CA ESP Workload Automation:
Creating Installation Standards & Configuration

Steve DeWalt, Principal Consultant, CA Services
Terms of This Presentation

This presentation was based on current information and resource allocations as of October 2009 and is subject to change or withdrawal by CA at any time without notice. Notwithstanding anything in this presentation to the contrary, this presentation shall not serve to (i) affect the rights and/or obligations of CA or its licensees under any existing or future written license agreement or services agreement relating to any CA software product; or (ii) amend any product documentation or specifications for any CA software product. The development, release and timing of any features or functionality described in this presentation remain at CA’s sole discretion. Notwithstanding anything in this presentation to the contrary, upon the general availability of any future CA product release referenced in this presentation, CA will make such release available (i) for sale to new licensees of such product; and (ii) to existing licensees of such product on a when and if-available basis as part of CA maintenance and support, and in the form of a regularly scheduled major product release. Such releases may be made available to current licensees of such product who are current subscribers to CA maintenance and support on a when and if-available basis. In the event of a conflict between the terms of this paragraph and any other information contained in this presentation, the terms of this paragraph shall govern.
For Informational Purposes Only

Certain information in this presentation may outline CA’s general product direction. All information in this presentation is for your informational purposes only and may not be incorporated into any contract. CA assumes no responsibility for the accuracy or completeness of the information. To the extent permitted by applicable law, CA provides this document “as is” without warranty of any kind, including without limitation, any implied warranties or merchantability, fitness for a particular purpose, or non-infringement. In no event will CA be liable for any loss or damage, direct or indirect, from the use of this document, including, without limitation, lost profits, lost investment, business interruption, goodwill, or lost data, even if CA is expressly advised of the possibility of such damages.
Agenda

> Installation & Configuration Standards
> CSF Extensions
Agenda

> This session takes you beyond the standard implementation and looks at how you can build standards into an application to make maintenance easier.

> You will also learn how to create your own CSF extensions so that you can customizing your command and control options to best suite your company’s

> Focus will be primarily on mainframe-hosted scheduling ESP WA 5.5.
Installation and Configuration Standards
Installation and Configuration Standards

> EVENTS
  - Event Naming Standards
  - Consolidate Data Set -triggered Events
  - Regular Clean up of test Events

> CALENDARs
  - Consolidate
  - Merge Special Names
  - Add multiple years – “perpetual”
Installation and Configuration Standards

> ESP Procedures and Applications (APPLs)

- Naming standards: member, APPLs, SUBAPPLs, variables
- Choose an AFTER -or- RELEASE
- Use of AFTER ADD(RELEASE ADD() RESOURCE ADD()
- Use of ENDJOB
- One job in AFTER, one job in RELEASE, on resource in RESOURCE
- JOB and ENDJOB wob keywords always start in column 1
- Order of APPL and JOB statements
- Indentation—JOB statements and IF logic
- EXTERNALs and DSTRIG wob placement
Installation and Configuration Standards

> ESP Procedures and Applications (APPLs)

- SCHEDULED vs. SCOPE for EXTERNALs
- Spacing between APPL level statements and between wobs.
- Naming standards-- TASKs, DSTIGs
  - Maintenance, association
    ```
    JOB TA001.PRODA TASK
    RELEASE ADD(PRODA)
    
    DISTRIG DS001.PRODB
    RELEASE ADD(PRODB)
    ```
- Use IF statement **DO** and **ENDDO** constructs in all cases
Installation and Configuration Standards

APPL PROD1 WAIT POST_OLEDEST JOB_ANCESTOR_WAIT(ANY)
JCLLIB 'PROD.ESP.JCL'
TEMPLIB 'PROD.ESP.TEMPLIB'
COPYJCL 'PROD.ESP.COPYJCL'

JOB PRODEX1 EXTERNAL

DSTRIG xxxxxxx

JOB PROD1
  DELAYSUB xxxx
  RUN xxxx
  RELEASE ADD(xxxx)
  IF xxxx THEN DO
    XXX
    IF XXXX THEN DO
      XXX
      ENDDO
    ENDDO
  ENDDO
  CCCHK xxxxx
ENDJOB
Installation and Configuration Standards

