#SHAREorg

SHARE
Technology · Connections · Results

Accessing SDSF data using Rexx and Java

Chip Wood
SDSF Design/Development
IBM Poughkeepsie
chipwood@us.ibm.com

Thursday, August 9, 2012
Session 11701

IBM

SHARE in Anaheim 2012

Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

IBM®
MVS
JES2
JES3
RACF®
REXX
z/OS®
zSeries®

* Registered trademarks of IBM Corporation

The following are trademarks or registered trademarks of other companies.

Java and all Java-related trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.
UNIX is a registered trademark of The Open Group in the United States and other countries.
SET and Secure Electronic Transaction are trademarks owned by SET Secure Electronic Transaction LLC.

* All other products may be trademarks or registered trademarks of their respective companies.

Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

Overview

- With SDSF's REXX and Java support, you can perform most of the tasks that you can perform interactively, such as:
  - Display and modify jobs
  - Display and modify resources and devices
  - Browse SYSOUT data sets
  - Print SYSOUT data sets
- REXX (added in z/OS 1.9) uses the same panel commands, action characters and column overtypes as with interactive SDSF
- Java (added in z/OS 1.12) ultimately uses a similar interface into SDSF but the programming interface is a collection of objects and methods which are more Java-friendly.
- This presentation will discuss the REXX techniques first, since they more closely resemble the interactive commands, then discuss the equivalent function in Java

Getting Started with REXX

In a basic SDSF REXX exec, you:

1. Add the REXX host command environment; before issuing any SDSF commands, using **ISFCALLS**
   - Allows use of **"Address SDSF"** for commands
2. Issue an SDSF command to access a panel, using **ISFEXEC**
3. Issue an action character or "overtype" a column using **ISFACT**

- Data is returned in stem variables
- Use special variables to control results
  - These correspond to SDSF commands such as PREFIX and OWNER

Rexx Example – Cancel a Job

```
rc=isfcalls(”ON”)

isfowner = “D96CLW1”
Address SDSF “ISFEXEC ST”
do ix=1 to JNAME.0   /* variable names same as FLD names */
    if pos(“CHIP”,JNAME.ix) = 1 then
      Address SDSF “ISFACT ST TOKEN(‘”TOKEN.ix”’) PARM(NP P)”
    […lines omitted…]
end

rc=isfcalls(“OFF”)
```

Add host command environment

Access the ST panel

Find the job

Take an action on the job

Remove the host command environment (after closing the loop)

Getting Started with Java

- Update CLASSPATH environment variable to reference SDSF jar file:
  - export CLASSPATH=/usr/include/java_classes/isfjcall.jar:$CLASSPATH
- Update LIBPATH to reference SDSF DLL:
  - export LIBPATH=/usr/lib/java_runtime:$LIBPATH (31-bit)
  - export LIBPATH=/usr/lib/java_runtime64:$LIBPATH (64-bit)
- SDSF requires Java SDK V6
  - Either 31-bit or 64-bit mode

Getting Started with Java …

- Create a runner that corresponds to the panel you want to work with
  - A runner is a Java class that provides access to SDSF
  - Contains a results object describing completion of request
- Create request settings and associate it with runner
- Invoke SDSF to create a list of objects
- Process the returned objects and obtain column values for each row
- Invoke methods on a row object to retrieve information or modify the object

Example Java Application

```
// Create optional settings object
ISFRequestSettings settings = new ISFRequestSettings();
settings.addISFOwner("D96CLW1");  // Set owner

// Get a runner used to access SDSF ST panel
ISFStatusRunner runner = new ISFStatusRunner(settings);

List<ISFStatus> statObjList = null;

statObjList = runner.exec();
// Missing exception handling – more on that later

// Cancel job
if (statObjList != null) {
  for (ISFStatus statObj : statObjList) {
      String jobname=statObj.getValue("jname")
       if (jobname.startsWith("CHIP"))
          statObj.cancel();
      }
```

Access the ST panel

Find the job

Take an action on the job

Accessing an SDSF Panel with REXX

- Use ISFEXEC to access a panel

- Syntax:
  Address SDSF "ISFEXEC *sdsf-command* (*options*)"
  - *sdsf-command* is the same SDSF command as you use interactively, including parameters, for example:
    - Address SDSF “ISFEXEC DA”
    - Address SDSF “ISFEXEC CK ALL”

DA command

CK command with the ALL parameter

Java Runners and Settings

- A runner provides access to SDSF similar to SDSF commands
  - Choose the runner corresponding to the panel you want to access
    - ISFStatusRunner – ST (status panel)
    - ISFOutputRunner – O (output panel)
    - ISFHealthCheckRunner – CK (health checks)
    - etc.
    - ISFRunner – slash command, WHO, QUERY
  - Complete cross reference of runners to panels contained in the Javadoc

Accessing an SDSF Panel with Java

- Create a runner for the panel
  - Each panel has a different one, for example:
    - ISFStatusRunner for ST
    - ISFHealthCheckRunner for CK
    - Etc.
- Execute the runner using exec() method
  - Output is a list of objects (Java.util.List)
    - ISFStatus for ST
    - ISFHealthCheck for CK

```
ISFStatusRunner runner = new ISFStatusRunner();
List<ISFStatus> statObjList = null;

statObjList = runner.exec();
```

ST command example

Accessing an SDSF Panel – Options (REXX)

Options you can use when accessing a panel with ISFEXEC or ISFACT:

- **PREFIX**: specify a prefix for column variables that are created
- **PRIMARY**: use the primary field list
- **ALTERNATE**: use the alternate field list
- **DELAYED**: include delayed-access columns
- **NOMODIFY**: don't return row tokens for use in modifying values
- **VERBOSE**: add diagnostic messages to the isfmsg2. stem variable (more on this later)

Accessing an SDSF Panel – Options (Java)

Options are specified within a ISFRequestSettings object, via specific methods for each

- settings.addPrimary(): use the primary field list
- settings.addAlternate() : use the alternate field list
- settings.addDelayed(): include delayed-access columns
- settings.addNoModify(): don’t return row tokens for use in modifying values
- settings.addVerbose(): add diagnostic messages to the ISFRequestResults object (more on this later)

