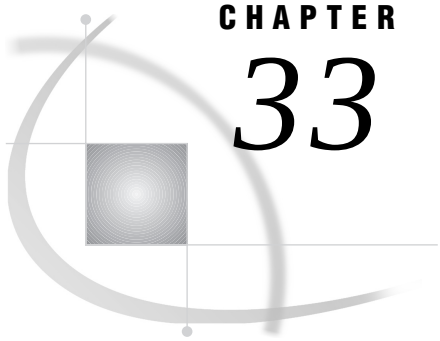

CHAPTER

33

An Example of Combining Direct and Indirect Messaging

Introduction

The following sections of code are part of a sample store-and-forward application called "The Application Development Manager (ADM)" that was developed by using the SAS System and its direct messaging and queuing facilities. The ADM is designed to facilitate application development in a work group by allowing a restricted number of people to work on a shared set of SAS catalogs. The ADM server maintains a single centralized copies of the application catalogs and allows users to check-out and check-in individual entries.

A message is broadcast to all members of the work group to notify them whenever an updated entry is checked back into the central catalogs or a new entry is added to the central catalogs. They have the option of receiving any number of the updated entries so that they can be assured of running the most current version of the application. Because the notifications are sent to message queues, the members of the work group can also choose to request the entries at the time of the notification or at any time in the future.

The following actions are available to the users of this application:

register
: add your identity to the work group that is defined for a specific application. A broadcast queue is also created for you as part of the registration.

check-out
: get a copy of a specific entry from the central catalogs and put it into your private work area for development

check-in
: return an updated entry to the central catalogs. The entry is updated in its central catalog, and a message about the updated entry is broadcast to each of the work group members' queues.

add
: add a new entry to a central catalog. The entry is updated in the central catalog, and a message about the new entry is broadcast to each of the work group members' queues.

query
: query your message queue for any outstanding messages.

receive
receive one or more new or updated entries from the central catalogs.

purge
delete any broadcast messages that you may have accumulated.

de-register
remove yourself from the work group that is associated with a specific application.

To implement the ADM application, each of the above actions is mapped to a message type. In order to perform an action, the client sends the ADM server a message. The message contains a message type that represents an action, as well as any information needed by the ADM server to process the requested action.

The following sections contain code segments to illustrate the implementation of the check-out, query, and check-in actions. Initialization code and error checking have been taken out of these code segments to draw attention to the messaging interface.

Check Out a Catalog Entry

The check-out action involves a user making a request for a specific entry and receiving an immediate response. The response is either the requested entry or a message indicating why the entry cannot be returned. Direct-messaging was chosen for this portion of the application because the user needs an immediate response to the check-out request in order to continue working.

The following code, which was taken from the client portion of the ADM application, allows a user to check out a specified catalog entry. The application was written to send a message type of 30 to the ADM server in order to request that a specific catalog entry be checked out by the user. The ADM server will respond with one of two message types. A message type of 35 has been defined to mean that the requested catalog entry is available. In this case, the entry is received and written to the location specified by the user. A message type of 39 has been defined to mean that the requested catalog entry is locked to another user in the work group. In this case, the client application prints a message and control returns to the main menu.

The following variables would be initialized prior to the code fragment:

srvname
supplied to the _open method when opening a client station. This variable contains the service name of the server portion of the ADM application.

uname
supplied to the _send method. This variable contains the userid of the client and will be sent as the first parameter in the message.

incat
supplied to the _send method. This variable contains the four-level entryname in the central catalogs that are controlled by the server portion of the ADM application. It is the second parameter in the message.

outcat
used to build the TLIST parameter that is supplied to the _acceptAttachment method. This variable contains the four-level entryname of the location to which the received entry will be written.

Note: The following code is a portion of an SCL program that is used for illustration purposes only. It is not a complete program. △

```
/*****************************************/
/*                                       */
```

```
   /* CHECK-OUT A CATALOG ENTRY              */
   /*                                        */
   /******************************************/

❶
stationid = loadclass('sashelp.connect.station');
station = instance(stationid);
call send(station, '_open', "TESTPTP", rc);

❷
cnctionid = loadclass('sashelp.connect.cnction');
cobj = instance(cnctionid);
call send(cobj, '_open', station, srvname, rc);

❸
call send(cobj, '_send', 30, 0, alist, rc,
   uname, incat);

❹
call send(cobj, '_query', eventtype,
   msgtype, 0, alist, rc);

❺
if msgtype = 39 then do;
   call send(cobj, '_recv', rc, str);
   put 'NOTE: ' str;
end;

❻
else if msgtype = 35 then do;
   call send(cobj, '_recv', rc, str );

❼
   catlist = makelist();
   catlist = getiteml(alist,1);
   outlib = scan(outcat,1,'.');
   outcat = scan(outcat,2,'.');

❽
   rc = setnitemc(catlist, outlib, "OUTLIB");
   rc = setnitemc(catlist, outcat, "OUT");
   tlist= makelist();
   catlist = insertl(tlist,catlist,-1);
   call send(cobj, '_acceptAttachment',
      tlist, rc, "COMPLETE");
   put 'NOTE: Requested entry has been received.';
end;

❾
else do;
   put 'ERROR:  Received unknown msgtype--'msgtype;
   call send(cobj, '_recv', rc);
end;
```

❶ Open a station for the client.