> ESP Procedures and Applications

- Use of SELECT
  - Reduces maintenance if APPL or job groups need rescheduled
  - All jobs or groups of jobs have same schedules

```
SELECT (PAYJOB1, PAYJOB2)
IF TODAY ('WORKDAY') THEN SELECT (PAYJOB1, PAYJOB2)
```
Installation and Configuration Standards

ESP Procedures and Applications

- Use Templates
  - Reduces maintenance if like jobs need requirement change
  - All jobs or groups of jobs have same set of requirements

```
TEMPLATE PAY (1, JOBNAME FREQ(DAILY) NEXT())
  JOB %JOBNAME
  IF FREQ NE "" THEN RUN %FREQ
  IF NEXT NE "" THEN RELEASE(%NEXT)
  TAG PJOB
ENDJOB
ENDTEMPL
PAY PAYJOB1 NEXT(PAYJOB2, PAYJOB3)
PAY PAYJOB2 NEXT(PAYJOB4)
PAY PAYJOB3 FREQ(WEEKDAYS) NEXT(PAYJOB4)
PAY PAYJOB4 NEXT(PAYJOB5)
PAY PAYJOB5 FREQ(LAST WORKDAY OF MONTH)
```
Installation and Configuration Standards

> ESP Procedures and Applications

- Use of SUBAPPLs for...
  - Organization
  - Maintenance
  - Control (hold, force complete, filtering)
  - Additional functionality (wait, request)
- SUBAPPLs in own members (organization, maintenance)
- Common APPL member(s) to invoke SUBAPPLs
Installation and Configuration Standards

> ESP Procedures and Applications

- Combined Job documentation + scheduling criteria
  - Not typically recommended but can serve a need
  - Default DOCLIB format—cannot have

- Symbolics for Agent Names

- APPLENDs
  - Selfcompleting workload objects
  - Automatically perform processing at the end of an Application
  - Reduce maintenance, complexity

  ```
  APPLEND PAYROLL.APPLEND
  RELEASE ADD(LASTJOB1)
  RELEASE ADD(LASTJOB2)
  ENDJOB
  ```
Installation and Configuration Standards

> ESP Procedures and Applications
  - Common Procs and Applications using symbolics and IF logic
    - Organization
    - Control
    - Additional functionality
Installation and Configuration Standards

> CSF and Workstation
  - Common Views: CSF and Workstation
    - ESPCSFA – ispf ISPPROF member
    - C:\Documents and Settings\All Users\Application Data\Cybermation\ESP Workstation\User Profiles
  - PURGSCHF and PURGAPPL
  - Data_Objects for local variables
  - Common RESOURCE to stop all workload
  - *CSF Extensions*
CSF Extensions
CSF Introduction

> First of all... What is CSF: **Consolidated Status Facility** → A TSO ISPF-based ESP tool for monitoring/controlling workload across enterprise.

> Provides realtime status of active and complete jobs and other workload objects

> It is a *user interface*, **which allows commands to entered**, to...
  - Manipulate workload objects
  - Get detailed information on workload objects

> CSF commands are normally issued as line commands next to a listed workload object in a CSF view, or in secondary CSF panels
CSF Introduction
CSF Extensions Introduction
CSF Extensions Introduction

> Now, what are these “extensions”?
  - CSF Extensions are programming interfaces
    - Normally written in REXX
    - “Extend” the list of, and/or capabilities of, existing (built-in) CSF line commands

> Customize CSF by writing your own commands or replace existing CSF line commands

> CSF Extensions become new or modified CSF line commands

> Only supported in the CSF, not Workstation
CSF Extensions Introduction

> Customize CSF to meet your specific needs
> Access other ISPF applications
> Reduce awkward or frequently performed functions to a single command

> Security/verification for current CSF functions:
  - Prevent users from issuing commands
  - Provide confirmation prompts or panels
  - Front-end existing commands to decide whether command should proceed/rejected
  - SAF Resource control (ACF2, Top Secret, RACF)
CSF Extensions: Samples