Special Variables to Control SDSF

- Special variables for use with SDSF REXX
  - Defined by SDSF
  - Some correspond to SDSF commands
  - Others provide access to fields or data, such as the title line on an SDSF panel
    - Some input only, some output only, some both
  - Names start with “ISF”

Special Variables – Input

- Special variables with panel commands:
  - Limit the response when accessing a panel
  - Use before invoking ISFEXEC or ISFACT
- Examples

  isfprefix=*

  isfowner=ken

  isffilter="jprio gt 5"

  isfcols="JNAME JOBID OWNERID ACTSYS"

  isfsort = "TGNUM D"

Corresponds to the command PREFIX *

Corresponds to the command OWNER KEN

Corresponds to the command FILTER PRTY GT 5

Limits the column variables created

Corresponds to the command SORT TGNUM D

Java Runners and Settings …

- Settings are used to qualify the request
  - Job name prefix, owner, destination
  - Most settings correspond to SDSF commands
  - Limit the column values retrieved
- Represented by ISFRequestSettings class
  - Create an instance of settings and associate it with runner
  - Various addISFxxxx methods to add a setting to the object
- settings.addISFPrefix("**");
  settings.addISFOwner("ibmuser");
  settings.addISFCols(“jname jobid”);

Java Runners and Settings …

```
// Create optional settings object
ISFRequestSettings settings = new ISFRequestSettings();

settings.addISFPrefix("**");

settings.addISFOwner("ibmuser");

settings.addISFCols(“jname jobid”);

// Get a runner used to access SDSF ST panel using settings
ISFStatusRunner runner = new ISFStatusRunner(settings);
```

Corresponds to PREFIX ** command

Corresponds to OWNER IBMUSER command

Requests just the JOBNAME and JobID columns

Note that both Rexx and Java use column names rather than column titles for sorting and filtering. See COLSHELP to see the relationship between names and titles.

Special variables and settings (input)

Interactive	Rexx	Java
SET PREFIX *	isfprefix = '*'	settings.addISFPrefix("*") settings.removeISFPrefix()
SET OWNER D96CLW1	isfowner = 'D96CLW1'	settings.addISFOwner("D96CLW1") settings.removeISFOwner()
FILTER JPRIO GT 5	isffilter = 'jprio gt 5'	settings.addISFFilter("jprio gt 5") settings.removeISFFilter()
SORT TGNUM D	isfsort = 'tgnum d'	settings.addISFSort("tgnum d") settings.removeISFSort()
n/a (limit number of data rows returned)	isflinelim = 1000	settings.setResponseLimit(1000) settings.removeResponseLimit()
n/a (limit columns returned)	isfcols = 'jname jobid'	settings.addISFCols("jname jobid") settings.removeISFCols()
s.server(SDSF)	isfserver = 'SDSF'	settings.addISFServer("SDSF") settings.removeISFServer()
… and lots more		

Accessing an SDSF Panel – Data (Rexx)

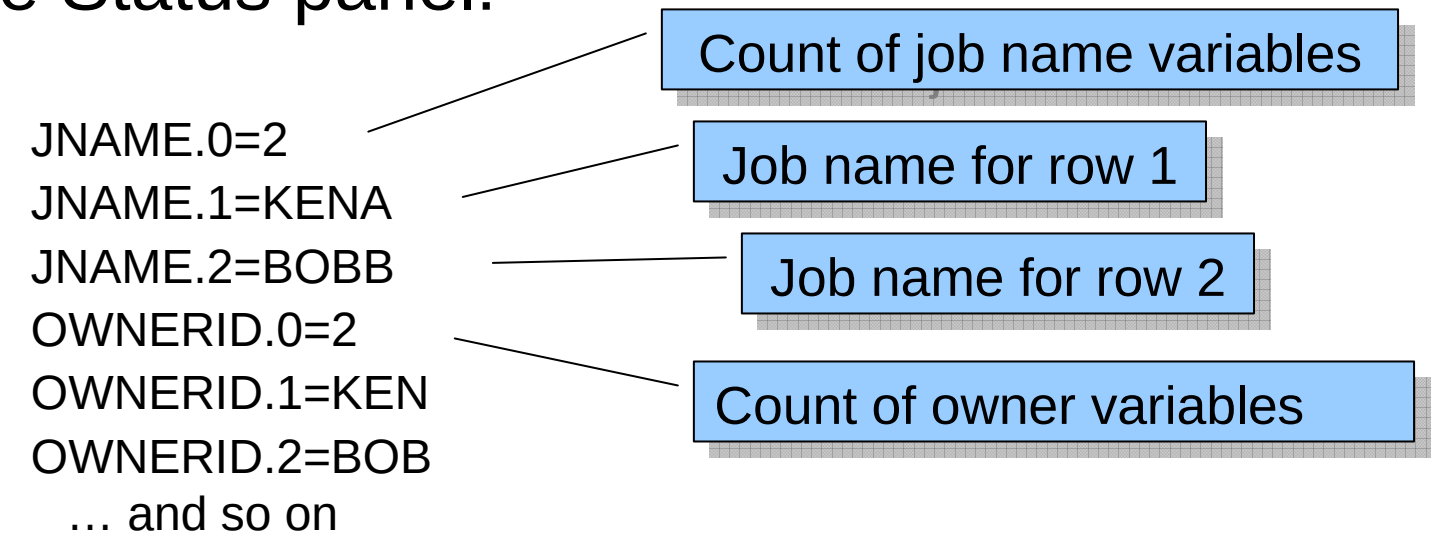
- SDSF builds stem variables/objects that correspond to the panel's rows and columns
  - *column-name*.*index* format
    - *column-name* is the name used on an FLDENT statement (not the column title), for example:

      FLDENT COLUMN(**OWNERID**),TITLE(OWNER),WIDTH(8)
    - *index* is the number of the row
      - *0 index is the number of variables in the stem*

➢ Display the column names with the COLSHELP command

Stem Variables for Panel Data - Example

REXX Stem variables and values for columns on the Status panel:



