

Web Based Report Ordering combined with Base/SAS® Mainframe Batch Processing

Andre Brainard, Systems Engineering Services Corp., Reston, VA

ABSTRACT

This paper presents a web-based on-demand report ordering system that uses UNIX System Services (USS) and WebSphere to update and report data stored on an IBM OS/390 mainframe. The business user issues a request from a desktop PC and gets the results delivered via e-mail and imbedded URL hyperlinks.

INTRODUCTION

The Management Analytics Reporting System (MARS) was developed in response to a need for a centralized reporting system that would allow business users to request ad-hoc reports in an automated fashion.

Prior to the implementation of this system, ad-hoc reporting relied upon informal request procedures between business users and an often limited number of technical resource personnel. This approach made the process more error prone and time consuming. In addition, users from different business areas often duplicated requests, which resulted in higher costs and at times, inconsistent results.

NEW SYSTEM REQUIREMENTS

A new system would require the following capabilities:

- The ability for users corporate wide to access the report ordering facility
- The ability for the user to specify varied and multiple selection criteria such as date ranges, customer identification, and product identification
- The ability to query large volume mainframe data stored on multiple mediums including DASD and magnetic tape
- The ability to query data stored in various forms including relational database, VSAM files, GDG flat files, and IMS database
- The ability to query data residing on multiple platforms including mainframe and UNIX.
- The ability to query data using different access protocol such as Sybase, UDB/DB2, IMS, VSAM, SAS, flat files, GDG datasets, and PDS datasets
- The ability to leverage reuse of legacy mainframe ad-hoc reporting code including COBOL, SAS, and SAS format libraries
- The ability to provide automated delivery of the ordered report results to the user
- The ability to provide automated notification of job failures

THE FRONT-END – A WEB BASED USER INTERFACE

The Web browser, e-mail and spreadsheet were the three tools chosen to meet the front-end user interface requirements for several reasons:

- On an enterprise level these familiar and friendly tools were already in place and being used on a daily basis by the business areas.
- Little or no training would be necessary for the users and no installation or maintenance would be required for the users' PCs.
- The use of HTML and JavaScript would allow the technician to easily develop input forms for the query selection criteria.
- E-mail would provide both the automated notification and delivery of the query results via imbedded URL hyperlinks.
- Business users agreed that it would be desirable to receive the query results in spreadsheet form, which would facilitate viewing, printing, and further analysis.

THE BACK-END – BASE/SAS AND OS/390

The back-end tools and environment chosen were Base/SAS running on an OS/390 Enterprise Server (formerly known as the mainframe) with MVS, OpenMVS, UNIX System Services (USS), and WebSphere.

SAS batch processing running in the OS/390 MVS environment would provide the flexibility and power to process large volume data on multiple platforms residing in diverse storage form using a variety of access protocols. SAS also would provide the ability to easily sort, merge, and summarize large volume complex data.

The robust OS/390 MVS environment would provide the necessary system memory, disk workspace and processing power to accomplish complex queries in a reasonable time frame. OS/390 USS and WebSphere would provide the critically important bridge between the mainframe environment and the corporate wide intranet.

SAMPLES OF CRITICAL CODE

The body of this paper provides samples of the front-end JavaScript code, the back-end MVS JCL with associated PROCS, and the back-end Base/SAS code necessary for the development of a similar web-based on-demand report ordering system.

The complete REXX CGI script will be provided upon request.

JAVASCRIPT CALL OF WEBSERVER CGI SCRIPT

INTERNET, INTRANET AND THE WEB

The front-end JavaScript does the following:

- Establishes input variables from Web page forms
- Performs minimal validations
- Assembles a delimited parameter list to be passed to the mainframe CGI script
- Substitutes imbedded space characters with a "+" in the parameter list
- If all data is valid, calls mainframe CGI script residing on the OS/390 OpenMVS webserver

```
function submitJob()
{ var validData = true;

  var e = document.forms[0].seller;
  var seller = e.value;
  if (seller == "") {var seller = "****";}

  if (validData && seller != "" &&
      (isNaN(seller) || seller.length != 6 || seller < "000001"))
    { alert("Enter a valid Seller Number"); var validData = false;}

  var parmList = "?" + "MARS.CNTL(MARS001)" +
    "?" + "MARS0001" + "?" + startdate +
    "?" + enddate + "?" + seller + "?" + reportby;

  out = ""; // replace this
  add = "+"; // with this
  temp = "" + parmList; // temporary holder
  while (temp.indexOf(out)>-1)
    { pos= temp.indexOf(out);
      temp = "" + (temp.substring(0, pos) + add +
        temp.substring((pos + out.length), temp.length)); }

  var parmString = temp;
  var webServer = "http://omvs2.company.com/cgi-bin/";
  var cgiScript = "webparms.cgi";

  if (validData) { alert("Your query results will be sent via e-
    mail.");
    window.open(webServer + cgiScript + parmString) }
}
```

THE CGI SCRIPT EDIT AND JOB SUBMISSION

The called CGI script written in the REstructured eXtended eXecutor (REXX) language performs the following:

- Retrieves the targeted mainframe job from the specified MVS library
- Edits the JCL scanning for the anchor point statement "/*JS00 EXEC WEBPARMS" which is replaced with the following JCL and the parameters passed by the JavaScript
- Submits the edited Jobstream to the OS/390 MVS Job Entry System (JES) internal reader
- Returns notification of either success or failure to requesting user

```
/**** THE FOLLOWING INSERTED BY CGI SCRIPT ***
//WEBPARMS EXEC PGM=IEBGENER
//SYSUT1 DD *
LIBRARY(JOBNAME)
USERID
CCYYMMDD
HH:MM:SS
Parm(s) each on separate line, start cc 2, max 70 char
/*
//SYSUT2 DD DSN=&&WEBPARMS
```

```
// DISP=(NEW,PASS,DELETE),
// SPACE=(TRK,1),UNIT=SYSDA,
// DCB=(LRECL=80,BLKSIZE=0,RECFM=FB)
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
```

THE TARGET JCL STRUCTURE

The following mainframe batch Job Control Language (JCL) example would be stored as a member of an OS/390 MVS mainframe Partitioned Data Set (PDS) library. The JES bind statement is essential to insure that the JES binds these jobs to the appropriate MVS environment, in this case production.

```
/*MARS0001 JOB (000,RK),'MARS',CLASS=M,MSGCLASS=P
/*+JBS BIND SERVER.USSM2
/*
/**** ESTABLISH INPUT WEB PARMS ***
//JS00 EXEC WEBPARMS <== anchor for cgi script
/*
/**** SAS QUERY PROCESSING ***
//JS10 EXEC SAS
//SYSIN DD DSN=MARS.SOURCE(MARS001S), DISP=SHR
//JOBINFO DD *
//WEBPARMS DD DSN=&&WEBPARMS,DISP=SHR
//OMVSXFER DD DSN=&&OMVSXFER, DISP=(NEW,PASS),
// UNIT=...,SPACE=...,DCB=(.....)
//EMAILTO DD DSN=&&EMAILTO, DISP=(NEW,PASS),
// UNIT=...,SPACE=...,DCB=(.....)
/**** INSERT DD STATEMENT(S) FOR ANY INPUT FILES
/*
/**** XFER RESULTS TO WEBSERVER USING OPUT ***
//JS20 EXEC WEBDATA,COND=(4,LT)
//SYSTSIN DD DSN=&&OMVSXFER,DISP=SHR
/*
/**** SEND EMAIL QUERY RESULTS TO USER ***
//JS30 EXEC SENDMAIL,COND=(4,LT)
//SYSTSIN DD DSN=&&EMAILTO,DISP=SHR
/*
/**** Optional Failure Notification goes here (refer later)
```