> As part of the distribution package for ESP Workload Manager, CA provides:

- **Sample** CSF extensions
- A sample **CYBESCSU** input table
- A REXX exec (**CYBESTBG**) to generate an ISPF table

> CSF Extensions are documented in **Part 4** of the **ESP Workload Manager 5.5 Installation and Configuration Guide**
## CSF Extensions: Samples

### Sample CSF Extensions

<table>
<thead>
<tr>
<th>Command</th>
<th>REXX Exec</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAA</td>
<td>CYBESS60</td>
<td>Complete all generations of Application older than specified time</td>
</tr>
<tr>
<td>EDJ</td>
<td>CYBESS61</td>
<td>Edit JCL for job prior to submission</td>
</tr>
<tr>
<td>NX</td>
<td>CYBESS69</td>
<td>Display next 10 executions for an Event associated with job</td>
</tr>
<tr>
<td>TSC,TST,U</td>
<td>CYBESS66</td>
<td>Display information retrieved by RETSCBD() function</td>
</tr>
<tr>
<td>SD</td>
<td>CYBESS63</td>
<td>Invoke SDSF</td>
</tr>
<tr>
<td>SIM</td>
<td>CYBESS64</td>
<td>Simulate Event associated with the job</td>
</tr>
<tr>
<td>SUB</td>
<td>CYBESS65</td>
<td>Submit JCL associated with a job outside of Application</td>
</tr>
<tr>
<td>TR</td>
<td>CYBESS67</td>
<td>Trigger Event associated with job</td>
</tr>
</tbody>
</table>
CSF Extensions: Samples Installation

In order to install sample CSF Extensions you need:

- REXX exec library to store samples
- Name of ISPF table library to store commands

Libraries must be allocated to your TSO session

- Allocate libraries through an exec or CLIST or as part of a logon process
- TSO ISRFIND to list libraries currently allocated to your TSO session; you can re-use this command and specify member CYBESCSU to verify the CSF Extension ISPF table was stored in the correct library – typically an ISPTT- member such as ISPTLIB, ISPTENU.
CSF Extensions: Samples Installation
CSF Extensions: Samples Installation
CSF Extensions: Samples Installation

> Copy sample execs from sample library to REXX exec library which is allocated to SYSEXEC DD

> Copy CYBESTBG exec from sample library to REXX exec library which is allocated to SYSEXEC DD

> Copy CYBESCSU member from sample library to any library you have access to
Update CYBESCSU member by changing the LIBRARY statement to point to a table library that is allocated to your ISPTLIB DD

```plaintext
LIBRARY CYBER.ESP.SSCPTENU
MEMBER CYBESCSU
VARS CMDTABNM CMDTNAME CMDTPARM
KEYS CMDTABNM CMDTNAME
OPTIONS REPLACE
DATA ' '
LC CAA EXEC(CYBESS60)
LC EDJ EXEC(CYBESS61)
LC NX  EXEC(CYBESS69)
LC SD  EXEC(CYBESS63)
LC SIM EXEC(CYBESS64)
LC SUB EXEC(CYBESS65)
LC TR  EXEC(CYBESS67) ISPF
LC TSC EXEC(CYBESS66) RETATR RETJTR ISPF
LC TST EXEC(CYBESS66) RETATR RETJTR
LR U  EXEC(CYBESS66) RETATR RETJTR RETJSR
```
CSF Extensions: Samples Installation

> Run CYBESTBG exec to generate ISPF table library member containing the new commands.

```tsoc
tso cybestbg 'cyb.esp.ssepclst(cybescsu)'
```

- location of CYBESCSCU member is specified (input)
- assumes rexx lib is allocated to your session
CSF Extensions: Implement Your Own

> Libraries you may need to use in your CSF extension and the corresponding DD statement:

- REXX exec library - SYSEXEC
- Table library - ISPTLIB
- Panel library - ISPPLIB
- Message library - ISPMLIB

> Panel and message libraries are required only if you want to present customized panels or messages
CSF Extensions: Implement Your Own

