

IBM Copy Services Manager White Paper

IBM Copy Services Manager session automation

This document can be found on the web: www.ibm.com/support/techdocs

Author: Thomas Luther
IBM Consulting IT Specialist

Version 1.1, 17. Jan 2020



IBM® Systems
Storage Platform



Disclaimer and Trademarks

No part of this document may be reproduced or transmitted in any form without written permission from IBM Corporation. Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This information may include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or programs(s) at any time without notice. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATIONS "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

IBM shall have no responsibility to update this information. IBM products are warranted according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. IBM is not responsible for the performance or interoperability of any non-IBM products discussed herein. The performance data contained herein was obtained in a controlled, isolated environment. Actual results that may be obtained in other operating environments may vary significantly. While IBM has reviewed each item for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Statements regarding IBM's future direction and intent are subject to change or withdraw without notice, and represent goals and objectives only.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

IBM®, the IBM logo are trademarks of the International Business Machines Corporation in the United States, other countries, or both. A full list of U.S. trademarks owned by IBM may be found at <http://www.ibm.com/legal/copytrade.shtml>

Microsoft®, Windows® are registered trademarks of Microsoft Corporation.

Other company, product, or service names may be trademarks or service marks of others.

Copyright © 2020 by International Business Machines Corporation.

A Note to the Reader

This White Paper assumes familiarity with the general concepts of IBM DS8000 Copy Services, in particular Metro Mirror, Global Mirror, Global Copy and Multi Target PPRC.

Additionally, the reader will find it helpful to be familiar with general usage and concepts of IBM Copy Services Manager (or the former product TPC for Replication) as well as its CLI component (CSMCLI).

The given script examples as well as the scripting best practices focus on REXX scripting as used on IBM z/OS. However, the discussed scripting concepts can also be implemented on another platform (e.g. via a REXX interpreter on Windows) or in another scripting language as commonly used on platforms like Linux (e.g. Perl, Bash,...). Readers with general scripting knowledge will find it easier to follow the discussed scripting concepts and should be able to adopt those to other scripting languages as well.

CSMCLI scripting on z/OS was selected as example, because it has some specifics that need to be mentioned and the various, not commonly known options on z/OS for scripted CSMCLI execution were the original driver to create this paper.

The CSMCLI scripting part of this paper is applicable to any CSM release. The CSMCLI framework specifics discussed in this paper are valid up to CSM release 6.2.2, but might change and improve in future releases.

The new CSM Scheduled Task feature which is discussed in the second part of this paper is applicable for CSM release 6.2.2 and later. Future CSM releases might further enhance the CSM Scheduled Tasks capabilities, and as such further simplify some automation topics described in this context.

For readers unfamiliar with these topics and for additional information, please refer to references listed in the chapter '5 References' on page 52.

Acknowledgements

The author would like to say "thank you" to Randy Blea for his support in clarifying various topics described in this paper. I also want to thank Pascal Fusellier for helping me to implement and validate the scripted scenario used as example in this paper.

Document History

Version/Date	Remarks
V1.0 28/03/2018	Initial release using Copy Services Manager 6.2.2
V1.1 17/01/2020	Updated using Copy Services Manager 6.2.7

Table of contents

1	Purpose of Document	7
2	Session automation via CSMCLI scripting	8
2.1	CSMCLI scripting options on z/OS	8
2.1.1	Option 1: Using IKJEFT01 with OSHELL	8
2.1.2	Option 2: BPXBATCH	9
2.1.3	Option 3: Rexx script, utilizing BPXWUNIX function or Syscalls ..	10
2.2	CSMCLI scripting option on Windows: Rexx interpreter	13
2.2.1	Adoptions of csmcli.bat for scripting	13
2.3	CSMCLI framework characteristics	14
2.3.1	CSMCLI Return Code handling	14
2.3.2	Error message routing.....	15
2.3.3	Message Prefixes	15
2.4	CSMCLI scripting best practices	16
2.4.1	Adopt common CSMCLI Java options as necessary.....	16
2.4.2	Password-less CSMCLI execution	16
2.4.3	Password-less CSMCLI execution via z/OS Security Authentication Facility.....	17
2.4.4	Parameterized scripting	19
2.4.5	CSMCLI call wrapper	19
2.4.6	Common set of reusable procedures	19
2.5	Script example.....	21
2.5.1	Program Flow overview.....	22
2.5.2	Script execution	22
2.5.3	Script Return codes.....	23
2.5.4	Script Runtime environment.....	23
2.5.5	Script Parameters.....	23
2.5.6	Script execution via JCL	24
2.5.7	Script customization	24
2.5.8	Script output control	25
2.5.9	Procedure overview	26
3	Session automation via CSM Scheduled Tasks	33
3.1	Scheduled Task introduction	33
3.2	Create Scheduled Task with multiple actions	34
3.3	Modify Scheduled Task with multiple actions	40
3.4	Manage Scheduled Tasks	42
3.4.1	Scheduled execution.....	43
3.4.2	On demand execution	44
3.4.3	Task monitoring	44
3.5	Remove Scheduled Tasks	47

3.6	Advanced Scheduled Task example	48
3.6.1	Task and Actions definition	48
3.6.2	Task execution	49
4	CSM session automation conclusion	51
5	References	52
5.1	CSM and DS8000 Copy Services	52
5.2	Rexx Scripting	52
6	Appendix: REXX Script example	53
7	Appendix: Output of REXX Script example.....	76

1 Purpose of Document

IBM Copy Services Manager offers a Command Line Interface (CSMCLI) which can be used for automated session management. This becomes quite useful, if you need to integrate CSM actions into higher level automation routines or batch environments, e.g. for creation of a Practice Copy at a certain time.

Beside running individual (or a series of) scripted CSMCLI commands, starting with CSM 6.2.1 there is a new Scheduled Task capability which can also be utilized to accomplish a certain degree of session automation.

In this paper we will discuss usage and best practices for both session automation options. In the scripting part, we will focus on CSMCLI scripting options on z/OS, since automated execution of Unix System Shell commands might not be commonly known for mainframe system programmers or storage administrators. Therefore the used script language in this paper is Rexx, which is available on z/OS platforms and can be utilized for integration into batch environments.

Rexx however is not only available on z/OS, but can also be used on other platforms via commonly available, portable Rexx interpreters, e.g.

- Regina Rexx (<https://regina-rexx.sourceforge.io/>)
- Brexx (<https://sourceforge.net/projects/brexx/>)
- Open Object Rexx (<http://www.oorexx.org/about.html>)

Throughout this document we will be using a concrete example scenario for CSM session automation. This scenario leverages a DS8000 4 site replication topology, that is managed by a CSM 4 site replication session with Multi Target MM-GM with a cascaded Global Copy replication from the GM target site. In the example scenario we will create a consistent copy of data on the 4th site for testing purposes while restarting the Global Mirror leg afterwards. This explicit example scenario is described in more detail in chapter '2.5 Script example'.

2 Session automation via CSMCLI scripting

2.1 CSMCLI scripting options on z/OS

The CSMCLI can be installed on a z/OS LPAR, independently where the actual CSM server is running. You can download any CSMCLI installation image from IBM Fix Central. Follow the '[Latest Downloads for IBM Copy Services Manager](#)' page for a link to the appropriate version:

- <http://www-01.ibm.com/support/docview.wss?uid=ssg1S1005482>

Follow the instructions in the [CSM Knowledge Center to install CSMCLI separately](#) on a z/OS LPAR (CSMCLI is already installed along with the CSM server on an LPAR)

- https://www.ibm.com/support/knowledgecenter/en/SSESK4_6.2.7/com.ibm.storage.cs.m.help.doc/csm_installing_cli_on_remote_system.html

The CSMCLI is a Java application which is installed in the z/OS Unix System Space (USS). z/OS provides various options to execute z/OS Unix shell commands from a job, which is required to include CSMCLI commands into the z/OS batch environment.

For a scripted automation it is recommended, that a CSMCLI authentication properties file is prepared in the appropriate user HOME folder under the `./csm-cli/` subdirectory. The authentication properties file contains a CSM username and the corresponding encrypted password, which prevents that authentication details must be included in the script itself. Please refer to chapter '2.4.2 Password-less CSMCLI execution' for more details.

Following sections discuss 3 different options how CSMCLI commands can be executed from a z/OS job.

2.1.1 Option 1: Using IKJEFT01 with OSHELL

The IKJEFT01 program is used to launch TSO commands. OSHELL is a TSO command that launches a temporary z/OS Unix Shell and can pass multiple commands to this temporary shell.

Advantages:

- Allows to specify USS environment parameters for multiple commands, e.g. for the CSMCLI executable path and HOME folder. Specifying the HOME folder prevents that each TSO user who is submitting the job must have its own CSMCLI authentication properties file.
- Multiple commands can be passed to same shell

Disadvantages:

- Output can only be directed to Joblog or a specified dataset.
- Limited conditional based execution possible for multiple OSHELL commands

OSHELL as well as the Unix shell commands must be passed via the input DD statement of the IKJEFT01 program. Each OSHELL command must be separated with a semicolon

‘;’ and all except the last line in the SYSTSIN DD statement must be concatenated with a dash ‘-’ to indicate that the input statement is a single line.

Note: The JCL ISPF editor might need to be enabled to allow lower case characters for OSHELL commands since the Unix System Shell is case sensitive.

Examples how to utilize IKJEFT01 for CSMCLI commands:

In the first example, the path of the csmcli.sh executable must be included in the USS shell PATH variable of the TSO user that is executing the job. Additionally the CSMCLI authentication properties file must either exist in the TSO Users home folder or the CSMCLI home folder in USS. Otherwise the job would fail because either csmcli.sh is not understood as valid USS command/executable or because authentication to the CSM server fails. This example job will list the role pairs for a specific session in the job log:

```
//CSMCLI01 JOB MSGCLASS=T,CLASS=T,NOTIFY=&SYSUID,REGION=0M
//*
//PAX1      EXEC   PGM=IKJEFT01
//SYSPROC   DD    DISP=SHR,DSN=SYS1.SBPXEXEC
//SYSTSPRT  DD    SYSOUT=*
//STDOUT    DD    SYSOUT=*
//SYSTSIN   DD    *
OSHELL -
  csmcli.sh -noinfo -server 1.2.3.4 lsrolepairs -l mySession ;
```

The second example shows how to use independent HOME and PATH settings for the temporary shell, which will ensure the csmcli.sh executable is found through the path and the CSMCLI is looking for the authentication properties file in the specified HOME folder. Additionally it demonstrates how CSMCLI commands can be customized with shell parameters, like defining variables for the active server address and the session Name:

```
//CSMCLI01 JOB MSGCLASS=T,CLASS=T,NOTIFY=&SYSUID,REGION=0M
//*
//PAX1      EXEC   PGM=IKJEFT01
//SYSPROC   DD    DISP=SHR,DSN=SYS1.SBPXEXEC
//SYSTSPRT  DD    SYSOUT=*
//STDOUT    DD    SYSOUT=*
//SYSTSIN   DD    *
OSHELL -
  export HOME=/u/myuser ; -
  export PATH="$PATH;/opt/IBM/CSM/CLI/" ; -
  export csmserverA=1.2.3.4 ; -
  export sessname="mySession" ; -
  csmcli.sh -noinfo -server $csmserverA lsrolepairs -l $sessname ;
```

2.1.2 Option 2: BPXBATCH

The program BPXBATCH can be used to run shell scripts and executables in USS directly. It does not require to open a USS shell first. For more details refer to the z/OS knowledge center:

- https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.bpxa500/bpxbatr.htm

Advantages:

- Simple formatting for multiple shell commands

Disadvantages:

- BPXBATCH 'sh' can run only executable scripts, which each run in its own shell.
- Output can be redirected to job log or specified datasets/files only
- Very limited conditional based execution of multiple commands

The syntax of the BPXBATCH program requires that each parameter line must be started with 'sh' followed by the executable. Use ';' as separators between multiple shell lines in the Parameter DD statement. 'sh' causes BPXBATCH to execute a login shell which runs the /etc/profile script (and runs the user's .profile file). These can be modified to include common path changes to find the CSMCLI executable.