THE STORED JCL PROCEDURES

The following three mainframe JCL procedures will need to reside as members of the production OS/390 MVS PROC library.

```
/**** PROC: WEBPARMS ***
//WEBPARMS PROC
//WEBPARMS EXEC PGM=IEBGENER
//SYSUT1 DD DUMMY
//SYSUT2 DD DSN=&&WEBPARMS, DISP=(NEW,PASS),
// UNIT=SYSDA, SPACE=(80,(1,1)),
DCB=(BLKSIZE=0,DSORG=PS)
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
```

```
/**** PROC: WEBDATA ***
//WEBDATA PROC
//WEBDATA EXEC PGM=IKJEFT01
//SYSPROC DD DISP=SHR,DSN=SYS2.CLIST
//SYSTSPT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSIN DD DUMMY <== USER OVERRIDES
```

```

**** PROC: SENDMAIL ****
//SENDMAIL PROC
//SENDMAIL EXEC PGM=IKJEFT01
//SYSPROC DD DISP=SHR,DSN=SYS2.CLIST
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSIN DD DUMMY      <== USER OVERRIDES
//*****
/* PURPOSE: TO SEND E-MAIL FROM THE MAINFRAME
/*
/*-----
/* SYSTSIN DD MUST BE OVERRIDDEN BY USER SYSTSIN
/* WITH EITHER A REAL OR TEMPORARY DSN.
/* TEMPLATE OF SYSTSIN INPUT AS FOLLOWS:
/*-----
/* SENDMAIL TO(SOME_ONE@COMPANY.COM) +
/* CC(SOMEBOY_ELSE@COMPANY.COM) +
/* SUBJECT(SUBJECT MATTER) +
/* DATASET('DSN FOR E-MAIL BODY') +
/* BATCH
/*-----

```

THE SAS QUERY

PART 1 - PICKUP SYSTEM ASSIGNED JOB-ID

The first section of code performs the initialization necessary to support the remaining processing. In order to provide the pertinent job identification information in the e-mail notification, the MVS JOB-ID is picked up from the Job File Control Block (JFCB) by using a dummy DD JCL statement.

```

/* INITIALIZE: PICKUP JOB-ID EXECUTION INFO */
DATA _NULL_;
INFILE JOBINFO JFCB=JFCB; JOB_ID= ' ';
IF INDEX(JFCB, 'JOB')
THEN
JOB_ID=J'||SUBSTR(JFCB, (INDEX(JFCB, 'JOB')+4,5);
CALL SYMPUT('JOB_ID', JOB_ID);
RUN;

```

PART 2 - WEB PARMS INTO GLOBALS

As part of initialization, this code inputs each Web parameter and converts it into a global variable to enable usage by the processing code sections that follows. Adding PUT PARMVAR= statements for each input parm would provide a nice audit trail in the SASLOG file.

```

/* INITIALIZE: INPUT WEB PARMS AS GLOBAL VARIABLES
*/
DATA _NULL_;
INFILE WEBPARMS MISSEVER END=ENDPARMS;
IF _N_ = 1 THEN DO;      /* LIBRARY(MEMBER) */
INPUT @2 JOB_RUN $50.;
L_PAREN=INDEX(JOB_RUN, "(");
R_PAREN=INDEX(JOB_RUN, ")");
JOB_LNTH=(R_PAREN - L_PAREN) - 1;

JOB_NAME=SUBSTR(JOB_RUN, L_PAREN+1, JOB_LNTH);
CALL SYMPUT('JOB_RUN', JOB_RUN);
CALL SYMPUT('JOB_NAME', JOB_NAME);
END;
IF _N_ = 2 THEN DO;      /* MAINFRAME USER-ID */
INPUT @2 USER_ID $7.;
CALL SYMPUT('USER_ID', USER_ID);
END;