> Write the REXX exec:

```rexx
/* REXX CSFHELLO */
SAY 'Hello There'
```

> Update list of commands in CYBESCSU member:

```rexx
LC HI EXEC(CSFHELLO)
```

> Run CYBESTBG exec:

```rexx
TSO CYBESTBG 'CYB.ESP.SSEPCLST(CYBESCSU)'
```

> Re-enter ESP and test command
CSF Extensions: CYBESCSU Member

> Identify ISPF table library
> Define CSF Extensions and their associated parameters

```
LIBRARY CYBER.ESP.88CPTENU
MEMBER CYBESCSU
VARS CMDTABNM CMDTNAME CMDTPARM
KEYS CMDTABNM CMDTNAME
OPTIONS REPLACE
DATA ' '
LC CAA EXEC(CYBESS60)
LC EDJ EXEC(CYBESS61)
LC NX EXEC(CYBESS69)
LC SD EXEC(CYBESS63)
LC SIM EXEC(CYBESS64)
LC SUB EXEC(CYBESS65)
LC TR EXEC(CYBESS67) ISPF
LC TSC EXEC(CYBESS66) RETATR RETJTR ISPF
LC TST EXEC(CYBESS66) RETATR RETJTR
LR U EXEC(CYBESS66) RETATR RETJTR RETJSR
LC HI EXEC(CSFHELLO)
```
CSF Extensions: CYBESCSU Member

> Use one line per new or replaced command

> Order of command entries does not matter

> Make updates anywhere after the DATA ‘ ’ line

> Specify the following for each command:
  - type of command
  - name of the command
  - process you want executed
  - other options
CSF Extensions: CYBESCSU Member

> Command type determines from where command can be issued

- **LC** - CSF main view line command input fields
- **LR** - LR command’s "secondary" panel commands
- **LX** - L command’s "secondary" panel commands
- **MR** - MR command’s "secondary" panel commands

```
LIBRARY CYBER.ESP.SSCPTENU
MEMBER CYBESCSU
VARS CMDTABNM CMDTNAME CMDTPARM
KEYS CMDTABNM CMDTNAME
OPTIONS REPLACE
DATA ' '
LC CAA EXEC(CYBESS60)
LC NX EXEC(CYBESS69)
LC TR EXEC(CYBESS67) ISPF
LC TSC EXEC(CYBESS66) RETATR RETJTR ISPF
LC TST EXEC(CYBESS66) RETATR RETJTR
LR U EXEC(CYBESS66) RETATR RETJTR RETJSR
MR MM EXEC(CSFTEST)
LX D NOOP
```
CSF Extensions: CYBESCSU Member

> Command names (new or replacement)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBRARY CYBER.ESP.SSCPTENU</td>
<td></td>
</tr>
<tr>
<td>MEMBER CYBESCSU</td>
<td></td>
</tr>
<tr>
<td>VARS CMDTABNM CMDTNAME CMDTPARM</td>
<td></td>
</tr>
<tr>
<td>KEYS CMDTABNM CMDTNAME</td>
<td></td>
</tr>
<tr>
<td>OPTIONS REPLACE</td>
<td></td>
</tr>
<tr>
<td>DATA <code> </code></td>
<td></td>
</tr>
<tr>
<td>LC CAA</td>
<td>EXEC(CYBESS60)</td>
</tr>
<tr>
<td>LC NX</td>
<td>EXEC(CYBESS69)</td>
</tr>
<tr>
<td>LC TR</td>
<td>EXEC(CYBESS67) ISPF</td>
</tr>
<tr>
<td>LC TSC</td>
<td>EXEC(CYBESS66) RETATR RETJTR ISPF</td>
</tr>
<tr>
<td>LC TSY</td>
<td>EXEC(CYBESS66) RETATR RETJTR</td>
</tr>
<tr>
<td>LR U</td>
<td>EXEC(CYBESS66) RETATR RETJTR RETJSR</td>
</tr>
<tr>
<td>MR M</td>
<td>EXEC(CSFTEST)</td>
</tr>
<tr>
<td>LX D</td>
<td>NOOP</td>
</tr>
</tbody>
</table>
CSF Extensions: CYBESCSU Member