Working with Row Objects in Java

- SDSF creates one object per row
  - Column values are contained within the object
  - Use getValue() method to retrieve a column value
    - Use the SDSF column name (FLD name), not the column title
      - *String jobname=statObj.getValue("jname")*
      - *String owner=statObj.getValue("ownerid")*
  - Use getFixedField() method for fixed field
    - *String fixedField=statObj.getFixedField();*
  - Convenience methods exist for certain columns
    - *String jobname=statObj.getJName();*

Working with Objects …

```
…
statObjList = runner.exec();
…
for (ISFStatus statObj : statObjList) {

        String jobname = statObj.getValue("jname")
or      String jobname = statObj.getJName();

        System.out.println(statObj);

        System.out.println(statObj.toVerboseString());
}
```

Get job name

Print short form of row properties

Print all properties for row

Special Variables – Output

- Return data not associated with a particular row
- Examples

  isftline – title line

  isfrows – number of rows returned

  isfcols – list of columns returned

  isfmsg – short message

  isfmsg2. (stem variable) –detailed message information

  isfulog. (stem variable) – contents of user log (ULOG)

Request Results (Java)

- The runner references an ISFRequestResults object that is updated after each request
  - Contains messages describing completion of request
  - Return and reason codes
  - List of columns returned
  - Convenience methods to print messages
- Always check the results after each request
  - ISFRequestResults results = runner.getRequestResults();
  - string = results.getTitleLine()
  - string = results.getColumnNames()
  - results.printMessageList(*print stream*)

Rexx error handling

Should also check the return code from the SDSF command, for example: if rc<>0 then …

Return codes for ISFEXEC and ISFACT:

- **00** The request completed successfully.
- **08** An incorrect or invalid parameter was specified for an option or command.
- **12** A syntax error occurred parsing a host environment command.
- **16** The user is not authorized to invoke SDSF.
- **20** A request failed due to an environmental error.
- **24** A request failed due to an environmental error.

Rexx Message Variables

- Message variables contain SDSF messages
  - **isfmsg** contains the SDSF short message (displayed in the upper right corner on an SDSF panel)
  - **isfmsg2.** stem contains the SDSF numbered messages
  - **isfulog.** stem is for the user log (ULOG)

- Check after each SDSF request to ensure the request was successful

Java error handling

- Invocation of the exec() method on a runner can cause an exception, so those exceptions need to be handled
  - Exceptions generally represent a non-zero return code from SDSF

```
try {
   statObjList = runner.exec();
    } catch (ISFException ie) {
    ie.printStackTrace();
    System.out.println(ie.toVerboseString());
    System.out.println(ie.getMessage());
    System.out.println(ie.getRequestResults().getMessageList());
    }
```

Access the panel

Handle exception

Issue any messages that were generated

Message Variables Example with Slash

```
Address SDSF “ISFEXEC ‘/$da’ (WAIT”

if isfmsg<> “” then
  Say “isfmsg is:” isfmsg
do ix=1 to isfmsg2.0
  Say “isfmsg2.”ix “is:” isfmsg2.ix
end

do ix=1 to isfulog.0
   Say "isfulog."ix "is" isfulog.ix
end
```

Issue the w/$da command with WAIT option

Check for a short message

Check for a numbered message. The 0 stem contains a count of the numbered messages.

Check the ULOG

ISFSLASH Command

- Simplifies issuing system commands
- Similar to ISFEXEC, but:
  - Multiple commands can be entered on same invocation
  - Use either a stem variable or list of commands
  - All responses come back together in isfulog stem variables
- Syntax:
  - Address SDSF “ISFSLASH (*stemname*) | *command-list* (*options*”

ISFSLASH Command Syntax

- Address SDSF “ISFSLASH (*stemname*) | *command-list* (*options*”
  - *stemname* names a stem variable containing the commands to be issued
    - stemname.0 contains the count of variables that follow
  - *command-list* is a list of one or more commands to issue
- **isfcmdlim** special variable
  - Specifies a command limit to prevent excessive number of commands from being issued.
  - Default is no limit

Using ISFSLASH to Issue Multiple Commands

```
rc=isfcalls(”ON”)

cmd.0=2
cmd.1=“$da”
cmd.2=“$dq”

Address SDSF “ISFSLASH (cmd.) (WAIT)”

do ix=1 to isfulog.0
  say “isfulog.”ix “is:” isfulog.ix
end

rc=isfcalls(“OFF”)
```

Add the host command environment

Add commands to the stem variable

Issue the commands

Display messages from ULOG

Remove the host command environment

ULOG Variables Example - Results

```
isfulog.1 is: SY1      2009061  12:47:58.49          ISF031I CONSOLE KJONAS
isfulog.2 is: SY1      2009061  12:47:58.49         -$da
isfulog.3 is: SY1      2009061  12:47:58.49  J0000032   $HASP890 JOB(KJONASR)
isfulog.4 is:                                $HASP890 JOB(KJONASR)
isfulog.5 is:                                $HASP890
isfulog.6 is: SY1      2009061  12:47:58.50         -$dq
isfulog.7 is: SY1      2009061  12:47:58.50          $HASP643     10 PPU LO
isfulog.8 is: SY1      2009061  12:47:58.54          $HASP646 24.0000 PERCE
```

MVS Commands from Java

- Can issue one or more MVS commands
- Use ISFRunner with system method
  - Takes an array of string commands

```
String[] commands = new String[] {"$da","$dq"};
runner.system(commands)
```

- Get ISFRequestResults object using getRequestResults()
- Get command responses using
  - results.getResponseList() or
  - results.printResponseList(*print stream*)

Actions and Overtypes (Rexx)

- Use the ISFACT command to issue an action character or modify a value (overtype a column)

- Syntax:

  Address SDSF "ISFACT *SDSF-command* TOKEN((*stemname*) | *token.1, token.2, … , token.n*) PARM(*parms*) (*options*"

  - ***SDSF-command*** is the same SDSF command you used with ISFEXEC to access the panel

