility with the structure of your distributed application and the ability to minimize the cost of your client/server computing.

Application Design

With indirect messaging, two or more applications communicate with each other indirectly using message queues. Therefore, the applications do not have to be running at the same time. The data is sent in the form of a message and is saved on a message queue until the receiving application is ready to fetch it. As a precaution, messages are written to disk. Therefore even if your queue terminates, the application can retrieve its messages.

A typical *server* application that uses indirect messaging would execute the following steps:

1. Initialize the messaging environment.
2. Check queue(s) 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 that uses indirect messaging would execute the following steps:

1. Initialize the messaging environment.
2. Locate the collection manager.
3. Establish communication with queue(s).
4. Send or receive messages from an opened queue(s).
5. Repeat steps 3 and 4 as desired.
6. Release queue(s).
7. Shut down the messaging environment.

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