CHAPTER 11 Examples That Use Remote Library Services (RLS)

Example 1. RLS: Accessing Remote Data to Print a List of Reports

Purpose

The following example uses RLS to access a small portion of the data that exists in a remote SAS data set, in order to print a list of the reports that are being requested by the local workstation. This is a good use of RLS, provided the data set REPORTS.REQUEST has a small number of observations.

Program

```
signon rempc;
```

1

```
libname reports REMOTE 'd:\prod\reports'
   server=rempc;
   data _null_;
   set reports.request;
```

```
      if (copy = "Y") then do;
         put "Report " report_name
            " has been requested";
      end;
```

❶ Define a remote library to a local session. The value for SERVER= is the same as the remote session id that is used in the SIGNON statement.

Example 2. RLS: Accessing Remote Data by Using the WHERE Statement

Purpose

In this example, WHERE statement processing modifies the previous example in order to reduce the amount of data that is being requested and the impact on network traffic. The WHERE statement moves to local processing only those observations for which a report is being requested. This move is more efficient than moving every observation to local processing and checking the COPY variable for a **Y** value.

Program

```
   signon rempc;

❶
   libname reports 'd:\prod\reports'
      server=rempc;

❷
   data _null_;
      set reports.request;
      where copy = "Y";
      put "Report " report_name
         " has been requested";
   end;
```

❶ Define a remote library to a local session.
❷ Use the WHERE statement to filter unneeded observations.

Example 3. RLS: Updating Remote Data

Purpose

This example enables you to take advantage of the mainframe's superior data handling and security features, while you work in a user-friendly GUI environment.

RLS is used to update remote data. This application of RLS eliminates the need to transfer a disk copy of the data to the local system before processing the data. It also involves low volume, transaction processing.

Program

```
signon remos390;

❶
libname rlib REMOTE 'hrs.emp.data'
   server=remos390;

❷
proc fsedit data=rlib.employee;
run;
```

❶ Define the remote human resource library to the local SAS session.

❷ Execute a local FSEDIT to update the employee data set that exists on the OS/390 host.

Example 4. RLS: An SCL Program That Uses the WHERE Statement

Purpose

This example is an excerpt from an SCL program that uses RLS to query a remote reservation database. Reservations are selected based on the value that is stored in the variable RESNUM. The use of the WHERE clause in this example is important because the WHERE clause is applied in the remote session before any data is transferred. As a result, only the observations that meet the criteria are moved to the local session.

This example is a good use of RLS because (as in the previous example) it involves transaction-type processing and enables the local GUI to be used for data entry on the selected observations in the database.

However, if you were to use the SCL LOCATEC function, every observation would be transferred to the local session and compared against the specified criteria. The response time in this case would be poor, at best. These alternative programming choices emphasize the importance of being aware of the amount of data that local processing is requesting and minimizing this amount when using RLS.

Program

```
signon os390;

libname master REMOTE "hq.prod.data"
```

```
   server=os390;

❶
rdsid = open("master.reserv", 'u');

❷
wherecls="resnum=" || "'" || resnum || "'";
rc = where(rdsid, wherecls);
call set(rdsid);
rc = fetchobs(rdsid, 1);
```

❶ Open the remote Headquarters database.

❷ Build and apply the WHERE clause to speed up retrieval.

Example 5. RLS: Updating a Remote Data Set by Applying a Local Transaction Data Set

Purpose

In cases where data must be kept current and the number of updates that you need to perform is small, RLS can be used efficiently between a local and a remote host. RLS enables you to perform a local update to a remote data set.

This example creates a data set remotely by remotely submitting a DATA step. Next, it creates a local transaction data set. Using RLS, it assigns a local LIBNAME to the remote library. Finally, the program modifies the remote data set with the local transactions.

Program

```
   signon;
   rsubmit;
❶     data sasuser.my_budget;
       length category $ 9;
       input category $ balance;
       format balance dollar10.2;
       datalines;
   utilities   500
   mortgage   8000
   telephone  1000
   food       3000;
   run;

   endrsubmit;

❷ data bills;
       length category $ 9;
       input category $ bill_amount;
       datalines;
   utilities    45.83
   mortgage    649.95
   food         68.21;
   run;

❸ libname rlslib slibref=sasuser
      server=&rsession;

❹ data rlslib.my_budget;
      modify rlslib.my_budget bills;
      by category;
      balance=balance-bill_amount;
   run;

❺ data _null_;
      set rlslib.my_budget;
      put 'Balance for ' category  @25
         'is: ' balance;
   run;

❻ signoff;
```

❶ Create the master data set MY_BUDGET in the library SASUSER in the remote session.

❷ Create a local or work transaction data set for updating the remote data set MY_BUDGET.

❸ Assign a local library to the library SASUSER in the remote session.

❹ Apply the transaction data set to the remote data set MY_BUDGET.

❺ Review the results. All items except TELEPHONE will be updated.

❻ Sign off from the remote host. The libref RLSLIB is deassigned as part of the sign-off processing.

Example 6. RLS: Subsetting Remote Data for Local Processing and Display

Purpose

If the amount of data that is needed for a processing job is small, RLS is an efficient way to gather current data on a remote host for local processing and display. This program subsets the data on the remote host so that only the data you need is transferred. This method saves computing resources on the remote machine and diminishes network traffic while it gives you access to the most current data.

In this example, a large reservations database exists on a remote UNIX platform. Several local procedures need to be run against a small subset of the data that is contained in the master reservations database. This situation is ideal for RLS.

The LIBNAME statement is issued in the local SAS session to define the remote library that contains the data set RESERVC. The PROC SORT statement sorts the remote data set and writes the subset data to the local disk.

The WHERE= and KEEP= options are specified in the PROC SORT statement to reduce the amount of data that moves through the network to local processing. Only the data that meets the WHERE= and KEEP= criteria is moved across the network to the local session.

PROC SORT creates the subset data set on the local machine and allows all subsequent processing to run on the local machine without further remote CPU consumption. PROC SUMMARY and PROC REPORT summarize and format the local data so that it can be displayed to the user by using the NOTEPAD command.

Program

```
  init:
  submit continue;
❶  libname remlib '/u/user1/reservations'
      server=srv1;

❷  proc sort data=
      remlib.reservc(keep=company origin
      where=(origin='ATLANTA'))
      out=tmp;
      by company;
   run;

❸  proc summary data=tmp
      vardef=n noprint;
      by company;
      output out=tmp2;
   run;

❹  proc printto new print=work.view.report.source;
   run;

   proc report ls=74 ps=85 split=
   "/" HEADLINE HEADSKIP CENTER NOWD;
   column
     ("Totals" "" "" "" company _freq_);
   define company / group format=$40.
      width=40 spacing=2 left "Company";
   define _freq_ / sum width=14
      spacing=2 right "# Reservations";
   rbreak after /ol dul skip summarize
      color=cyan;
   run;

   proc  printto print=print;
   run;
endsubmit;

❺ call execcmdi('notepad work.view.report.source;
      color back blue;');

   _status_='H';
   return;
```

```
main:
return;

term:
return;
```

❶ Submit local LIBNAME statement to define the remote library.

❷ PROC SORT runs locally but accesses the remote data set RESERVC. A subset of RESERVC is written to the local data set TMP. The WHERE= and KEEP= options are passed to the server session and evaluated there to minimize the amount of data that must move across the network.

❸ Summarize the local data set.

❹ Create a report using this local, summary data set.

❺ Display the report.

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