Actions and Overtypes - continued

TOKEN(*stemname*) is the name of stem variable containing row tokens

- Name is enclosed in parentheses
- *stemname.0* contains the count of variables that follow
- A stem variable can be null to skip a row

- **TOKEN(*token.1*, *token.2, … token.n*)** is a list of row tokens

PARM(*parms*)

- Describes the action or modification
  - PARM(OCLASS A FORMS 1234)
  - PARM(NP C)

Change both class & forms

Use NP for action characters

Example - Change Output Forms

```
isfprefix="**"
isfowner="RJONES"

Address SDSF "ISFEXEC O"
do ix=1 to JNAME.0
  if pos("BOB",JNAME.ix) = 1 then
    do
      Address SDSF "ISFACT O TOKEN(""TOKEN.ix"")
    PARM(FORMS 1234)"
    end
end
```

Set filters

Access O panel to set variables

Find a row with job name BOB

Change forms

Use the token for that row to identify it, enclosing it in single quotes

Actions (Java)

- You can modify an object similar to an action character
- Rows are represented by objects, lists of which are retrieved by executing runners
- Actions are represented by methods
  - Available actions defined in the interface for the object
    - See the Javadoc for com.ibm.zos.sdsf.core
  - For example:
    - ISFStatus.cancel()
    - ISFInitiator.start()
    - ISFHealthCheck.activate()
    - etc.

Overtypes (Java)

- You can modify an object similar to an overtype
  - Use the requestPropertyChange method
  - Method takes two input arrays:
    - Column name array
    - Column value array
  - Each column in the name array is changed to the corresponding value in the value array

Overtypes (Java) …

```
// Change job class to class A

// Build column name array
String[ ] propName = { "jclass" };

// Build column value array
String[ ] propValue = { "a" };

// Change the job class
statObj.requestPropertyChange(propName, propValue);
```

Browse Job Data Sets (Rexx)

- Use ISFACT to issue the SA action character against a job
  - Allocates the data set (free=close)
  - SA action is not allowed interactively
- Allocated ddname is returned in **isfddname.** stem variable
- Data set name is in **isfdsname.** stem variable
- Use EXECIO to read the data set

Example: Browse Job Data Sets

```
Address SDSF “ISFEXEC ST”
…
Address SDSF “ISFACT ST TOKEN(‘”TOKEN.ix”’) PARM(NP SA)”

do jx=1 to isfddname.0
   Say "Now reading" isfdsname.jx
   "EXECIO * DISKR" isfddname.jx "(STEM line. FINIS“
    Say “Lines read” line.0
    do kx=1 to line.0
       Say "  line."kx "is:" line.kx
    end
end
```

Access the ST panel, then use logic to find a job (not shown)

Issue SA action

Loop through ddnames

EXECIO reads the data set

Browse Job Data Sets (Java)

- Use results.getAllocationList() method to obtain an array of allocated DD names
  - Allocates the data sets (free=close)
- Use ZFile.read() method to read the data set
- See ISFBrowseSample.java for an example

SDSF/Rexx SYSLOG/OPERLOG

- Syntax of ISFLOG command:
  - **ISFLOG ALLOCATE**
    - Returns isfddname. stem variable, similar to data set browsing
    - Use EXECIO to read data
    - SYSLOG only (no OPERLOG)
  - **ISFLOG READ TYPE(SYSLOG | OPERLOG)**
    - Can read either SYSLOG or OPERLOG
    - Data returned in isfline. stem variable

Java SYSLOG/OPERLOG

- Create ISFLogRunner object
- Allocate using runner.browseAllocate()
  - Similar to browsing data sets

    OR

- Get lines using runner.readSyslog() or runner.readOperlog()
  - results.getResponseList() retrieves array of lines

ISFLOG Allocate Example

```
rc=isfcalls(“on”)

Address SDSF “ISFLOG ALLOCATE”
do ix=1 to isfddname.0

  "EXECIO 10 DISKR" isfddname.ix "(FINIS STEM log."
  do jx=1 to log.0
    Say mid "log."jx "is:" log.jx
  end
end

rc=isfcalls(“off”)
```

Allocate the logical SYSLOG

Loop through DD names

Read contents into log. stem variable

Report the log data

ISFLOG Read Example

```
rc=isfcalls(“on”)

Address SDSF “ISFLOG READ”

do ix=1 to isfline.0
  Say mid "isfline."left(ix,5)":" isfline.ix
end

rc=isfcalls(“off”)
```

Read the logical SYSLOG into the isfline. stem

Report the log data

ISFLOG Special Variables

- Used only by READ (not by ALLOCATE)
- Starting date and time
  - isflogstarttime (hh:mm:ss.th) / settings.addLogStartTime
    - Default is 00:00:00.00
  - isflogstartdate (mm/dd/yy) / settings.addLogStartDate
    - Default is current day
- Ending date and time
  - isflogstoptime (hh:mm:ss.th) / settings.addLogStopTime
    - Default is 23:59:59.59
  - isflogstopdate (mm/dd/yy) / settings.addLogStopDate
    - Default is current day
- isfdate (specify date format) / settings.addISFDate

ISFLOG Special Variables …

- isflinelim / settings.addISFLinelim
  - Specifies the maximum number of variables to be created
  - Default is no limit

- isflinelim=10000 / settings.addISFLineLim(10000)
  - Create a maximum of 10,000 variables

ISFLOG Read Example By Time/Date

```
rc=isfcalls(“on”)

isfdate="mmddyyyy /"
currday=date("C")
currday=currday-1   /* yesterday */
isflogstartdate=date("U",currday,"C")   /* yesterday in mm/dd/yy */
isflogstarttime=time("N")   /* current time */
isflogstopdate=date("U")   /* current date in mm/dd/yy */
isflogstoptime=time("N")   /* current time */

isflinelim=1000

Address SDSF “ISFLOG READ TYPE(OPERLOG)”
```

Set time and date parameters

Set maximum number of variables to create

Read the OPERLOG
This example also works if you specify TYPE(SYSLOG)