Common environment settings for the shell can be defined with the STDENV statement, either as data stream in the JCL or a given dataset or Unix System file that include the environment parameters. However, The ENV parameter changes are not affective when the 'sh' executes the called program, only for sub scripts called by the executable.

Note: The JCL ISPF editor might need to be enabled to allow lower case characters for SHELL commands since the Unix System Shell is case sensitive.

Example how to utilize BPXBATCH for CSMCLI commands:

Following example shows how to use BPXBATCH to issue shell commands. In this case, the STDENV definition is not used, therefore the path of the csmcli.sh executable must be included in the USS shell PATH variable of the TSO user that is executing the job. Additionally the CSMCLI authentication properties file must either exist in the TSO Users home folder or the CSMCLI home folder in USS. Otherwise the job would fail because either csmcli.sh is not understood as valid USS command/executable or because authentication to the CSM server fails.

This example job will list the defined CSM HA servers and the configured sessions in the job log:

```
//CSMCLI02 JOB MSGCLASS=T,CLASS=T,NOTIFY=&SYSUID,REGION=0M
//*
//BPXBAT EXEC PGM=BPXBATCH,REGION=0M
//STDIN DD DUMMY
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//STDENV DD DUMMY
//SYSOUT DD SYSOUT=*
//STDPARM DD *
sh csmcli.sh -noinfo lshaservers ;
sh csmcli.sh -noinfo lssess ;
```

2.1.3 Option 3: Rexx script, utilizing BPXWUNIX function or Syscalls

Using a scripting language such as Rexx provides most flexibility in executing USS commands and analyzing output for conditional based execution. Rexx provides a very useful function to execute USS commands and capture standard and error output streams

in stem variables for subsequent parsing of the output. It also allows to specify stem variables for the environment variables to be used for the USS call. That means the environment stem variable can be used to specify the path to the CSMCLI executable as well as the HOME folder, where CSMCLI is looking for the authentication properties file that contains the user and encrypted password.

For more details on BPXWUNIX, please refer to these links:

- https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.bpxb600/wunix.htm
- https://www.ibm.com/developerworks/community/blogs/MartinPacker/entry/bpxwunix_z_os_best_kept_secret10?lang=en_us

If Rexx is running on z/OS, you can also switch the Rexx execution address space dynamically to make USS system calls, which lets you directly invoke shell commands. This is a very convenient way to issue shell commands when no specific environment variables or output parsing is required.

The Rexx script itself can then be executed from TSO or as job in a batch environment.

Advantages:

- Executes a shell command from a Rexx script with customizable environment parameters
- Captures output in stem variables for subsequent parsing
- Complex conditional execution can be coded easily in Rexx
- Full CSMCLI output can be hidden from User to prevent filling Job logs

Disadvantages:

- Users must be familiar with Rexx scripting to a certain degree

Examples for Rexx BPXWUNIX function and syscalls:

The following example script demonstrates how shell commands can be executed via syscalls. For instance, it shows how the USS 'sleep' command can be utilized to wait a specific number of seconds, e.g. when delays must be added between subsequent CSMCLI commands (see lines 70-81).

The example also demonstrates a very basic approach for a CSMCLI wrapper procedure (see USSCMD procedure), which utilizes bpxwunix and stores the output for subsequent parsing and conditional execution of CSMCLI commands (see lines 50-67).

This specific Rexx script can be used to kick off the resynchronization of the Global Mirror leg of a Multi Target MM-GM session. It will verify whether the actual session state is either Suspended or Target Available, and then restart GM from the active Host site to H3. It can be used for an automated Global Mirror resynchronization after a planned or unplanned HyperSwap. For instance, system automation could catch the HyperSwap completion message IOSHM0414I and execute this Rexx via a Job.

```
0001 /* REXX */
0002 address tso
0003 debug = 0 /* Set to 1 for more debug output */
0004
0005 /* Verify if USS syscalls are possible from Rexx */
```

```

0006 if syscalls('ON') > 3 then do
0007     say 'Unable to establish the SYSCALL environment'
0008     return 12
0009 end
0010
0011 /*IOSHM0414I 11:50:20.07 PLANNED HYPERSWAP COMPLETED*/
0012 /*IOSHM0414I 11:50:20.07 UNPLANNED HYPERSWAP COMPLETED*/
0013
0014 /* Setup environment for CSMCLI calls */
0015 env.0 = 2 /* Qty of env variables */
0016 env.1 = "HOME=/u/username" /* Home for auth file */
0017 env.2 = "PATH=/opt/IBM/CSM/CLI/" /* Path to csmcli.sh */
0018 cli = "csmcli.sh -noinfo" /* default executable */
0019
0020 session = "MT-MM-GMP" /* Name of Session */
0021
0022 say 'Listing session:' session
0023 cmd = cli "showsess" session
0024 call USSCMD(cmd)
0025 do i=1 to out.0
0026     /* Parse State and Active Host of session */
0027     if pos("State",out.i) > 0 then do
0028         sess_state = word(out.i,2) word(out.i,3) /* up to 2 words */
0029     end
0030     else if pos("Active Host",out.i) > 0 then do
0031         sess_host = word(out.i,3)
0032     end
0033 end
0034 say "State: " sess_state "; Host: " sess_host
0035 if left(sess_host,1) = "H" & ,
0036     (sess_state = "Target Available" | sess_state = "Suspended") then do
0037
0038     cmd = cli "cmdsess -action startgm_" || sess_host || ":h3" ,
0039         "-quiet" session
0040     totrc = USSCMD(cmd)
0041     if totrc = "0" then do
0042         say session "started successfully from" sess_host || ":H3"
0043     end
0044     else do
0045         say "Session" session "failed to start" sess_host || ":H3"
0046     end
0047 end
0048 return totrc
0049
0050 /*-----*/
0051 /*- SUBROUTINE -*/
0052 /*- -*/
0053 /*- Call USS with specified cmd -*/
0054 /*- Eg: call USSCMD command */
0055 /*-----*/
0056 USSCMD:
0057     parse arg command
0058     if debug > 0 then say "CMD:" command
0059     myrc = bpxwunix(command,,out.,err.,env.)
0060     do i=1 to out.0
0061         if debug > 0 then say strip(out.i)
0062     end
0063     do i=1 to err.0
0064         if debug > 0 then say strip(err.i)
0065     end
0066     if debug > 0 then say "RC:" myrc
0067 return myrc
0068
0069
0070 /*-----*/

```

```

0071 /*- SUBROUTINE                                     -*/
0072 /*-                                                 -*/
0073 /*- Wait x seconds, utilizing USS system call        -*/
0074 /*- Eg: call WAIT(2)                               */
0075 /*-----*/
0076 WAIT:
0077     parse arg seconds
0078     address syscall
0079     'sleep (seconds)'
0080     address tso
0081     return 0

```

2.2 CSMCLI scripting option on Windows: Rexx interpreter

Windows by itself provides limited scripting language support for more advanced CSMCLI scripting. However, Rexx is a script language that can be ported to various platforms, similar to Perl, Bash and others. You just need to install a platform specific Rexx interpreter in order to run Rexx scripts on windows. You can find some interpreters listed in chapter '5.2 Rexx Scripting' which can be installed on windows to execute Rexx scripts.

Note: If platform specific system calls are used, you will need to adopt your script accordingly. It depends on the Rexx interpreter how system calls are implemented.

The example script listed in chapter '6 Appendix: REXX Script example' was tested with Regina Rexx on Windows. It determines the used platform (TSO or Windows) and performs conditional execution of platform specific system calls. That allows you to run the script on a Windows platform as well as on a z/OS LPAR, wherever the CSMCLI is installed. In order to run it either on z/OS or Windows, you just need to adopt a few environment variables without changes to the script itself.

2.2.1 Adoptions of csmcli.bat for scripting

As of CSM 6.2.7, the csmcli.bat executable is more optimized to launch an interactive CSMCLI command window within Windows, but less for customized scripting. Following *.bat lines may cause problems with external scripting:

- TITLE IBM Replication Manager CLI
 - This will modify the Window title of the command window from where you execute your script. You can add REM at the beginning of the line to comment it and prevent its execution.
- if not %ERRORLEVEL% == 0 pause
 - This will wait for key input when the CSMCLI framework returns an error. Your script may appear to hang in that case. You can add REM at the beginning of the line to comment it and prevent its execution.
- The return code from the CSMCLI framework is not passed back to the caller.
 - You may add following line at the end of the script:

```
EXIT /B %ERRORLEVEL%
```

These topics are addressed and may be changed in future versions of the csmcli.bat file.

Note: Any customization you do in the CSMCLI executables are overwritten during a CSMCLI upgrade.

2.3 CSMCLI framework characteristics

A scripted CSMCLI automation relies on consistent CLI framework handling in regards to error codes, messages and printing to output/error streams in order to validate warnings or errors from issued CSMCLI commands. When you start to script conditional CSMCLI execution, you need to be aware of some general CSMCLI framework characteristics.

Note: The following characteristics apply up to CSM 6.2.7 release. Future releases might change/improve some of the characteristics to simplify conditional based scripting.

2.3.1 CSMCLI Return Code handling

The CSMCLI return code in general does not reflect whether a command was executed successfully by the server. A CSMCLI return code of 0 reflects only that the CLI framework could be started and authenticated against the CSM server, that the command is valid and properly formatted, and that it has been issued successfully against the server. It does not tell whether the command was executed successfully or returned with a warning or error message. That means a reliable script solution needs to evaluate the CLI return code, as well as the standard and error output streams for any CSM Error message returned from the server.

Following return codes are documented for the CSMCLI:

- https://www.ibm.com/support/knowledgecenter/SSESK4_6.2.7/com.ibm.storage.csm.help.doc/fre_c_output2.html

Code	Category	Description
0	Success	The command was successful.
2	Syntax error	The syntax of the command was not correct.
3	Connection error	A connectivity error or protocol error occurred.
4	Server error	An error occurred during a function call to the application server.
5	Authentication error	An error was detected during authentication checking.
6	Application error	An error occurred during processing that is performed by the MetaProvider client application.

Following are examples of different and undocumented return codes as of CSM 6.2.7. Additional information for the error might be printed to either the standard output or the error output stream. In these examples, the error output stream is shown after the 'ERR:' indicator.

The CSMCLI framework currently does not distinguish between authentication errors (RC = 5) and connection errors (RC = 3), it always returns undocumented RC = 8 (Login Failed).