❷ Connect to the ADM server.
❸ Send the ADM server a check-out request and include the userid and the desired entryname.
❹ Wait for a response from the ADM server.
❺ A message type of 39 means that the entry is already checked out.
❻ A message type of 35 means that the entry is available.
❼ Prepare to receive the requested entry.
❽ The OUTLIB and OUT named items are used by the _acceptAttachment method to determine where to write the checked-out entry.
❾ Should only receive msgtype 39 (fail) or 35 (success). All other msgtypes are unknown.

Query for Broadcast Messages

As the members of the work group modify their private copy of the application, they may choose to update their copy by merging any pieces that have been added or modified by others in the work group. Different members may have different schedules for updating their private copy. For example, developers may need adhoc updates that occur when they reach certain points in their development. Testers may require regular updates based on their testing schedules. To meet the individual needs of each member in the work group, message-queuing is used for this portion of the application.

The following code, which was taken from the client portion of the ADM application, allows users to query their queues for existing messages. This portion of the application has been coded to recognize that a message type of 100 means a broadcast message about a new or modified catalog entry type. These messages are broadcast from the ADM server portion of the application to notify users of new or updated catalog entries.

The message contains a status field that indicates whether the entry was added or modified and the name of the entry that was added or modified. If the entry is a new entry, the message also contains the name of the developer who made the addition. If the queue contains one or more messages, the messages are fetched one at a time and a list of entries is constructed. The new or updated entries can then be selected from the list, fetched from the ADM server, and merged into the user's existing catalog.

The following variable would be initialized prior to the code fragment:

qname — supplied to the _open method. This variable contains the name of the broadcast queue that will be queried for messages.

Note: The following code is a portion of an SCL program that is used for illustration purposes only. It is not a complete program. △

```
/*****************************************/
/*                                       */
/*      QUERY FOR BROADCAST MESSAGES     */
/*                                       */
/*****************************************/

❶
stationid = loadclass('sashelp.connect.station');
qstation = instance(stationid);
call send(qstation, '_open', "TESTQM", rc);

❷
queueid = loadclass('sashelp.connect.queue');
```

```
queue = instance(queueid);
call send(queue, '_open', qstation, qname,
   "FETCH", qrc, "POLL");

❸
rc = clearlist(alist);
hlist = makelist();
call send(queue, '_query', eventtype,
   msgtype, hlist, alist, qrc);

❹
do while(eventtype ='DELIVERY');

❺
   select(msgtype);
   when(100)

❻
      call send(queue, '_getfield' ,parms,
         qrc, status);
      if upcase(status) = 'CREATE' then do;
         call send(queue, '_getfield', parms,
            qrc, catname, developer);
         call display("testdata.sapp.addlist.scl",
            catname, developer);
      end;  /* if create */
      else do;
         call send(queue,'_getfield',parms,
            qrc,catname);
         call display("testdata.sapp.addlist.scl",
            catname);
      end;  /* else update */

   otherwise
      call send(queue,'_recv',qrc);
      put 'Unknown message type: ' msgtype;
      put 'Message discarded.';
   end; /* select */

❼
   rc = clearlist(alist);
   rc = clearlist(hlist);
   call send(queue, '_query', eventtype,
      msgtype, hlist, alist,qrc);
end; /* do while */
```

❶ Create a station for the client.

❷ Access the client's queue.

❸ The OPEN method was successful; are there any messages for the client to fetch?

❹ DELIVERY indicates that the client has messages; stay in the loop as long as messages are found.

❺ Message type (msgtype) 100 means there are new or updated entries.

❻ Check if the status is "create", else it must be "update".

❼ Fetch the next message, if any.

Check In a Catalog Entry

The check-in action involves a user sending an entry to the ADM server portion of the application to be merged into the central copy of the catalog. When an entry is checked in, immediate confirmation of the receipt of that entry is needed. Therefore, direct messaging is used for this portion of the application. The result of checking in an entry is for the ADM server to broadcast notification to everyone in the work group so that they can request the updated entries at their convenience. Message queuing is used by the ADM server for the broadcast.

The following code, which was taken from the client portion of the ADM application, allows a user to check in a specified catalog entry. The client program was written to send a message type of 40 to the ADM server program to indicate a catalog entry check in. If the ADM server portion of the application has the catalog entry listed as checked out by this user, it will check in the entry and broadcast a message with a message type of 100. This notifies the users in the work group of the updated entry.

The following variables would be initialized prior to the code fragment:

srvname
: supplied to the _open method when opening a client station. This variable contains the service name of the server portion of the ADM application.

uname
: supplied to the _send method. This variable contains the userid of the client and will become the first parameter in the message.

incat
: used to build the ALIST parameter that is supplied to the _send method. This variable contains the four-level entry name of the private entry to be checked into the central catalogs that are controlled by the server portion of the ADM application.

outcat
: supplied to the _send method. This variable contains the four-level entry name of the central copy of the entry that is controlled by the server portion of the ADM application. It is the second parameter in the message.