> Process Options

- **EXEC(execname)** - invoke REXX exec
- **INVALID** - command is invalid
- **NOOP** - ignore command
- **PGM(pgmname)** - call program - requires load module be accessible to the ISPF session or link-listed

```sql
LIBRARY CYBER.ESP.SSCPTENU
MEMBER CYBESCSU
VARS CMDTABNM CMDTNAME CMDTPARM
KEYS CMDTABNM CMDTNAME
OPTIONS REPLACE
DATA ' '
LC CAA EXEC (CYBESS60)
LC EDJ EXEC (CYBESS61)
LC SD EXEC (CYBESS63)
LC CA NOOP
LC TR EXEC (CYBESS67) ISPF
LC TSC EXEC(CYBESS66) RETATR RETJTR ISPF
LC HI EXEC (CSFHELLO)
```
CSF Extensions: CYBESCSU Member

> Command names can be 1 to 3 characters
  - 1 or 2 characters for LR, LX, and MR command types
    (*i.e. when CSF extensions being created for "secondary" screen input lines*)

> Create a new line command
  - Must use a unique name

> Replacing an existing line command
  - Use an existing command name
  - ESP searches CSF Extensions first before CSF line commands are executed — "concatenation"
  - Replace, front-end / supplement, invalidate
CSF Extensions: CYBESCSU Member

> Other options:

- **PARMS** (parameters)
  - parameter string passed to exec or program

- **ISPF**
  - ISPF services are required (e.g. tables, variables)
    - REXX exec called via an ISPF SELECT function call instead of being invoked directly by CSF
CSF Extensions: CYBESCSU Member

> Other options:

- **RETATR**
  - Retrieve Address space of Application Tracking Record

- **RETJSR**
  - Retrieve Address space of Job Schedule Record

- **RETJTR**
  - Retrieve Address space of Job Tracking Record
CSF Extensions: Return Values

> When a CSF Extension exec ends, CSF examines its return value to determine how to proceed

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOOP</td>
<td>Command successful</td>
</tr>
<tr>
<td>(null) / 0</td>
<td>Command successful (default)</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>Exec internal command with same name</td>
</tr>
<tr>
<td>REJECT</td>
<td>Command failed</td>
</tr>
<tr>
<td>REJECTM</td>
<td>Command failed with message</td>
</tr>
</tbody>
</table>
CSF Extensions: Return Values

The return value can be specified in the Rexx exec using CSFRET or RETURN...

The Rexx RETURN statement is available only if the exec was invoked directly by ESP Workload Manager, and not via the ISPF SELECT function.

If you use the ISPF option for the exec, you must pass back the return value via the CSFRET function.
CSF Extensions: Return Values

**ISPF used**

If RC ≤ 0 then
X = CSFRET(REJECTM)
else X = CSFRET(CONTINUE)

**Direct Invocation**

If RC ≤ 0 then RETURN REJECTM
else RETURN CONTINUE
CSF Extensions: Line command examples

> Invoke a REXX exec:

```
LC SIM EXEC(CSFSIM)
```

> Invalidate an existing command:

```
LC CA INVALID
```

> Access ISPF variables / Display Panels:

```
LC ABC EXEC(MYEXEC) ISPF
```

> Invoke a program:

```
LC LST PGM(LISTIT)
```
CSF Extensions: Functions

> CSF Extensions provide several built-in REXX functions...