For wrong authentication on an existing CSM server, it should return 5, but returns 8:

```
E:\CSM\Cli>csmcli -noinfo -server 1.2.3.4 cmdsess -action suspend test
Login failed for: username=dummyuser, server=1.2.3.4, and port=9560.
ERR:
RC=8
```

For connection errors, it should return 3, but returns 8:

```

E:\CSM\Cli>csmdi -noinfo -server 10.10.10.10 cmdsess -action suspend test
Login failed for: username=myuser, server=10.10.10.10, and port=9560.
ERR:
RC=8
E:\CSM\Cli>ping 10.10.10.10

Pinging 10.10.10.10 with 32 bytes of data:
Request timed out.

Ping statistics for 10.10.10.10:
    Packets: Sent = 1, Received = 0, Lost = 1 (100% loss)

```

A solid CSMCLI wrapping procedure needs to evaluate the CSMCLI return code and possibly add more error details from either the error or standard output stream to indicate the problem. For instance, if the return code is not 0, the wrapping procedure should return the first line from the error output stream, or if empty, return the first line from the standard output stream.

2.3.2 Error message routing

As documented in the CSM Knowledge Center, all error messages should go to the error output stream.

- https://www.ibm.com/support/knowledgecenter/SSESK4_6.2.7/com.ibm.storage.csm.help.doc/fre_c_output2.html

However, some IWNxxxxxE messages seem to be printed to the standard output stream instead of the error output stream. Following example shows an IWNxxxxxE error message that is printed to standard output (Note: ERR: shows the error Output stream, which is empty):

```

E:\Projects\CSM\Cli>csmdi -noinfo cmdsess -action suspend test
IWNR1527E [Jan 30, 2018 10:46:28 AM] The available commands for session test
could not be obtained because the session does not exist.
ERR:
RC=0

```

A solid CSMCLI wrapping procedure needs to parse the standard output as well as the error output stream for IWNxxxxxE messages to evaluate whether the CSMCLI command execution resulted in an error.

2.3.3 Message Prefixes

As documented in the CSM Knowledge Center, all CSMCLI messages are supposed to start with IWNxxxx.

- https://www.ibm.com/support/knowledgecenter/SSESK4_6.2.7/com.ibm.storage.csm.help.doc/frg_r_message_codes.html

However, as of CSM 6.2.7 some error messages from the CLI framework might return with the prefix CMMxxxxxE. These are printed to the **error output stream** and mainly occur on command syntax errors or incorrect/missing parameters. In those cases, the CSMCLI return code is also <> 0.

Following is an example for a command with a missing parameter:

```
E:\CSM\Cli>csmdi -noinfo cmdsess -action suspend
ERR:
CMMCI9022E Missing required parameter: session_name.
Usage: cmdsess [ { -help|-h|-? } ] [-quiet] -action command [-restorefrom
snapshot_group_name] session_name | - [-priority 1|2|3|4] [-newname
snapshot_group_name]
Tip: Enter "help cmdsess" for more information.
RC=2
```

Following is an example for a wrong command:

```
E:\CSM\Cli>csmdi -noinfo lspairs -l -rolepair h1:j3 SG_MMGM
ERR:
CMMCI9013E Command: lspairs was not found.
Tip: Enter "help" for a list of available commands.
RC=1
```

A solid CSMCLI wrapping procedure needs to consider this additional message prefix when parsing the error output stream for error message codes.

2.4 CSMCLI scripting best practices

This section describes some common best practices which are recommended when scripting more complex CSMCLI sequences, which need to include conditional based execution.

2.4.1 Adopt common CSMCLI Java options as necessary

Starting with CSM 6.2.7, there is a new `javaoptions.properties` file that can be used to modify parameters for the Java framework used by CSMCLI. For details on the properties file, please refer to the CSM Knowledge Center:

- https://www.ibm.com/support/knowledgecenter/SSESK4_6.2.7/com.ibm.storage.csm.help.doc/frg_r_javaoptions_properties_file.html

When you use an IBM Java Runtime Environment, you can utilize the Xquickstart option. This enables a faster start of the JRE, which is especially useful for short term Java applications like scripted CSMCLI single shot commands. Especially on z/OS you may benefit from 10-15% faster starts of the JRE for CSMCLI execution.

You may also want to change the language used by the CSMCLI for your scripted execution, since the CSMCLI will otherwise use the default locale as determined by the JRE.

Note: For previous CSMCLI releases, such JRE property changes can also be done in the CSMCLI executable directly. However, any customization you do in the CSMCLI executable are overwritten during a CSMCLI upgrade. Therefore, modifications of CSMCLI executables are not fully supported.

2.4.2 Password-less CSMCLI execution

A common good practice for CSMCLI scripting is to avoid that you put user names and/or passwords somewhere in a script. They would be unencrypted and human readable and

as such usually violate common security policies. The CSMCLI supports following options for password less scripting.

2.4.2.1 Password-less CSMCLI execution via authentication properties file

Independent of the CSMCLI server platform, the CSMCLI supports an authentication properties file, which can contain a CSM user name and the corresponding password. The password will be encrypted in the properties file by the first use of the CSMCLI executable. For more details, please refer to the CSM Knowledge center:

- https://www.ibm.com/support/knowledgecenter/SSESK4_6.2.7/com.ibm.storage.csm.help.doc/frg_t_settingup_automatic_authentication.html

There are 2 locations where the CSMCLI looks for an authentication properties file during launch. The location is case sensitive for non Windows platforms:

- The operating system user defined HOME folder, e.g.
<HOME>/csm-cli/csmcli-auth.properties
- The CSMCLI installation folder, e.g.
<CSMINSTALLPATH>/CLI/csmcli-auth.properties

In order to use a specific csmcli-auth.properties file location, you simply can set the <HOME> environment variable accordingly within the script when it opens a system shell to execute the CSMCLI command. You just need to ensure that the operating system user who is executing the script (or Job) will have read permissions to the csm-cli/csmcli-auth.properties file in the specified <HOME> folder.

For auditing purposes, it is also recommended to setup a functional user on the CSM servers that is used only for scripting purposes, e.g. '*scriptuser*'. The '*scriptuser*' authentication would then be defined in the authentication properties file used by your CSMCLI scripts. As such, you can audit in the CSM console log whether certain commands have been issued by dedicated GUI/CLI users, or via any scripts using the functional '*scriptuser*'.

2.4.3 Password-less CSMCLI execution via z/OS Security Authentication Facility

Starting with CSM 6.2.5, another password less option exists when the CSMCLI client and the CSM server are both running on LPARs that belong to the same z/OS Security Authentication Facility (SAF) Sysplex. This can be utilized for example, if the CSM CLI and server are both running on LPARs that share a common RACF database for security management. It provides the additional benefit, that a TSO user password update does not require an update of the password in the CSMCLI authentication properties file.

A Security Facility Class can be setup in z/OS (e.g. in RACF) to indicate which users have authorization to run certain programs. The CSMCLI allows now to use the specified Facility Class to authenticate the currently logged in OMVS shell user (or the user specified in the BPXBATCH job that runs the CSMCLI command). If that user is in the Facility Class, a password will not need to be specified.

Following are the high-level steps to create z/OS Security Facility authentication for the CSMCLI:

- Create z/OS security facility for CSMCLI
 - Utilize the IWNRAFC sample job to simplify setup:
 - https://www.ibm.com/support/knowledgecenter/SSESK4_6.2.7/com.ibm.storage.csm.help.doc/frc_t_config_iwnracf.html
 - Define the security facility (default IWNSRV.CLIAUTH):

```
RDEFINE FACILITY IWNSRV.CLIAUTH UACC(NONE)
```

- Provide read access for required users or groups to the security facility, e.g.:

```
PERMIT IWNSRV.CLIAUTH CLASS(FACILITY) +  
ID(#userid) ACCESS(READ)
```

- The SAF native calls require that the SAF users and the CSM server task (IWNSRV) owner be granted access to BPX.SERVER and BPX.DAEMON

```
RDEFINE FACILITY BPX.SERVER UACC(NONE) OWNER(SYS1)  
PERMIT BPX.DAEMON CLASS(FACILITY) ID(#userid) ACCESS(NONE)  
PERMIT BPX.SERVER CLASS(FACILITY) ID(#userid) ACCESS(READ)  
PERMIT BPX.SERVER CLASS(FACILITY) ID(#serverid) ACCESS(READ)
```

- Refresh the security facility

```
SETROPTS RACLIST(FACILITY) REFRESH
```

- CSMCLI SAF setup under Unix System Services (OMVS):

- Give program control to the Java executable of the CSM server:

```
extattr +p <CSM_ProductionRoot>/Java/bin/java
```

- Note: To be able to use the extattr +p command, the OMVS user must have at least read access to the BPX.FILEATTR.PROGCTL resource in the FACILITY class.

- Ensure following authority for the CLI folders (should be default setting from installation):

```
chmod 775 <CSMCLI_ProductionRoot>  
chmod 775 <CSMCLI_ProductionRoot>/CLI  
chmod 775 <CSMCLI_ProductionRoot>/CLI/cliTrace.log
```

- Give the following authority for the history file in users home directory:

```
chmod 760 <#userid>/.sh_history
```

- Update the CSM configuration file to enable the SAF login module:

```
<CSM_ProductionRoot>/wlp/usr/servers/csmServer/properties/csm.conf
```

- The default of the first line in the file is CSMDisabled, which you need to replace with CSMServer
- The SAF Login Module line needs to be added

```
CSMServer {  
com.ibm.csm.server.security.CertificateLoginModule SUFFICIENT;  
com.ibm.csm.server.security.SAFLoginModule SUFFICIENT;  
com.ibm.csm.server.security.WebSphereLoginModule SUFFICIENT;  
};
```

- Restart CSM server to activate the changes (IWNSRV job)

Once this setup has been done, a one time SAF authentication can be used with the -saf parameter of the CSMCLI executable:

```
csmcli.sh -saf
```

Following are some limitations on usage of the SAF authentication as of CSM 6.2.7:

- SAF authentication for CSM authorized group users might fail unless the GUI has been logged in to already after CSM server start up. This is because only the first GUI

login does an extra initialization that sets up the group-to-role mapping in Copy Services Manager.

- If the CSM server address space owner (IWNSRV) does not have UID(0) access or permission granted to authorized services, the WebSphere Liberty Angel Process must be used. Otherwise the procedure fails and marks the process as program controlled.
- LDAP authentication must be disabled for SAF usage (default on z/OS CSM servers)

2.4.4 Parameterized scripting

In order to create re-usable scripts that are easy to customize, you should consider the definition of variables at the beginning of the script for each parameter that should be customizable. Optionally you can allow to specify more dynamic parameters as arguments to the script upon execution. Some common customizable parameters could be the CSM server name/IP, session names, timeouts, CSMCLI environment variables, a default CSMCLI execution prefix, the delimiter char for CLI output separation, etc.

Once you have developed and tested such a parametrized CSMCLI script, you can easily adopt it to different environments or different session names, without the need to make any changes in the tested script itself. Examples of such variable definitions can be found in chapter '6 Appendix: REXX Script example', see lines 55-90. Examples of script argument parsing for more dynamic parameters can be found in lines 106-155.