ISFLOG Read Example By Time /Date

```
do ix=1 to isfline.0
 Say mid "isfline."left(ix,5)":" isfline.ix
end

do ix=1 to isfmsg2.0
  Say isfmsg2.ix
end

rc=isfcalls(“off”)
```

Report the log data

Report any messages

Java LOG Read Example By Time/Date

```
// Get date formatters for the time and date
 final Calendar calendar = Calendar.getInstance();
 final DateFormat dateFormat = new SimpleDateFormat("MM/dd/yyyy");
 final DateFormat timeFormat = new SimpleDateFormat("hh:mm:ss");

 final Date today = calendar.getTime();
 calendar.add(Calendar.DATE, -1);
 final Date yesterday = calendar.getTime();

 // Set the start and stop times to limit records obtained
 ISFRequestSettings settings = new ISFRequestSettings();
 settings.addISFLogStartTime(timeFormat.format(today));
 settings.addISFLogStartDate(dateFormat.format(yesterday));
 settings.addISFLogStopTime(timeFormat.format(today));
 settings.addISFLogStopDate(dateFormat.format(today));
 settings.addISFDate(“mmddyyyy /”);

 settings.addISFLineLim(1000);
```

Set time and date parameters

Set maximum number of lines to create

Java LOG Read Example By Time/Date

```
ISFLogRunner runner = new ISFLogRunner(settings);

// Read the system log
runner.readSyslog();

ISFRequestResults results = runner.getRequestResults();

results.printMessageList(System.err);

results.printResponseList(System.out);
```

Read the SYSLOG
This example also works if you specify readOperlog()

Report any messages

Report the log data

Avoiding Duplicate Variable Names (Rexx)

- Use the **PREFIX** option on ISFEXEC and ISFACT to add a prefix to variable names created by SDSF
  - Prevents duplicate variable names in existing scripts
    - Needed when accessing the job data set panel, so that column variables don't conflict
  - Format: (PREFIX *prefix*)
- **PREFIX** only applies to column variables, not to special ISF variables.

Example: Using the PREFIX Option

```
Address SDSF “ISFACT ST TOKEN(‘”TOKEN.ix”’) PARM(NP ‘?’)
   (PREFIX jds_)”

do jx=1 to jds_DDNAME.0
say “DSName is” jds_DSNAME.jx
  Say “Stepname is” jds_STEPN.jx
  Say “Procstep is” jds_PROCS.jx
end
```

Access JDS using NP ? and define a prefix for all JDS variables.

References to variables all include the prefix

isfreset() Function

- REXX function to drop SDSF special variables
- Useful when multiple invocations of SDSF in same exec
- Syntax:
  - rc=isfreset(“ALL” | “INPUT” | “OUTPUT” | “INOUT”)
  - Drops all special variables of the type given
  - ALL (default)
    - rc=isfreset() will drop all SDSF special variables
- Not dependent on isfcalls(), can be placed anywhere in exec
- Not as interesting in Java as each runner can have its own unique ISFRequestSettings and ISFRequestResults objects
  - settings.reset() and results.reset() to clear them

Using SDSF with SYSREXX

- SDSF REXX Support works with System REXX
- Need proper security environment to access SDSF
  - Logon from console to get security environment
  - Need access to all commands used by EXEC
- Need to specify ISFJESNAME or ISFSERVER
  - ISFSERVER defaults to ‘SDSF’

Security

- SDSF security applies to REXX and Java usage
- No changes to ISFPARMS or SAF
- IBM recommends SAF for security instead of ISFPARMS for better control and auditing

Security – Assigning a User to a Group

- SDSF assigns users to a group in ISFPARMS with:
  - SAF: checks resource
    GROUP.*group-name*.*server-name* in the SDSF class
  - ISFPARMS: Uses user ID, logon proc, etc. to determine which group to use
    - With REXX, special values are assigned as follows:
      - Logon proc name: Set to **REXX**
      - TSO authority: Set to JCL authority
      - Terminal name: Derived from SAF or TSO based on the current environment

Diagnosing Problems

- Check ISFMSG variables and ISFMSG2. stem variable, or results.printMessageList()
- Use the VERBOSE option on ISFEXEC and ISFACT (settings.addVerbose())
  - Issues a message for each variable that is set
  - Useful in diagnosing problems such as ‘why doesn’t my job name comparison work?’
  - Example: Address SDSF “ISFEXEC DA (VERBOSE)”

    Results (in isfmsg2. stem variable):

ISF146I REXX variable JOBID.1 set, return code 00000001 value is ‘J0000040’.

ISF146I REXX variable OWNERID.1 set, return code 00000001 value is ‘RJONES’.

Diagnosing Problems (cont.)

- ISFDIAG variable/results.getDiag*xxx* methods
  - Intended for use by IBM Service
  - Contains internal reason codes for each request
  - You may be asked to employ it if you call IBM with a problem

COLSHELP

- Interactive command to relate column titles to column names
  - Column names (FLD name) are used anyplace in Rexx or Java a specific column is referenced, rather than column titles.
    - isffilter, isfsort, isfcols, ISFACT PARM(*column value*)
    - addISFFilter(), addISFSort(), getValue(), requestPropertyChange()
  - For example, JNAME for JobName column

- Context sensitive
  - Lists only columns for the panel
  - COLSH on DA lists only DA columns
    - Option to display all values

- Locate command to locate start of panel entries
- Filter command to filter by panel, name, or description

COLSHELP Example

```
                              Columns on SDSF Panels
Command ===>

Sort with F5 (panel), F6 (column), F10 (title). Use Filter to filter
   rows.

_ All panels          _ Descriptions

Panel     Column       Title                          Delayed?
DA        JNAME        JOBNAME
DA        STEPN        StepName
DA        PROCS        ProcStep
DA        JTYPE        Type
DA        JNUM         JNum
DA        JOBID        JobID
DA        OWNERID      Owner
```