- **RETSCBD**
  - returns information about specific jobs stored on the scoreboard

- **PAGEMODE**
  - allows you to enter ESP Workload Manager page mode and execute one or more ESP Workload Manager commands

- **JOBONCSF**
  - Retrieves information about any job currently available on CSF
  - Returns data in the form of REXX stem variables
CSF Extensions: Function Examples

> **REXX**

```
SAY 'THE NAME OF THE EVENT IS ' EVENT()
```

> **RETSDBD**

```
X=RETSDBD()
SAY 'JOBNAME IS ' JOBN
```

> **PAGEMODE**

```
X=PAGEMODE('SIMULATE EV('EVENT()')')
```

> **JOBONCSF**

```
NUM=JOBONCSF('JOBNAME()','X')
SAY 'THERE ARE' NUM 'OCCURRENCES OF JOB' JOBNAME()
DO I=1 TO NUM
    SAY JOBNAME() 'IS IN APPL' XAPPL.I'.XAPPLG.
END
```
CSF Extensions: Common Errors

> Invalid Select Code

- command is not valid
  - ISPF table has not been updated with new command
  - Table library not allocated
  - ISPF session has not been refreshed

> REXX exec load file SYSEXEC does not contain exec member

- command is valid but exec could not be located
  - ensure SYSEXEC contains member
Field Example 1: Prevent CA command

> Invalidate CA (complete application) CSF line command (simply add entry to CSU table and reload CSF commands via CYBESTBG exec):

```
LC  CA  NOOP
LC  CA  INVALID
```

> Usually handled via SAF (ACF2, TSS, RACF).

> Can also apply to the newer TA command
Field Example 1: Prevent CA command

> Another method...

- **REXX exec REJCA**
  
  ```rexx
  /* REXX */
  /* SET CUSTOMIZED MESSAGE AND REJECT COMMAND */
  ISPEXEC "SETMSG MSG(CSFX001)"
  RETURN REJECT
  ```

- **ISPF Message Library member CSFX00**
  
  CSFX001 'CA command not allowed' ALARM=YES
  'The CA command has been disallowed by my extension'

- **Command** *(added to CYBESCSU -- note same name as existing CA cmd.)*

  `LC CA EXEC(REJCA)`
Field Example 2: Add CSF Extension help

> Rexx exec

*/ REXX */
"ISPEXEC TBOPEN CYBESCSU NOWRITE SHARE"
"ISPEXEC TBTOP CYBESCSU"
"ISPEXEC TBDISPL CYBESCSU PANEL(LCSFX)"
"ISPEXEC TBCLOSE CYBESCSU"

> Line Command (CYBESCSU entry)

LC ? EXEC(LCSFX) ISPF
Field Example 2: Add CSF Extension help

> ISPF Panel (LCSFX)

```
)ATTR
/ TYPE(input)  INTENS(HIGH)  COLOR(red)
! TYPE(TEXT)    INTENS(HIGH)  COLOR(white)
$ TYPE(TEXT)    INTENS(LOW)   COLOR(BLUE)  skip(on)
< TYPE(OUTPUT)  INTENS(HIGH)  COLOR(BLUE)
| TYPE(OUTPUT)  INTENS(HIGH)  COLOR(YELLOW)
)BODY
$CMD==> /ZCMD
!
!
ACTIVE CSF EXTENSIONS
!
! CSF Cmd  Cmd Type
$---------  -----------

)MODEL
| cmdtname <cmdtabnm |cmdtparm
)INIT
)PROC
)END
*/
```
Field Example 2: Adding help messages

> Or, alternatively, create an extension that uses SAY instructions to add descriptions...

```/* REXX */
/* COMMAND NAME: EXT */
SAY 'CSF EXTENSIONS CURRENTLY TURNED ON:'
SAY '***********************************'
SAY 'CAA --> COMPLETE ALL GENERATIONS OF AN APPLICATION'
SAY 'EDJ --> EDIT THE JCL FOR A JOB'
SAY 'SD  --> INVOKE SDSF'
SAY 'SIM --> SIMULATE AN EVENT'
SAY 'SUB --> SUBMIT JCL OUTSIDE THE APPLICATION'
SAY 'TR  --> TRIGGER AN EVENT'
SAY 'NX  --> DISPLAY NEXT 10 EVENT EXECUTIONS'
SAY 'EXT --> DISPLAY LIST OF CSF EXTENSIONS'
```
Field Example 3: Bypass job with reason

> Force users to supply a reason when bypassing jobs
  - Replace the BY (Bypass) command
  - Present a panel and user to supply reason, and verify the reason is input
Field Example 3: Bypass job with reason