2.4.5 CSMCLI call wrapper

As discussed in chapter '2.3 CSMCLI framework characteristics', it is not that easy to programmatically validate a successful execution of a CSMCLI command. Therefore you should consider a central, re-usable CSMCLI call wrapper procedure, which does the required system shell call with the provided CSMCLI command, but also parses the output and evaluates whether the command execution might have caused an error. A meaningful error message can then be passed back to the caller of the CSMCLI wrapper, while the output of the CSMCLI command is held in variables to be further evaluated by your script if the command was successful.

By using such a common CSMCLI wrapper procedure you can handle all CSMCLI framework specifics in a central place. This enables modifications in a single place if framework characteristics should change in future releases.

An example of such a CSMCLI wrapper procedure can be found in chapter '6 Appendix: REXX Script example', see lines 1204-1282. It returns 0 if the CSMCLI command has been completed successfully. Otherwise it might return the CSM Error message grepped from the output, or the CSMCLI framework return code with some additional error information.

2.4.6 Common set of reusable procedures

When developing more complex CSMCLI scripting, you might find some steps in the script to become repetitive. You also might want to use consistent output formatting for your script, or easily enable a debug mode with more detailed output during development and test without modifying a lot of places in your script for changing the debug mode or the output formatting. Another common set of procedures could be to check a CSM

session, role pair or pairs for a specific state, or optionally wait up to a given timeout until they reach a specific or expected state.

It really simplifies your script development if you identify such re-usable functions of the script and create parameterized procedures for them to be re-used in various places of your script.

The example script in chapter '6 Appendix: REXX Script example' contains many of such common, reusable procedures, such as:

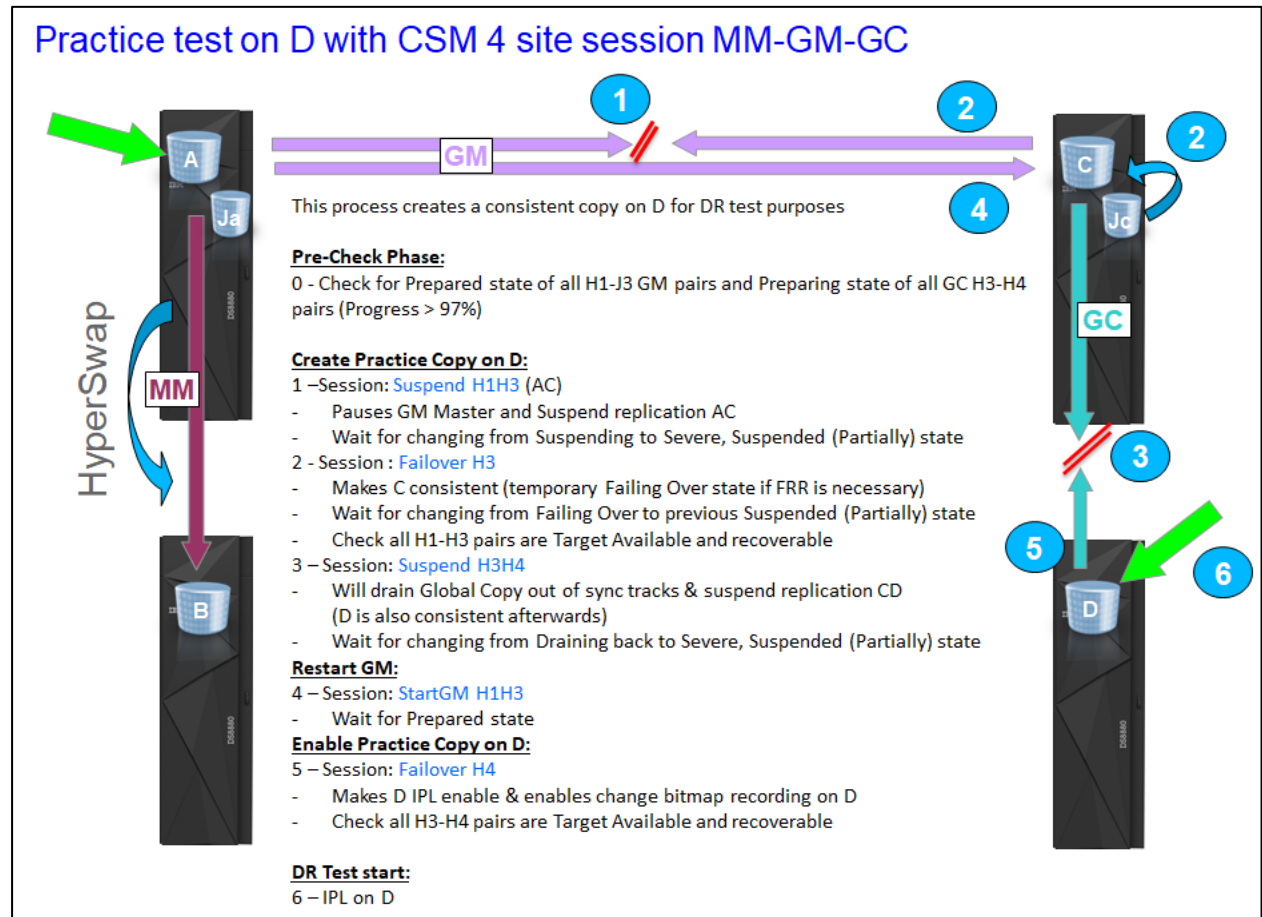
- Procedure to check a given session state, or optionally wait until it is reached
- Procedure to check a given role pair state, progress or recoverability, or optionally wait until any of those parameters is reached
- Procedure to check that all pairs in a given role pair reached a given state, or optionally wait until all reached it
- Procedure to check that a given task name exists and collect its ID, its current status, an optionally wait until the task has completed running
- Wrapper for consistent and centralized formatting of normal, debug or subroutine messages
- Wait procedure, to pause script execution when waiting for a given condition
- Procedure to calculate the run time based on given start time and format it as required
- Wrapper for CSMCLI command execution and output validation

The example script also contains CSMCLI command sequence procedures, which run a fixed CSMCLI command sequence as defined in the procedure, but the whole procedure is executed only if certain conditions are met in the overall automation routine. Chapter '2.5.1 Program Flow overview' gives you an impression, how such command procedures can be structured.

2.5 Script example

In this chapter we describe a concrete script example which utilizes the previous best practices. It is a REXX script which creates a consistent data copy on the fourth site of a 4 site DS8000 replication solution. Starting with CSM 6.2.3 there is a Metro Mirror – Global Mirror with site 4 replication session type which supports this topology.

Following Picture describes the used topology as well as the required procedure to be used to create a consistent practice copy on D volumes.



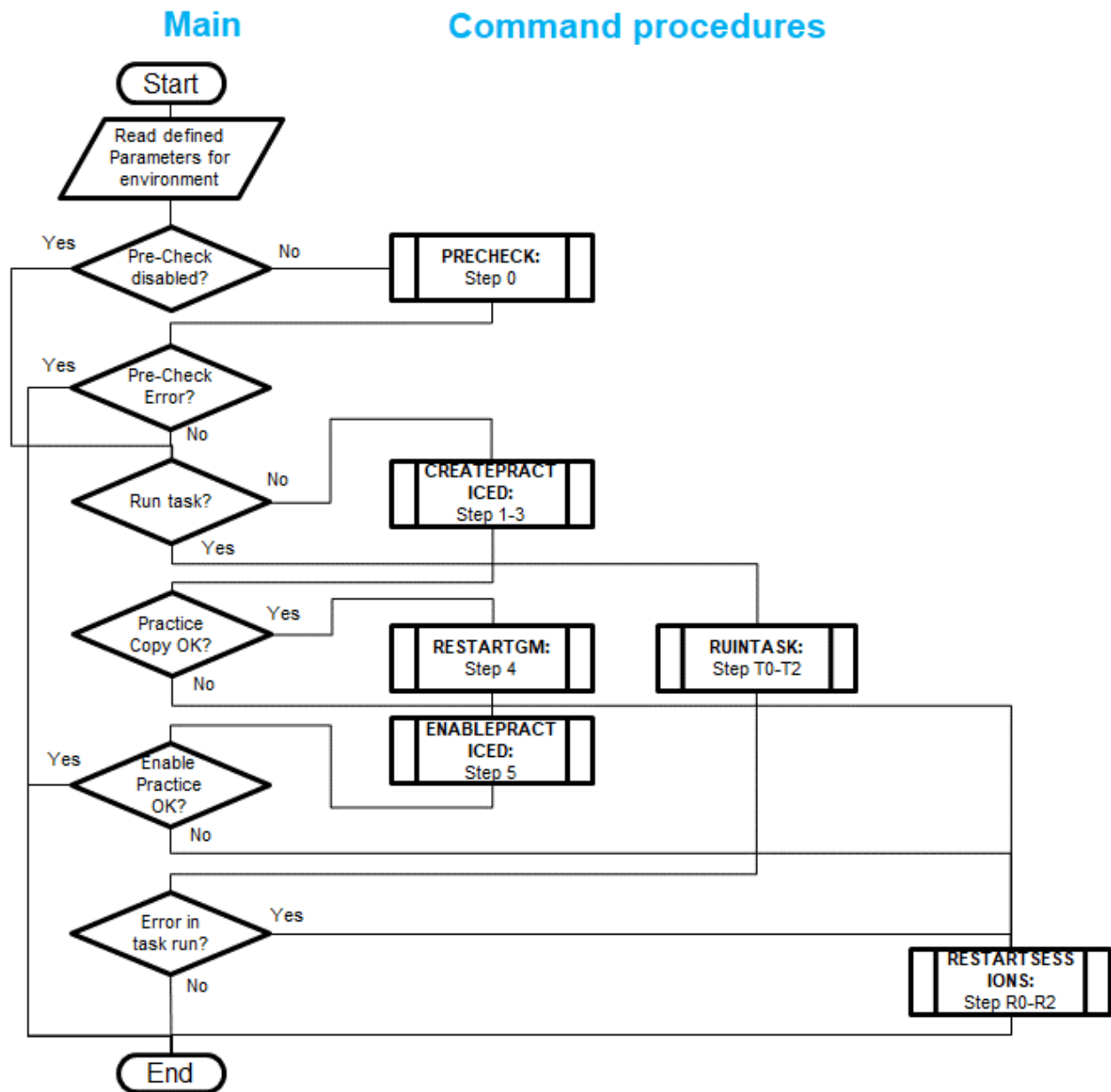
The script example will perform all these steps, but also restart the normal replication mode (4 site replication) in case something goes wrong during the creation of the practice copy.

Optionally, the steps can be defined as a task in the CSM GUI. The script can then be used to perform the optional pre-checks, execute the pre-defined CSM task and monitor its completion. In case of a task error, the script can also restart normal 4 site replication mode.

The full REXX script can be found in '6 Appendix: REXX Script example'. Following sections describe the workflow and the structure of the script.

2.5.1 Program Flow overview

Following picture illustrates how the described sequence is structured and processed in the REXX script.



2.5.2 Script execution

The script will try to create the practice copy when certain pre-check criteria are fulfilled:

- GM Role Pair must be Prepared
- GC Role Pair must be Preparing with a progress of at least xx % (xx is customizable)

If the creation of the practice copy is successful, it will try to restart only GM and enable the Practice Copy on the D volumes (H4). If the GM start fails, it will exit with a Warning RC = 4.