Option to display columns from all panels

Sorting indicated by underscore

Columns for DA only

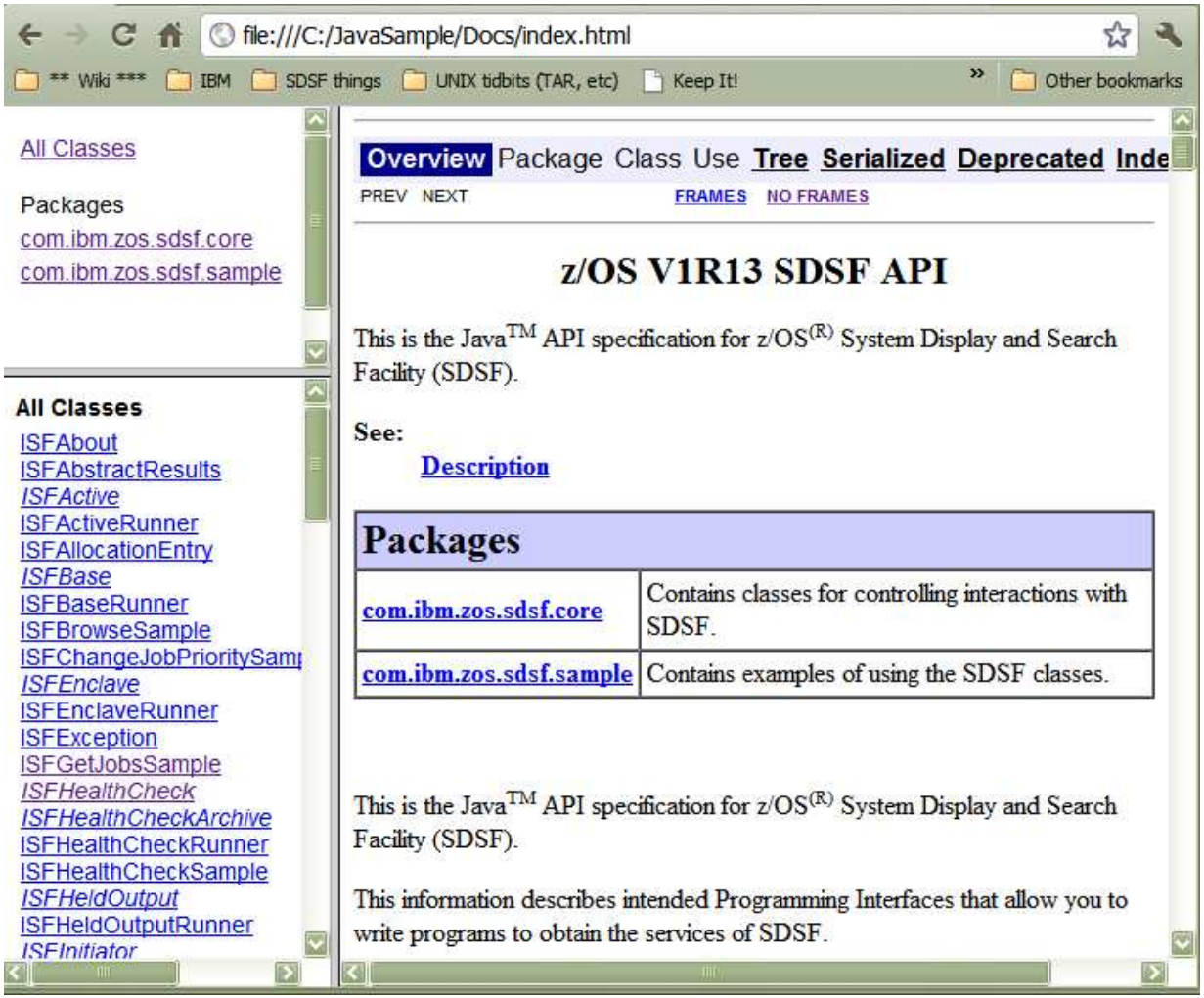
Java samples

- Sample Java scripts in **com.ibm.zos.sdsf.sample**
  - ISFBrowseSample
  - ISFChangeJobPrioritySample
  - ISFGetJobsSample
  - ISFHealthCheckSample
  - ISFSearchSyslogSample
  - ISFSlashCommandSample
  - ISFWhoCommandSample