**REXX**

```
/* REXX -----------------------------------------------------------*/
/* Blank out Variables used in panel by panel */
AA1 = ""
/* Display panel to obtain options */
Address ISPEXEC "DISPLAY PANEL(BYPPANEL)"
If Rc <> 0 Then
  Return
If AA1="" then x=csfret(rejectm)
else ADDRESS ESP "AJ " Job() "BYPASS REASON('"AA1"') APPL("appl()")"
/* Issue AJ command - bypass job and set user status field -------*/
```

**Command**

```
LC BY EXEC(BYPASS) ISPF
```
Field Example 4: Invoke SDSF for job **

> Issue the **SDSF Status** command for the job

```rexx
/* REXX */
/* Command Name: SD */
Address ISPEXEC,
"SELECT PGM(ISFISP) NOCHECK NEWAPPL(ISF)-
  PARM('ST' Jobname() '')"
Return NOOP
```

* Any current active SDSF filtering, such as PREFIX or OWNER, will affect the SDSF output
* You can use a similar technique for accessing other ISPF applications such as IOF, SAR, and so on.

** This example is included with the Sample CSF Extensions **
Field Example 5: Cancel (executing) job

> Issue a cancel command by triggering an ESP Event which invokes a procedure that contains a VS command

```rexx
/* Rexx */
/* Command Name: CJ */
/* CSF extension triggers Event to cancel a job */
/**************************************************************************/
x=retscbd()
say 'ESP will cancel job ' jobn
cmd = "trigger cybcb01.canjob user1("jobno")"
address ESP cmd
```
Field Example 5: Cancel (executing) job

> The procedure invoked by the ESP Event...

```
VS ' $CJ%USER1'
```
Field Example 6: Display Wob’s Script (distributed job)

> If you are using ESP Workload Manager to schedule work on a distributed platform such as Unix, you may wish to obtain the name of the script associated with a workload object.

> The following example traps the output from an ESP SIMULATE command, for a specific workload object, to obtain this information.

> As part of the simulation, two lines relating to the message are sent to the Agent running on the distributed platform. The script name appears as part of this message. Sample output looks like one of the following, depending upon the options used when defining the workload object:

```
Agentmsg 19970318 08194240+0500 SUN * OFFMVS1/PAY.0/MAIN RUN +
.Data(Script=/export/home/scripts/dw,ExitCode=(0-8,s)
---or---
Agentmsg 19970318 08194240+0500 AIX * OFFMVS2/PAY.0/MAIN RUN +
.Data(Script=/u1/dcooper/scripts/dw)
```
/* REXX */
/* Command Name: LX */
X = Retscbd()
Z=outtrap("line.")
"SIM EV(" || Event() || ") SCHED(""Sched'')- ROOT(""Job()"""
Z=outtrap("off")
parse var line.2 something '=' script1 ','
parse var line.2 something '=' script2 ')
if length(script1) < length(script2) then - scriptname=script1
else scriptname=script2
if scriptname = '' then say 'no script for this object'
else say 'scriptname is ' scriptname

• Logic above determines which variable (script1 or script2) contains the
desired script name by comparing lengths of variables resulting from use
of different delimiters.
Field Example 7: Generate an ESP History Report

/* REXX */
/* The exec displays the results in page mode: */
CMD = "REPORT"
CMD = CMD ";SETWIDTH 80"
CMD = CMD ";FROM 8AM TODAY"
CMD = CMD ";CRITERIA APPLSYS EQ " APPLNAME()
CMD = CMD ";DISPLAY JOBNAME JOBNO EXECST CPUTIME"
CMD = CMD ";ENDR"
X = PAGEMODE(CMD)
Field Example 8: Edit a data set **

/* REXX */

"ISPEXEC EDIT DATASET('ABC.PROD.DOCS("JOBNAME()")')"
Field Example 9: Issue LAP for a single job

/* REXX */
address ESP "LAP " APPL() "JOB(" JOB() ")"
Questions ?