If the creation of the practice copy fails, it will try to restart GM and the cascaded GC of the session to minimize replication impact by the failed practice copy. Prior restart, it will verify the state of the session and might issue a Stop command to either the GM or GC role pair to ensure the session and all pairs are in a state to allow a proper restart. The script will exit with an error RC = 8 if the restart was successful, otherwise with RC = 12 if restart was not successful and replication is still impacted.

2.5.3 Script Return codes

As a summary, the script has following overall return codes:

- 0 : Practice Copy was created and GM is back in Prepared state
- 4 : Practice Copy was created, but GM could not be restarted within customized timeout
- 8 : Practice Copy creation failed, but previous replication was restarted
- 12: Pre-check error or a practice creation error where overall replication restart failed afterwards
- 16: USS syscall environment cannot be established or missing/wrong parameters

2.5.4 Script Runtime environment

The Rexx script can be executed either on a Windows Platform or on z/OS (TSO & Batch). The platform where it is executed needs to have a Rexx interpreter in place and the CSMCLI needs to be installed with an existing authentication properties file for the CSM user. The location of the authentication properties file can be declared in the script with the environment parameter for HOME.

The script was tested on z/OS with embedded Rexx interpreter as well as on Windows with Regina Rexx (<https://regina-rexx.sourceforge.io/>) installed. A different Rexx interpreter on Windows might require adoptions in the script for system specific functions (e.g. reading and setting environment variables for the CSMCLI).

Other platforms are not supported at this time. However, with some slight modifications to the platform determination routine and the conditional execution routines, it can be easily enhanced to run on Linux platforms as well (search for usage of the 'os' variable to get an idea where operating system specific actions are necessary).

2.5.5 Script Parameters

Following parameters can be used as arguments with the script execution:

- acsm=addr**: Hostname or IP address of CSM server having the Active role. This will overwrite the defined 'actcsm' value of the script.
- sess=name**: Name of the 4-site session to be used. This will overwrite the defined 'defsess' value of the script. Name is case sensitive and single/double quotes must be used if it contains spaces. Either session name is required.
- task=name**: Name of the 4-site session scheduled task to be used instead of script steps 1-5 (Optional). Script will then run the task, monitor its completion, and

restore replication in case of task error. Name is case sensitive and single/double quotes must be used if it contains spaces.

pchk=off : This will disable the Pre-Checks of the script (step 0). It can be used if proper pre-checks are included in a given task.

debug=lvl : This will set the debug level of the script. It can be used to increase output details in case of unexpected errors Supported levels are 0 (default), 2 and 9.

The parameter names are not case sensitive, but a given session or task name value is case sensitive. Multiple arguments can be separated with space or tabs.

2.5.6 Script execution via JCL

On z/OS, the Rexx script can be executed via Job Control Language, for example to be scheduled in batch processing. Following JCL example shows how to execute the script:

```
//REXXJCLA JOB (A185,SYS), 'TLUTHER', CLASS=A, MSGCLASS=X,
//          MSGLEVEL=(1,1), REGION=0M, NOTIFY=&SYSUID
//*****
/* Run the REXX script as specified below
//*****
//STEP CNCT EXEC PGM=IKJEFT01
//SYSEXEC DD DISP=SHR, DSN=TLUTHER.CSM.CNTL
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
PROFILE NOPREFIX
%CL3PRACD
    aacsm=myacsm.domain.com
    sess='My MmGmGc'
    task=''
/*
//
```

You need to update the Job card, dataset and member name accordingly (red). The blue Rexx script parameters can be defined in the SYSTSIN DD statement as shown in this example. Make sure to concatenate multiple parameter lines with the script member name line by using the dash '-', otherwise it will not be passed as a single argument string.

2.5.7 Script customization

Following sections in the script should be customized to adopt the script to the local environment. The red parameters need to be reviewed and updated, or specified as arguments when the script is called. The blue parameters can be adjusted if necessary:

```
/* Modify environment for script */
pricsm = "" /* Primary CSM server IP/Name */
seccsm = "" /* Secondary CSM server IP/Name */
actcsm = pricsm /* CSM server with Active role */
stdcsm = seccsm /* CSM server with Standby role */

/* Define def. parameters, extra single quotes are mandatory if space in name*/
defsess = "" /* Name of active MM-GM-GC session */
deftask = "" /* Name of task to run alternatively */

/* Modify scenario parameters as required */
```



```

gcprog      = 97                /* Min. prog % of GC Session at start */
gmsuspto   = 300               /* max sec. for GM Suspend completion */
gcsuspto   = 120               /* max sec. for GC Suspend completion */
frrto      = 120               /* max sec. for FRR completion */
mmrecto    = 60                /* max sec. for MM Recovery completion */
gmstartto  = 300               /* max sec. for GM Restart completion */
gcstartto  = 60                /* max sec. for GC Restart completion */
gcstopto   = 120               /* max sec. for GC Stop (err. recovery)*/
taskto     = 600               /* max sec. for task duration */

/* Mofify environment for CSMCLI calls */
env.0 = 2                      /* # of entries in env. */
env.1 = "HOME=/u/username"     /* Home for auth file */
env.2 = "PATH=/opt/IBM/CSM/CLI/" /* Path to csmcli.sh */
cliex = "csmcli.sh -noinfo"    /* default executable */
dlmch = ";"                    /* Delimiter char for output */
/* Note: This script does not include CSMCLI username or password. It relies */
/* that the CSMCLI authentication properties file is setup in the CSM-CLI */
/* subfolder of the specified HOME folder: */
/* <HOME>/csm-cli/csmcli-auth.properties */

/* Set Debug level for additional output */
debug = 0                      /* Set >0 for more debug output */
/* 2: print more output of procedures */
/* 9: print also CSMCLI call details */

```

2.5.8 Script output control

Output details can be increased for script debugging purposes. Debug output will be marked and may contain more details on what is being processed in the sub procedures or what is sent and returned for the CSMCLI execution. For a standard output example of the script, please refer to '7 Appendix: Output of REXX Script example'.

The example script debug level can be easily adjusted with the debug value:

```

/* Set Debug level for additional output */
debug = 0                      /* Set >0 for more debug output */
/* 2: print more output of procedures */
/* 9: print also CSMCLI call details */

```

In the script itself, the output level is controlled by verifying the value of the debug variable, e.g.:

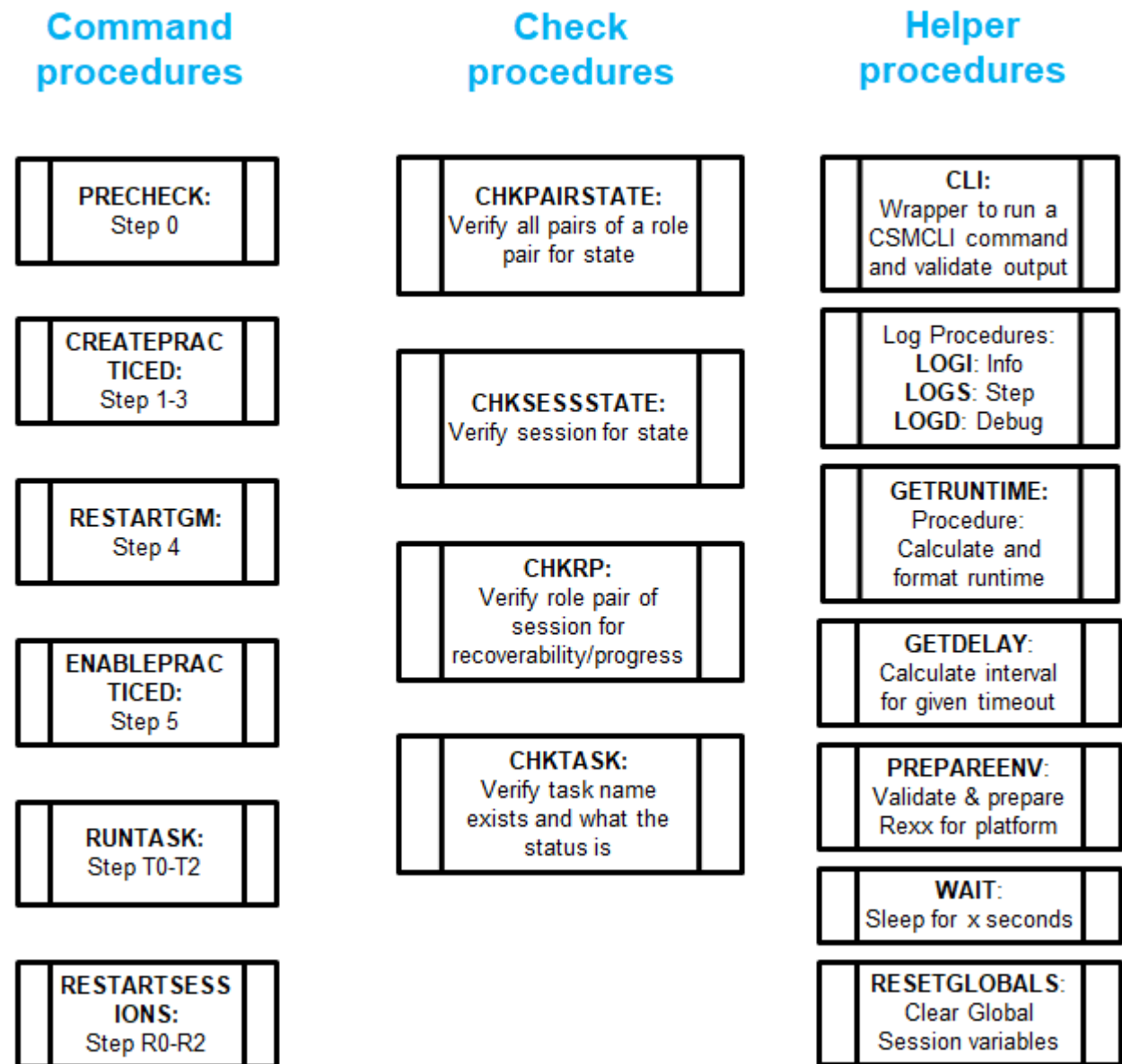
```

parse arg sess1, rp1, state1, to1
if debug >= 2 then say LOGD(2,"Parameters:" sess1","rp1","state1","to1)

```

2.5.9 Procedure overview

Whenever reusable parts are possible, they are provided as procedures in the example script. Following picture illustrates the available script procedures:



These procedures can also be used in a general CSMCLI script framework for automation that goes beyond this practice script. For the detailed script code, please refer to chapter '6 Appendix: REXX Script example'.