Note: The following code is a portion of an SCL program that is used for illustration purposes only. It is not a complete program. △

```
/*****************************************/
/*                                       */
/*       CHECK IN A CATALOG ENTRY        */
/*                                       */
/*****************************************/

❶
stationid = loadclass('sashelp.connect.station');
station = instance(stationid);
call send(station, '_open', "TESTPTP", rc);

❷
cnctionid = loadclass('sashelp.connect.cnction');
cobj = instance(cnctionid);
call send(cobj, '_open', station, srvname,
   rc);

❸
```

```
lib = scan(incat,1,'.');
cat = scan(incat,2,'.');
ename = scan(incat,3,'.');
etype = scan(incat,4,'.');
catlist = makelist();

❹
rc = setnitemc(catlist, "CATALOG", "TYPE");
rc = setnitemc(catlist, cat, "MEMNAME");
rc = setnitemc(catlist, lib, "LIBNAME");
entry = trim(left(ename)) || '.' ||
   trim(left(etype));
rc = setnitemc(catlist, entry, "SELECT");
alist = insertl(alist,catlist,-1);

❺
call send(cobj, '_send', 40, 0, alist, rc,
   uname, outcat);
if rc ne 0 then
   put 'send failed';
else do;
   rc=clearlist(alist);

❻
   call send(cobj, '_query', eventtype,
      msgtype, 0, alist, rc );
   if rc ne 0 then do;
      str = sysmsg();
      put 'QUERY for response failed' str;
   end;
   call send(cobj, '_recv', rc, str);
   put 'Received message = ' str;
end;

call send(cobj,'_disconnect',rc);
call send(station, '_close', rc);
return;
```

❶ Open a station for the client.

❷ Connect to the ADM server.

❸ Prepare to check in a specific entry to the ADM server.

❹ The TYPE, MEMNAME, and LIBNAME named items must be set so that the _send will know which catalog to send. The SELECT named item will be used to specify a single entry for the attachment.

❺ Send the catalog entry to the ADM server.

❻ If the ADM server did not accept the client's entry successfully, find out why.

Server Portion of the ADM Application

The following pseudo code is taken from the server portion of the ADM application. The server portion of this application is one main loop that both continuously receives messages with message types that have been defined to this application and performs

the appropriate action based on the message type. The server portion of the application receives its messages using direct messaging and sends messages using both direct messaging and message queuing.

The majority of the logic is contained in the server portion of the ADM application. This has the benefit of being able to modify the server portion of this application (either changing or adding functionality) without having to update the user portions of this application.

```
   /*****************************************/
   /*  Main Server Loop                     */
   /*****************************************/
do while (finish='N');
   call send(station, '_query', eventtype,
      msgtype, 0, alist, cobj, rc);
   if rc eq 0 then do;
      put 'Eventtype is ' eventtype;
      if eventtype = 'DISCONNECT' then do;
         put 'Client ' cobj ' is
            disconnecting from the server.';
      end;
   else do;
      if eventtype = 'CONNECT' then
         put 'Client ' cobj ' has connected
            to server.';
      else do;
         put '  ----  client is ' cobj;
         put 'Msgtype is ' msgtype;
         put 'Receiving message from client';
         select(msgtype);
            when(999);
               /*  Stop server message.      */
               finish='Y';
            when(10);
               /* User wants to register.    */
               /* Create qname based on      */
               /* userid.                    */
            when(20);
               /* User wants to de-register. */
            when(30);
               /* User wants to check out    */
               /* an entry; send the entry   */
               /* as an attachment.          */
            when(40);
               /* User wants to check in a   */
               /* catalog entry. Accept      */
               /* updated entry as an        */
               /* attachment and broadcast   */
               /* the updated entries to     */
               /* all registered users.      */
            when(50);
               /* User wants to add a new    */
               /* entry. Accept the new      */
               /* entry as an attachment     */
               /* and broadcast the new      */
               /* entry to all registered    */
```

```
            /* users.                    */
           when(60);
            /* User wants to receive new  */
            /* or updated entries; send   */
            /* entries as attachments.    */
           otherwise;
            /* Otherwise, unknown         */
            /* message type.              */
        end;  /* select msgtype */
     end;     /* else not disconnect */
  end;        /* if rc */
end;          /* do while */
```

The addition of messaging and queuing to the existing SAS System client/server toolset provides an integrated solution to your most complex client/server application needs. The logical flow of information in messages that underlie the SAS direct messaging facility allows you to layer your distributed applications to best meet your business needs.

The flexibility of the indirect communication that underlies the SAS message queuing facility allows you to implement, deploy, modify, and schedule the various programs that make up your applications independently and more efficiently.

Used together, direct messaging and indirect messaging allow you to minimize the cost of implementing and running your client/server applications by allowing your data sources, hardware resources, and information goals to dictate the logic structure, the platforms, and the schedules on which to run the programs that comprise your applications.

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