```

```

IF _N_ = 3 THEN DO;      /* DATE STAMP CCYYMMDD */
INPUT @2 IN_DATE $8.;
CALL SYMPUT('IN_DATE', IN_DATE);
END;
IF _N_ = 4 THEN DO;      /* TIME HH:MM:SS */
INPUT @2 IN_TIME $8.;
CALL SYMPUT('IN_TIME', IN_TIME);
END;
IF _N_ = 5 THEN DO;      /* MARS REPORT-ID */
INPUT @2 MARS_ID $8.;
CALL SYMPUT('MARS_ID', MARS_ID);
END;

/* - USER INPUT PARAMETERS FOLLOW - */
IF _N_ = 6 THEN DO;      /* BEGIN DATE */
INPUT @2 IN_BEGDT $8.;
CALL SYMPUT('IN_BEGDT', IN_BEGDT);
END;
IF _N_ = 7 THEN DO;      /* END DATE */
INPUT @2 IN_ENDDT $8.;
CALL SYMPUT('IN_ENDDT', IN_ENDDT);
END;
IF _N_ = 8 THEN DO;      /* Seller Number */
INPUT
@002 FOR_SLR $6.;
CALL SYMPUT('FOR_SLR', FOR_SLR);
END;
IF _N_ = 9 THEN DO;      /* Report-By Offering */
INPUT
@002 RPT_BY $3.;
CALL SYMPUT('RPT_BY', RPT_BY);
END;
IF ENDPARMS THEN DO;      /* Verify Parm Count */
IF _N_ < 9 THEN ABORT ABEND 255;
STOP;
END;

RUN;

```

PART 3 - MAIN DATA QUERY PROCESSING

This section should include the queries, data extracts, sorting, merging and summarization to produce the specific request results. The power of SAS batch mainframe processing allows tremendous flexibility to access diverse and large sources of data residing on DASD, magnetic tape, and remotely via SAS/Connect.

```

/* PROCESS: QUERY and DATA EXTRACT */

```

```

/* PROCESS: SORT, MERGE, SUMMARIZE, ETC. */
DATA RESULTS;

```

PART 4 - ALLOCATE OUTPUT RESULTS FILE

This code allocates the output file dynamically in the SAS code in order to generate a unique "DELETE.USERID." DSN thus avoiding duplicate DSN conflicts.

Important reminders:

- The OPUT transfer command must reference a system catalogued dataset.
- Use of delete.userid. catalogued datasets are permitted for any user-id with only minimal security access.
- Use of delete.userid. datasets facilitates automatic file cleanup by MVS.

INTERNET, INTRANET AND THE WEB

- Be sure to allow both an adequate logical record length (LRECL=) and an adequate DASD file space (SPACE=) as output needs dictate.

```
/* INITIALIZE: ALLOCATE UNIQUE DSN RESULTS FILE */
DATA _NULL_;
MARS_DAT = "DELETE.&SYSUID..D&RUN_JDAY..T" ||
  COMPRESS("&RUN_TIME",":") ||
  "&MARS_ID..DAT";
MARS_DAT = UPCASE(COMPRESS(MARS_DAT));
CALL SYMPUT(MARS_DAT,MARS_DAT);
RUN;

DATA _NULL_;
FILENAME &MARS_ID &MARS_DAT NEW
  DISP=(NEW,CATLG,DELETE) SPACE=(TRK,(100,10),RLSE)
  UNIT=SYSDA LRECL=800 BLKSIZE=0 RECFM=FB
  DSORG=PS;
RUN;
```

PART 5 – GENERATE OUTPUT .CSV RESULTS

This code generates the output results data as a Comma Separated Values (CSV) file. The .CSV file type extension is critical for enabling the Web browser to recognize this file type belongs to the spreadsheet application. When the user clicks on the results_file.csv hyperlink, inserted into the body of the e-mail, the browser will pass the .CSV file from the Webserver directly into the spreadsheet application.

```
/* OUTPUT RESULTS AS COMMA DELIMITED FILE */
DATA _NULL_;
SET RESULTS;
FILE &MARS_ID NOPRINT;
PUT ,,,,, comma delimited output ,,,,
RUN;
```