2.5.9.1 Command procedures

The script divides the overall sequence into 4 command procedures, which are called by the main routine as documented in '2.5.1 Program Flow overview'. Following is an overview of the 6 command procedures.

```
/*-----*/
/*- SUBROUTINE                                     -*/
/*-                                               -*/
/*- Precheck routine to validate everything is OK for CSMCLI sequence. -*/
/*- It performs Step 0 of the overall sequence.      -*/
/*- Return codes:                                   -*/
/*- 0       : All checks are OK                     -*/
/*- ErrMsg: Message describing the problem          -*/
/*-----*/
```

PRECHECK:

```
/*-----*/
/*- SUBROUTINE                                     -*/
/*-                                               -*/
/*- Sequence to create a practice copy on D volumes -*/
/*- It performs Step 1-3 of the overall sequence.    -*/
/*- Return codes:                                   -*/
/*- 0       : All steps executed successfully       -*/
/*- ErrMsg: Message describing the problem          -*/
/*-----*/
```

CREATEPRACTICED:

```
/*-----*/
/*- SUBROUTINE                                     -*/
/*-                                               -*/
/*- Restart Global Mirror of 4-site session         -*/
/*- It performs Step 4 of the overall sequence.     -*/
/*- Return codes:                                   -*/
/*- 0       : All steps executed successfully       -*/
/*- ErrMsg: Message describing the problem          -*/
/*-----*/
```

RESTARTGM:

```
/*-----*/
/*- SUBROUTINE                                     -*/
/*-                                               -*/
/*- Sequence to enable a practice copy on D volumes (Failover) -*/
/*- It performs Step 5 of the overall sequence.     -*/
/*- Return codes:                                   -*/
/*- 0       : All steps executed successfully       -*/
/*- ErrMsg: Message describing the problem          -*/
/*-----*/
```

ENABLEPRACTICED:

```

/*-----*/
/*- SUBROUTINE                                     -*/
/*-                                               -*/
/*- Check status of given taskname and run the task ID -*/
/*- Monitor task status and wait for task completion. -*/
/*- The task should contain step 1-5 of this script -*/
/*- Return codes:                                 -*/
/*- 0      : Task found and completed successfully -*/
/*- ErrMsg: Message describing the problem       -*/
/*-----*/
RUNTASK:

```

```

/*-----*/
/*- SUBROUTINE                                     -*/
/*-                                               -*/
/*- Restart 4-site session in case there was an error in the sequence -*/
/*- It performs a check whether GC or GM rolepair is in a state that might -*/
/*- require a Stop first (step R0). Then it performs a restart of -*/
/*- cascaded GC (step R1) and a restart of GM (step R2) in 4-site session -*/
/*- Return codes:                                 -*/
/*- 0      : All steps executed successfully     -*/
/*- ErrMsg: Message describing the problem       -*/
/*-----*/
RESTARTSESSIONS:

```

2.5.9.2 Check procedures

The script provides common check procedures which allow to be used with optional timeout parameters in order to validate or wait for a given state or condition.

```
/*-----*/
/*- SUBROUTINE -*/
/*- -*/
/*- Check all pairs in given session & role pair are in the given -*/
/*- state. Optionally specify timeout in sec how long to wait for state. -*/
/*- (It uses CSMCLI lspair -l -rolepair command) -*/
/*- Eg: call CHKPAIRSTATE(session,rolepair,state(,timeout)) -*/
/*- session : String with Session name, use ' ' if it includes spaces -*/
/*- rolepair: String with rolepair to use for pair state check -*/
/*- state : String with state to be validated -*/
/*- timeout : 0-3600 sec (optional, use to wait for given state) -*/
/*- Return codes: -*/
/*- 0 : All checks are OK -*/
/*- ErrMsg: Message describing the problem -*/
/*-----*/
CHKPAIRSTATE:
```

```
/*-----*/
/*- SUBROUTINE -*/
/*- -*/
/*- Check that session reached any of the provided states -*/
/*- Optionally specify timeout in sec how long to wait for valid state. -*/
/*- (It uses CSMCLI lssess -l command) -*/
/*- Eg: call CHKSESSSTATE(session,states(,timeout)) -*/
/*- session: String with Session name, use ' ' if it includes spaces -*/
/*- states : String with comma separated valid states (Use empty string -*/
/*- without timeout to update global variable with state info) -*/
/*- timeout: 0-3600 sec (optional, use to wait for given state) -*/
/*- Return codes: -*/
/*- 0 : All checks are OK -*/
/*- ErrMsg: Message describing the problem -*/
/*-----*/
CHKSESSSTATE:
```

```
/*-----*/
/*- SUBROUTINE -*/
/*- -*/
/*- Check given role pair in session. Optionally check whether recoverable -*/
/*- or whether progress exceeds a given percentage. -*/
/*- Optionally specify timeout in sec how long to wait for required condit. -*/
/*- (It uses CSMCLI lsrolepair -l command) -*/
/*- Eg: call CHKRP(session,rolepair,(recoverable,minprogress)(,timeout)) -*/
/*- session: String with Session name, use ' ' if it includes spaces -*/
/*- rolepair: String with rolepair to use for pair state check -*/
/*- recoverable: (optional) Specify "YES" to validate recoverability -*/
/*- minprogress: 0-100 % (optional, min. Progress in % to be validated) -*/
/*- timeout: 0-3600 sec (optional, use to wait for given state) -*/
/*- Return codes: -*/
/*- 0 : All checks are OK -*/
/*- ErrMsg: Message describing the problem -*/
/*-----*/
CHKRP:
```

```

/*-----*/
/*- SUBROUTINE                                     -*/
/*-                                               -*/
/*- Check if task name exists and what the status is. It will update global -*/
/*- task variables with state and last error message.                             -*/
/*- Optionally specify if check fails if task is active.                           -*/
/*- Optionally specify timeout in sec how long to wait for task completion. -*/
/*- (It uses CSMCLI ltask command)                                                 -*/
/*- Eg: call CHKTASK(taskname(,vfyinactive)(,timeout))                             -*/
/*-   taskname: String with task name, use ' ' if it includes spaces               -*/
/*-   vfyinactive: (optional) Specify "YES" to fail check if active                 -*/
/*-   timeout : 0-3600 sec (optional, use to wait for task completion)             -*/
/*- Return codes:                                                                    -*/
/*- 0      : Task found and completed successfully                                 -*/
/*- ErrMsg: Message describing the problem                                         -*/
/*-----*/
CHKTASK:

```

2.5.9.3 Helper procedures

The script provides common helper procedures for reuse during the execution. The CLI wrapper procedure as well as centralized procedures for output formatting belong to the helper procedures.

```

/*-----*/
/*- SUBROUTINE                                     -*/
/*-                                               -*/
/*- Call CSMCLI with specified cmd and verify RC & output streams.                 -*/
/*- Any CSMCLI framework RC <> 0 will be passed back with more error details -*/
/*- It means the command could not be sent to the server.                         -*/
/*- If the output streams contain a CSMCLI Error message, the full message -*/
/*- line will be returned.                                                         -*/
/*- 0 will be returned if the command was executed without Error message.         -*/
/*- Eg: call CLI(command)                                                           -*/
/*-   command: full single shot csmcli string including executable                 -*/
/*- Return codes:                                                                    -*/
/*- 0      : Command was executed without error                                 -*/
/*- ErrMsg: Message describing the problem                                         -*/
/*-----*/
CLI:

```

```

/*-----*/
/*- SUBROUTINE                                     -*/
/*-                                               -*/
/*- Reset Global Session variables (e.g. prior new CSMCLI queries)                 -*/
/*-----*/
RESETGLOBALS:

```

```

/*-----*/
/*- SUBROUTINE                                     -*/
/*-                                               -*/
/*- Prepare system environment for script execution.                               -*/
/*- It verifies whether the platform is supported by the script and if so         -*/

```

```

/*- it prepares the environment for execution.                -*/
/*- Return codes:                                           -*/
/*- 0      : Preparation completed successfully              -*/
/*- ErrMsg: Message describing the problem                  -*/
/*-----*/
PREPAREENV:

```

```

/*-----*/
/*- SUBROUTINE                                             -*/
/*-                                                     -*/
/*- Create common prefix for messages                     -*/
/*- Eg: LOGI(message)                                     -*/
/*-      message: String to be formatted with prefix     -*/
/*-----*/
LOGI:

```

```

/*-----*/
/*- SUBROUTINE                                             -*/
/*-                                                     -*/
/*- Create common prefix for Step messages               -*/
/*- Eg: LOGS(stepnum,message)                             -*/
/*-      stepnum: Step number to be used in prefix       -*/
/*-      message: String to be formatted with prefix     -*/
/*-----*/
LOGS:

```

```

/*-----*/
/*- SUBROUTINE                                             -*/
/*-                                                     -*/
/*- Create common prefix for debug messages              -*/
/*- Eg: LOGD(dbglvl,message)                              -*/
/*-      dbgnum : Debug level to be used in prefix       -*/
/*-      message: String to be formatted with prefix     -*/
/*-----*/
LOGD:

```

```

/*-----*/
/*- SUBROUTINE                                             -*/
/*-                                                     -*/
/*- Calculate runtime and format to mm:ss.s based on provided start time -*/
/*- Eg: call GETRUNTIME(starttime)                       -*/
/*-      starttime: Start time saved with time('E') to use for calculation -*/
/*-----*/
GETRUNTIME:

```

```

/*-----*/
/*- SUBROUTINE                                             -*/
/*-                                                     -*/
/*- Get delay depending on given timeout. Returns -1 for invalid timeouts. -*/
/*- Eg: call GETDELAY(timeout)                           -*/
/*-      timeout: Overall timeout to calculate appropriate delay (0-3600 sec) -*/
/*-----*/
GETDELAY:

```

```
/*-----*/  
/*- SUBROUTINE -*/  
/*- -*/  
/*- Wait x seconds, utilizing USS system call -*/  
/*- Eg: call WAIT(time) -*/  
/*- time: Number of seconds to wait -*/  
/*-----*/
```

WAIT:

3 Session automation via CSM Scheduled Tasks

CSM 6.2.1 introduced a new feature to create 'Scheduled Tasks' which can be run on demand or at regular schedules. In the original release, only Flash commands could be defined for sessions which support the Flash command. In CSM 6.2.2, the Scheduled Task feature was significantly enhanced to define multiple steps per task and the possible actions have been enhanced to all sessions and any management command that is supported by the session. Additionally, a new action type to wait for a specific state was introduced. With those enhancements, much more complex session scenarios can be defined as a scheduled task. Since CSM 6.2.7, two additional actions types have been introduced.

Following action types can be defined within a task as of CSM 6.2.7:

- **Command** action (CSM 6.2.1 or higher)
This action type will run the selected session command against the selected session. If the command fails, the task execution will stop with an error message.
- **Wait for state** action (CSM 6.2.3 or higher)
This action type will wait with further processing until the selected session reaches the selected state. You can specify a time-out in minutes for the maximum time to wait. If the state is not reached within the time-out, the action will fail and the task execution will stop with an error message.
- **Wait for percent complete** action (CSM 6.2.7 or higher)
This action type will wait with further processing until the selected role pair of the selected session reaches the selected progress percentage. You can specify a time-out in minutes for the maximum time to wait. If the progress is not reached within the time-out, the action will fail and the task execution will stop with an error message.
- **Validate role pair consistency** action (CSM 6.2.7 or higher)
This action type will verify if all pairs in the selected role pair of the selected session are in a recoverable state. If not, the action will fail and the task execution will stop with an error message.

Note: The CSM scheduled task capabilities might be further enhanced in future CSM releases. This chapter describes the advanced task capabilities as available with CSM 6.2.7.

3.1 Scheduled Task introduction

Scheduled tasks in CSM can be run against single or multiple sessions. The tasks can either be scheduled to run on defined intervals, or if the scheduled task is deactivated or has no schedule defined, it can also be run on demand only.