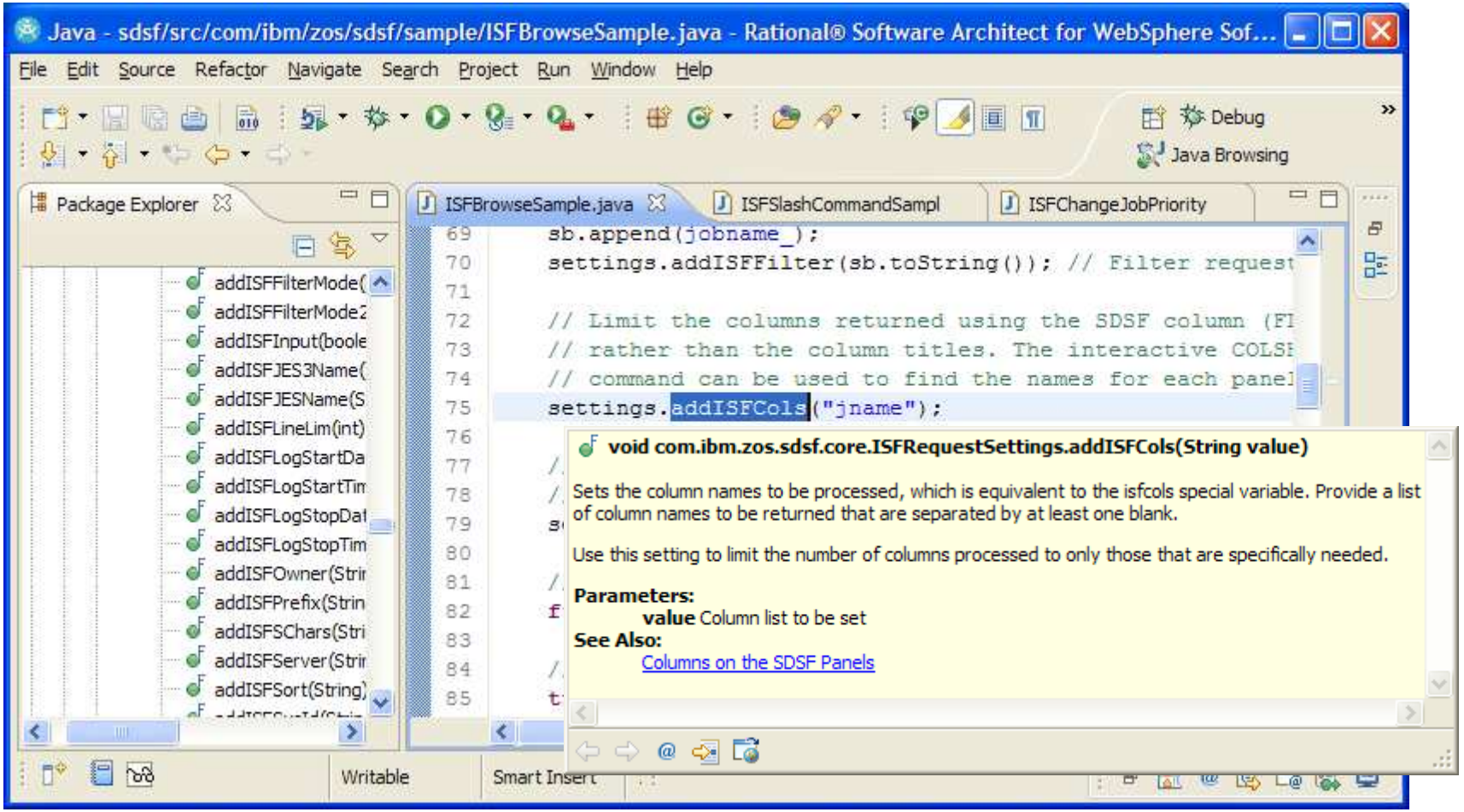
Installing Javadoc

- Download isfjcallDoc.jar to your workstation (in binary)
- Unzip the file
  - jar –xf isfjcallDoc.jar
- You can now access the index.html file in your web browser and navigate the Javadoc that way.
  - All the documentation for SDSF Java constructs reside here.
- You may also be able to access context-sensitive help, depending on tools you use to develop Java (e.g. RSA)

Javadoc example (web browser)



Javadoc example (RSA)



References

- Issue the **REXXHELP** command while using SDSF under ISPF
- Issue the **SEARCH** command while using SDSF under ISPF

- All Java documentation can be found in the Javadoc.

- See *SDSF Operation and Customization*: http://publibz.boulder.ibm.com/epubs/pdf/isf4cs70.pdf

- SDSF Web page, which will include examples for use with ISPF's MODEL command: http://www.ibm.com/servers/eserver/zseries/zos/sdsf/

- Redbook!
  - Loaded with interesting examples and experiences

SDSF REXX Redbook

Title: *Implementing REXX Support in SDSF,*
SG24-7419-00

http://www.redbooks.ibm.com/abstracts/sg247419.html

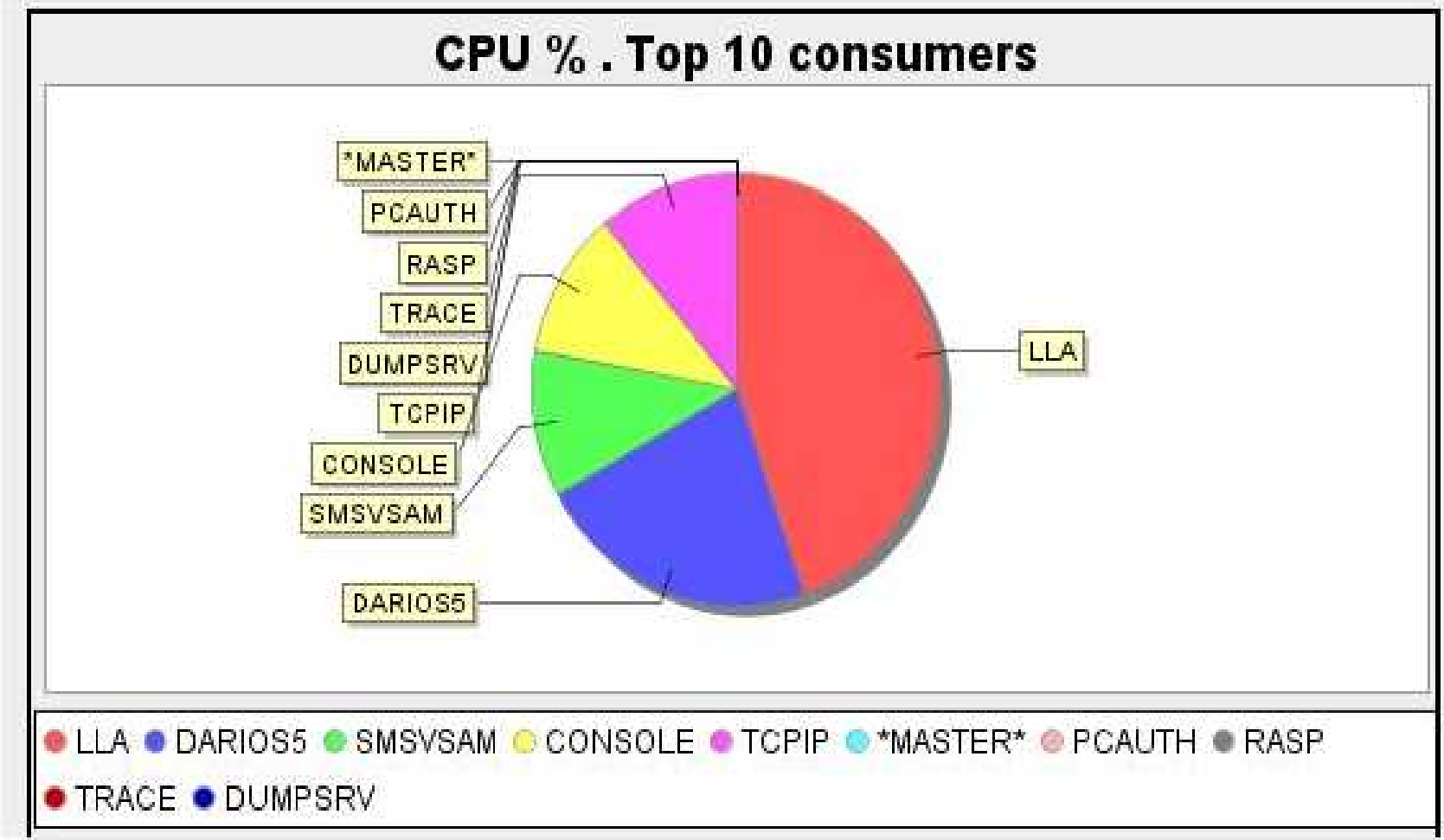
Abstract:

This IBM Redbooks publication describes the new support and provides sample REXX execs that exploit the new function and that perform real-world tasks related to operations, systems programming, system administration, and automation.

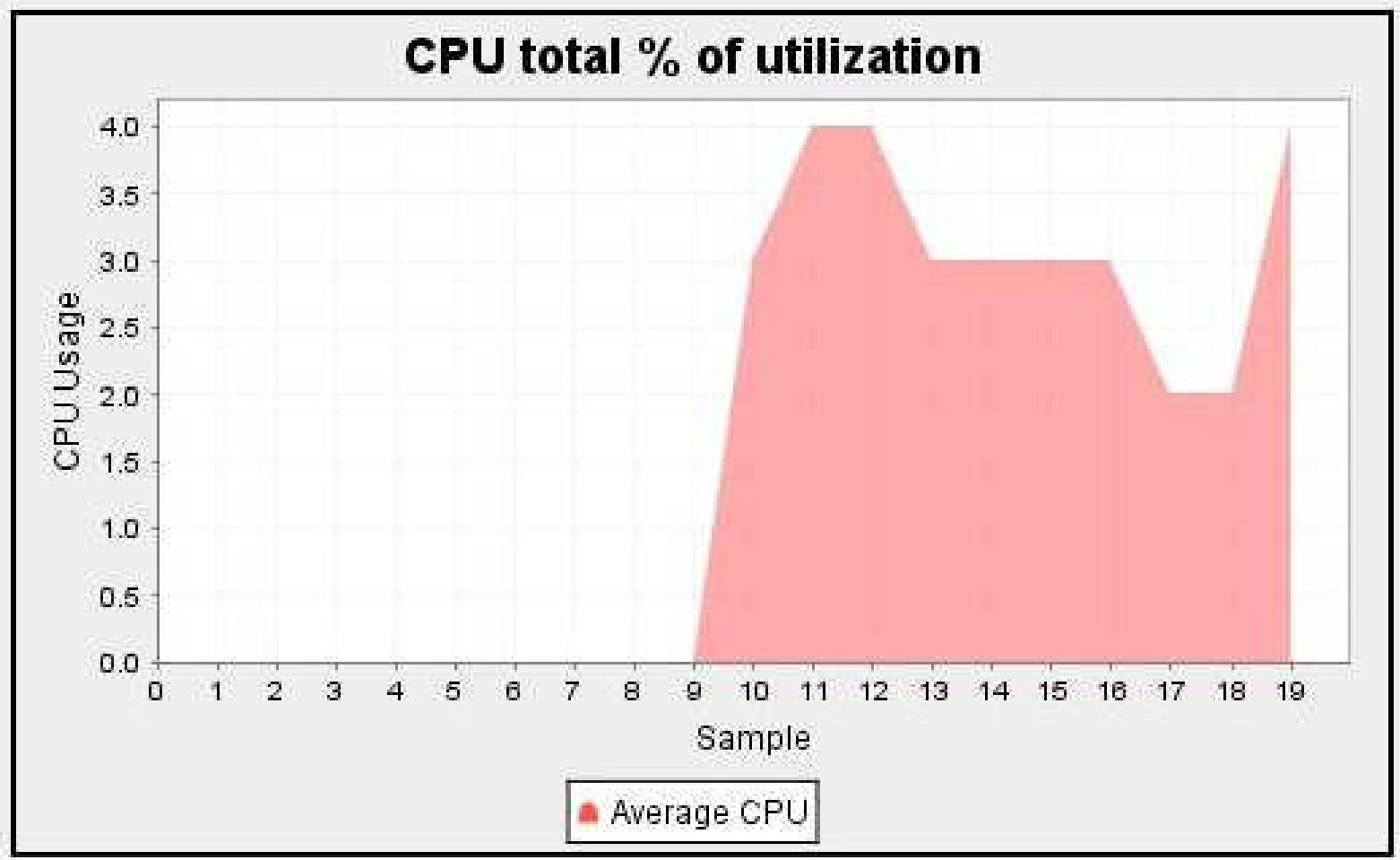
SDSF REXX Redbook - Topics

Chapter 1. Issuing a system command
Chapter 2. Copying SYSOUT to a PDS
Chapter 3. Bulk job update processor
Chapter 4. SDSF support for the COBOL language
Chapter 5. Searching for a message in SYSLOG
Chapter 6. Viewing SYSLOG
Chapter 7. Reviewing execution of a job
Chapter 8. Remote control from other systems
Chapter 9. JOB schedule and control
Chapter 10. SDSF data in graphics
Chapter 11. Extended uses

SDSF REXX Redbook - Examples



SDSF REXX Redbook - Examples



Summary

- Rexx
  - Use ISFCALLS to enable “Address SDSF”
  - Use ISFEXEC to access SDSF data
  - Use isf*xxxx* special variables to set up parameters
  - Use isf*xxxx* special variables to check results
  - Use stem variables to access row and column data
  - Use ISFACT TOKEN(token) PARM(xx) for actions and overtypes
- Java
  - Point CLASSPATH and LIBPATH to SDSF libraries
  - Use runners and exec() method to access SDSF data
  - Use ISFRequestSettings object to set up parameters
  - Use ISFRequestResults object to check results
  - Use list of row objects to access row and column data
  - Use methods on row objects for actions and overtypes