PART 6 - GENERATE FILE TRANSFER COMMANDS

The following SAS code will generate the OPUT command necessary for transferring the MVS output results over to the OpenMVS USS WebSphere area. The new file resides as an HFS file under WebSphere available for user access via the imbedded URL in the e-mail body.

```
/* OUTPUT COMMAND FILE FOR XFER TO WEBSHERE */
DATA _NULL_;
MARS_CSV = "&SYSUID..d&RUN_JDAY..t" ||
  COMPRESS("&RUN_TIME",":") || "&MARS_ID..csv";
MARS_CSV = LOWCASE(COMPRESS(MARS_CSV));
CALL SYMPUT(MARS_CSV,MARS_CSV);
PUT MARS_CSV;
RUN;

DATA _NULL_;
IF &TRANOUT = 0 THEN STOP;
OUT_UNIX = TRIM("&MARS_CSV");
FILE OMVSXFER NOPRINT;
PUT " oput &MARS_DAT +";
PUT " 'u/mars/prod/reports/ondemand/' OUT_UNIX +(-1) """;
```

PART 7 - GENERATE E-MAIL NOTIFICATION

This final section of the SAS Query program code generates the e-mail that serves as the notification and delivery vehicle of the query results. Assuming a successful query, the body of the e-mail will contain an imbedded hyperlink which the business user will click-on to conveniently have the results delivered directly in spreadsheet form.

Important points:

- The SENDMAIL command must reference a system catalogued dataset for the e-mail body
- The e-mail body DSN must exist after the job completes and be available to the queued SENDMAIL (up to 2 minutes)

```
/* GENERATE ROUTING AND BODY OF E-MAIL MESSAGE */
DATA _NULL_;
MARS_MSG = "DELETE.&SYSUID..D&RUN_JDAY..T" ||
  COMPRESS("&RUN_TIME",":") ||
  "&MARS_ID..MSG";
MARS_MSG = UPCASE(COMPRESS(MARS_MSG));
CALL SYMPUT(MARS_MSG,MARS_MSG);
RUN;

DATA _NULL_;
FILENAME EBODY &MARS_MSG NEW
  DISP=(NEW,CATLG,DELETE) SPACE=(TRK,(1,1),RLSE)
  UNIT=SYSDA LRECL=100 BLKSIZE=0 RECFM=FB
  DSORG=PS;

DATA _NULL_;
FILE EMAILTO NOPRINT;
USER_ID = PUT("&USER_ID", $MARSUSR.);
IF USER_ID = 'MISSING'
  THEN EMAIL_TO = "mars_project@company.com";
ELSE EMAIL_TO = USER_ID||"@company.com";
MARS_MSG = TRIM("&MARS_MSG");
PUT " SENDMAIL TO(" EMAIL_TO +(-1) ")" +";
PUT " CC(MARS_PROJECT@company.COM) +";
IF USER_ID = 'MISSING'
  THEN PUT " SUBJECT("ERROR" id: &USER_ID "
    "missing from MARS e-mail user table) +";
ELSE IF &TRANOUT > 0
  THEN PUT " SUBJECT("SUCCESS" YOUR MARS0001
    RESULTS ARE READY) +";
ELSE PUT " SUBJECT("NOTICE" NO RESULTS RETURNED
    FOR MARS0001) +";
PUT " DATASET(" MARS_MSG +(-1) ")" +";
PUT " BATCH";

DATA _NULL_;
FILE EBODY NOPRINT;
SET CDW_FHA;
IF _N_ = 1 THEN DO;
  PUT "Management * Analytics * Reporting * System" /;
  PUT "OS/390 ENTERPRISE SERVER: "
    "MVS JOBNAME=&SYSJOBID JOBID=&JOB_ID
    USERID=&SYSUID";
  REQ_DATE = INPUT("&IN_DATE", YMMDD8.);
  PUT "SUBMITTED ON: " REQ_DATE WEEKDATE29. " at:
    &IN_TIME"
    " by USER-ID: &USER_ID " /;
  PUT "REPORT: "MARS0001" REPORT TITLE" /;
  IF INPUT("&FOR_SLR", $1.) ne ""
    then do;
    SELLER = TRIM(PUT("&FOR_SLR", $SLRNAM.));
    PUT "FOR SELLER: " SELLER;
  end;

  IF INPUT("&RPT_BY", $1.) NE ""
```

```

THEN BYDESC =
TRIM(PUT(INPUT("&RPT_BY", $3.), $RPTBY.));
ELSE BYDESC = 'All Flow Offerings';
PUT 'REPORT-BY: ' BYDESC;
PUT "PERIOD: &BEG_DATE TO &END_DATE ";
CUR_DATE = TODAY();
PUT "STATUS AS OF: " CUR_DATE MMDDYY10. /;

IF &TRANOUT > 0 THEN DO;
  PUT "CLICK ON LINK TO VIEW RESULTS:";
  PUT "http://omvs2.company.com/marsreportsp/&MARS_CSV"
  /;
  END;
ELSE DO;
  PUT "'NOTICE' NO RESULTS RETURNED FROM THIS
  QUERY!" /;
  END;
RUN;