The detailed procedure to create scheduled tasks can be found in the CSM Knowledge Center:

- https://www.ibm.com/support/knowledgecenter/SSESK4_6.2.7/com.ibm.storage.csm.help.doc/csm_t_creating_scheduled_tasks.html

As of CSM 6.2.7, scheduled tasks can only be configured through the Graphical User Interface. However, the CSMCLI allows to list, execute and monitor pre-defined tasks to allow external automation or scheduling software to utilize the CSM tasks as well.

A single task can act on multiple CSM sessions and wait for specific states after each command before executing the next step. The only limitation of scheduled tasks is that we cannot define conditional based execution. As such we cannot define automated actions to be taken if certain steps of the task fail. In case of an error or timeout in a single step, the whole task execution will be aborted.

However, a major advantage of the provided scheduled task capability is the overall execution performance, since it directly integrates into the server with event driven state changes. Unlike external automation scripts, a scheduled task does not consume overhead time for launching the CSMCLI framework for each command execution or querying repetitively in intervals when waiting for specific states. Wait for State or Wait for percent complete actions are completed immediately once the condition is met on the server. Therefore the overall execution time of an advanced action sequence is shorter than the execution time via a CSMCLI automation script.

With the task monitor and control capabilities available in the CSMCLI, you can combine the benefits of either automation capability. For instance, you can pre-define a CSM task containing pre-checks and a fixed command sequence (including validation steps as necessary). This ensures fastest possible execution and validation of the command sequence. In the CSMCLI script which will just execute the pre-defined task, you can monitor completion status and eventually react on errors that might happen in the task. In the script example that is discussed in chapter '2.5.1 Program Flow overview', you get an idea how such an optional task execution can be structured in a customized CSMCLI script.

You can also find additional information about CSM scheduled tasks in Appendix A of the Redpaper: DS8000 4-Site Replication with IBM Copy Services Manager, REDP-5517-00

- <http://www.redbooks.ibm.com/redpieces/pdfs/redp5517.pdf>

3.2 Create Scheduled Task with multiple actions

To create a scheduled task, click **Settings > Scheduled Tasks**. On the Scheduled Tasks panel, click **Create Task...** This will open the Scheduled Task wizard.

Create a Scheduled Task

Task Name:

Description:

Create a PE package if error occurs running the task

In the first task panel, you can define the task name, a more detailed description and optionally enable the creation of a PePackage if the task should fail at any step. Click **Next** to proceed to the schedule panel.

How often do you want the task to run?

Schedule

Hourly

Every (hours):

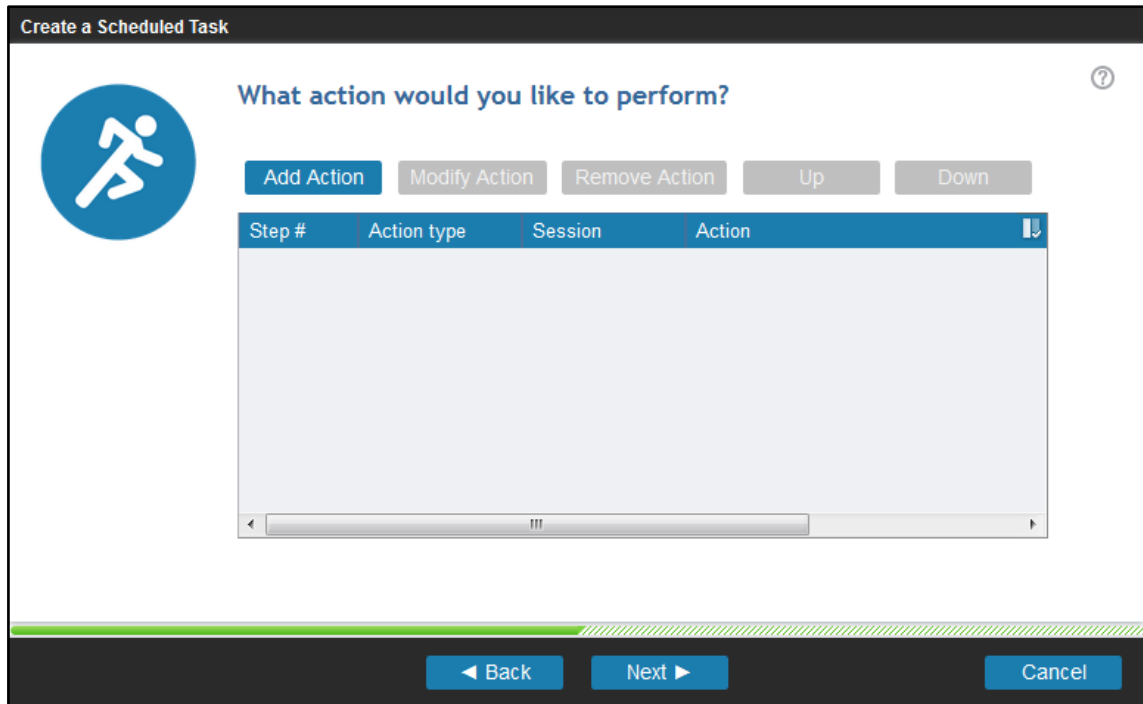
Daily / Weekly

Sun Mon Tue Wed Thu Fri Sat

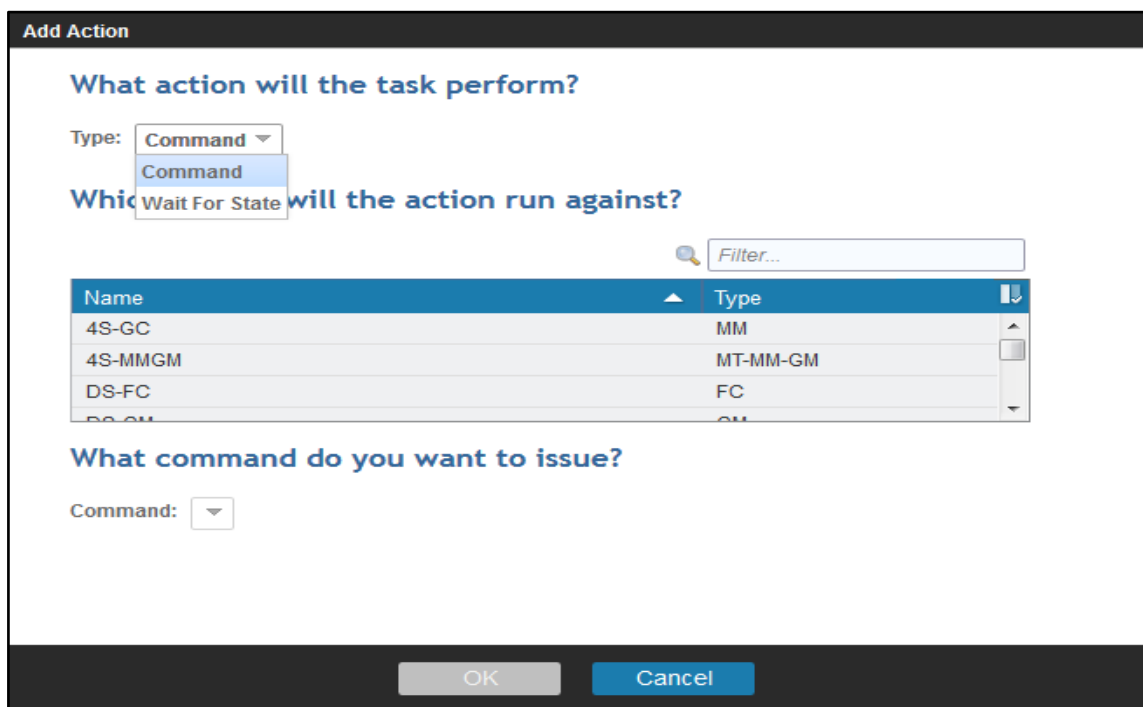
Time [Central European Standard Time]:

No schedule

Specify the required schedule for your task. This can be a regular interval per day, or a defined time at specific weekdays. If your task should not run automatically, select **No schedule** on this panel. Click **Next** to proceed to the Actions panel.



On the Action panel you can add any number of actions for the task. To add a new action, click **Add Action**, which will open the Add Action wizard. Depending on the CSM release, you can select various actions types.



To create a new session *Command* action, select type **Command**, then select the session, and last select the command that you want to run against the selected session. Only commands which are supported by the selected session type will be listed in the command list.

If you need to define multiple Actions to specific sessions only, you can use the Filter option for your session name to shorten the session list. This filter will persist during the whole task creation.

In the following example we want to run a Start H1->H2->H3 against a multi target session with Metro Mirror and Global Mirror.

Add Action

What action will the task perform?

Type: **Command**

Which session will the action run against?

Filter...

Name	Type
4S-GC	MM
4S-MMGM	MT-MM-GM
DS-FC	FC
DS-GM	GM

What command do you want to issue?

Command: **Start H1->H2 H1->H3**

OK Cancel

Click **OK** to create the action and return to the Actions panel.

Create a Scheduled Task

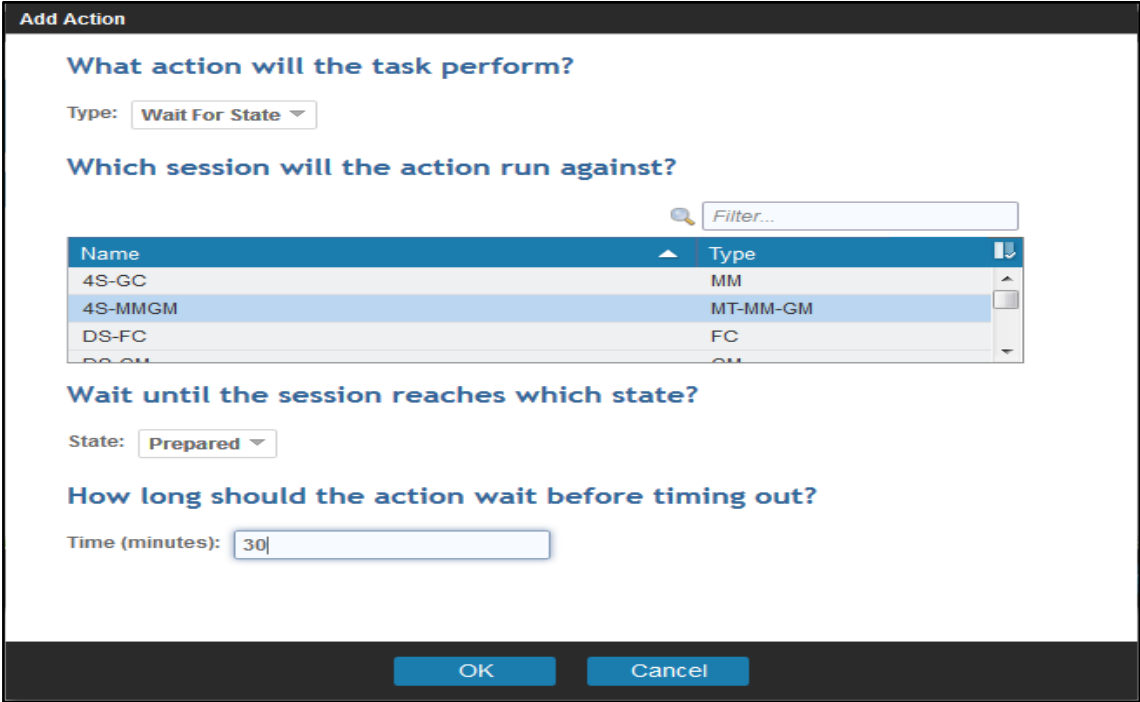
What action would you like to perform?