```

JOB FAILURE NOTIFICATION – JCL CODE

Add the following jobs steps to the end of the targeted JCL for the purpose of generating and sending e-mail notification of a job failure. Inclusion of job specific information, i.e. Job name and system assigned Job-ID, will help with providing efficient production support.

```

/*
**** HANDLE JOB FAILURE NOTIFICATION ****
//JC01 IF ABEND=TRUE OR RC > 4 THEN
/*
**** CREATE E-MAIL NOTIFICATION OF JOB FAILURE ****
//JS40 EXEC SAS
//SYSIN DD DSN=MARS.SOURCE(MARS911S),DISP=SHR
//JOBINFO DD *
//WEBPARMS DD DSN=&&WEBPARMS,DISP= DISP=SHR
//NOTIFY DD DSN=&&NOTIFY,DISP=(NEW,PASS),
// UNIT=...,SPACE=...,DCB=(.....)
/*
**** SEND E-MAIL NOTIFICATION OF JOB FAILURE ****
//JS50 EXEC SENDMAIL
//SYSTSIN DD DSN=&&NOTIFY,DISP=SHR
/*
//JC01 ENDIF

```

JOB FAILURE NOTIFICATION – SAS CODE

The SAS program MARS911S code necessary for generating the e-mail notification of job failure is easily replicated from the first and last sections of the MARS001S SAS Query program code.

SAS NOTIFY

Part 1 – Pickup System Assigned Job-ID

Copy the "Part 1- Pickup Job-ID" SAS code from the MARS001S SAS Query program above.

Part 2 – Web Parm into Globals

Copy the "Part 2 - Web Parm into Globals" SAS code from the MARS001S SAS Query program above.

Part 3 – Generate E-mail Notification

Copy the "Part 7 - Generate Delivery E-mail" SAS code from the MARS001S SAS Query program above and then modify the subject and body to announce job failure along with action to be taken and relevant contact information.

CONCLUSION

The MARS project provides an enterprise level solution that has not only brought ad-hoc reporting under control, but also has established a new technical infrastructure paradigm. Centralizing ad-hoc reporting on the corporate intranet opens it to the widest possible audience, while at the same time dramatically reducing future development costs. Accessibility of existing ad-hoc reports encourages their reuse and helps to avoid duplication. With the business user now able to independently order ad-hoc reports using flexible selection criteria, the need for doing "what if" scenarios has been made easier and faster. And finally, the new technical architecture and toolset improves the collaborative JAD/RAD approach to meet future ad-hoc reporting needs.

ACKNOWLEDGMENTS

Thanks to William A. Mitchell, Lead Tech Analyst, Freddie Mac Corporation, MacLean, VA for developing the REXX CGI script and for providing invaluable technical support.

Thanks to Michael G. Sadof, a SAS Quality Partner, MGS Associates, Bethesda, MD for encouraging me to present this paper.

SAS, BASE/SAS are registered trademarks or trademarks of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.

IBM, OS/390, REXX, and WebSphere are registered trademarks or trademarks of International Business Machines Corporation in the USA and other countries.

CONTACT INFORMATION

For those interested in attaining full version copies of the code, including the REXX CGI script, please make request to the e-mail address that follows.



Andre W. Brainard,
e-mail: andretech@onebox.com
Systems Engineering Services Corp.
On the Web at: <http://www.sesc.com>