Add Action Modify Action Remove Action Up Down

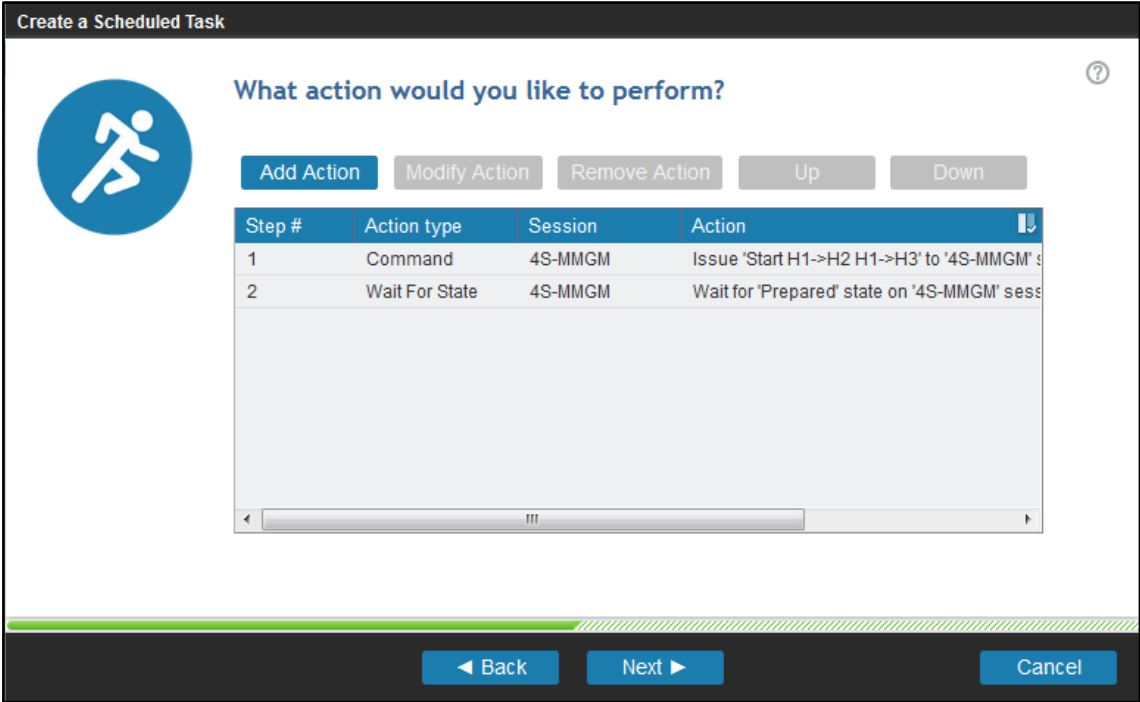
Step #	Action type	Session	Action
1	Command	4S-MMGM	Issue 'Start H1->H2 H1->H3' to '4S-MMGM' s

Back Next Cancel

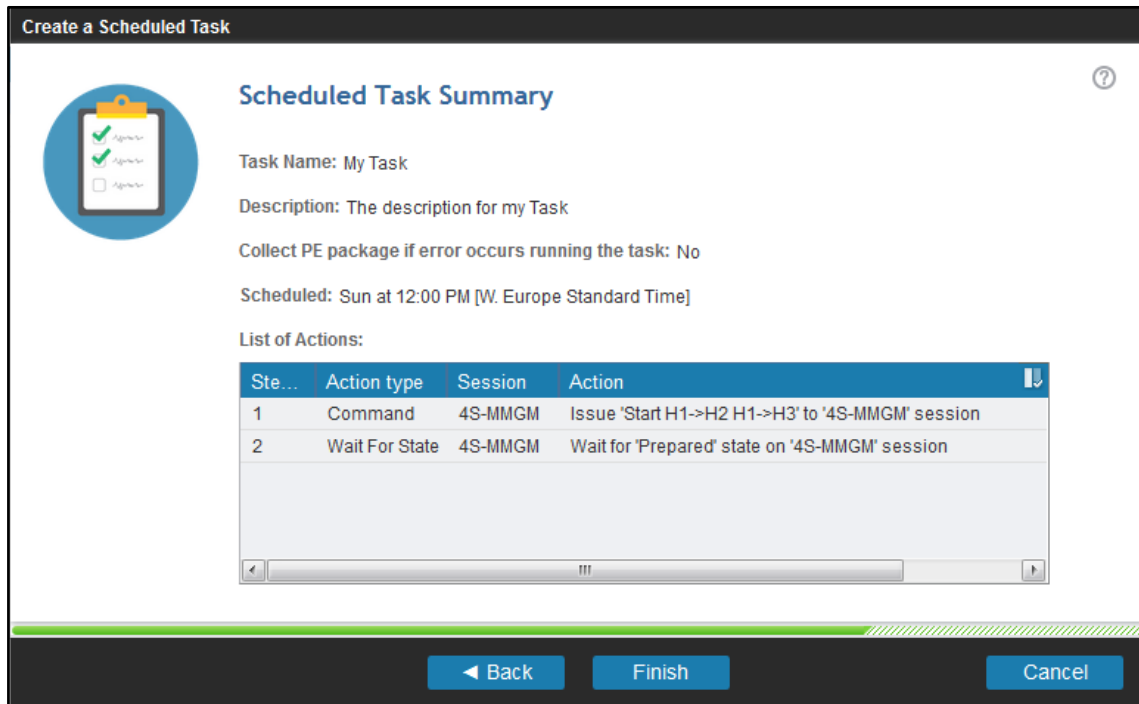
Next we add another action to wait for the session being prepared. Click on **Add Action** to add a new *Wait for State* action for the same session.



In each Wait for State action, you have to specify a timeout in minutes. Since we have a small session which is replicated in a couple of minutes, we select a timeout that is larger than the average replication time, e.g. 30 minutes. Click **OK** to add this second action to the task.

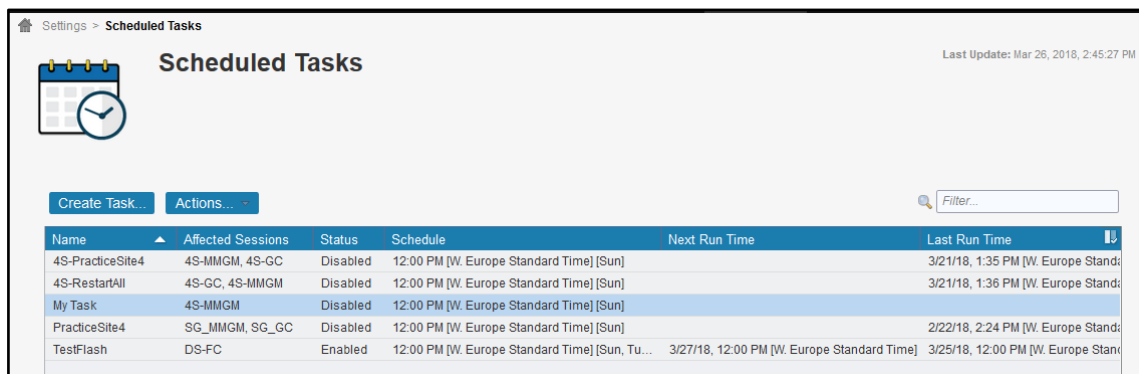


Once all actions for the task are defined, click **Next** to see a summary of the task you are going to create.



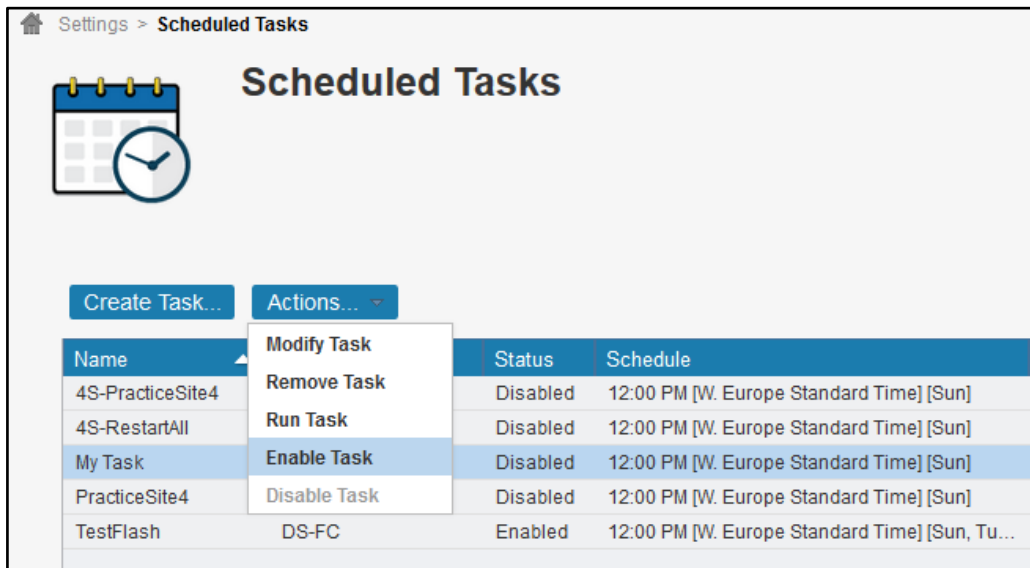
You can always go back to previous panels and modify your input before you complete the task creation.

Click **Finish** to create the task and return to the Scheduled Tasks panel.



Per default, any newly created task shows a status of Disabled. A disabled task means it does not run at the specified schedule if a schedule was defined for the task.

If you want to run a task automatically at the specified schedule, **select the task**, click on **Actions...** and **Enable the task**. From the Action menu you can also run the task on demand by clicking **Run Task**.



If the task has no schedule defined, you cannot enable the task. You can only run the task immediately.

3.3 Modify Scheduled Task with multiple actions

You can easily modify scheduled tasks if you need to change name, description, schedule, change the order of actions or insert/remove any of the actions or modify any of the actions itself. To modify a scheduled task, **select the task** and click **Actions...Modify**.

It will open the Modify Task wizard, which looks similar to the Create task wizard but is already prefilled with the information from the selected task.

Modify a Scheduled Task

Task Name:

Description:

Create a PE package if error occurs running the task

◀ Back Next ▶ Cancel

Simply click through the panels any modify the desired values. Once you reached the Modify Actions panel, perform necessary action changes as required. For instance, in order to change the timeout of our Wait for Prepared State action, select the action and click on **Modify Action**.

What action would you like to perform?

Add Action Modify Action Remove Action Up Down

Step #	Action type	Session	Action
1	Command	4S-MMGM	Issue 'Start H1->H2 H1->H3' to '4S-MMGM' session
2	Wait For State	4S-MMGM	Wait for 'Prepared' state on '4S-MMGM' session

◀ Back Next ▶ Cancel

In the Modify Action panel, you can change any of the action properties. You can even switch it to a Command Action if desired. In this example, we just decrease the timeout from 30 to 20 minutes and click **OK**.

Modify Action

What action will the task perform?

Type:

Which session will the action run against?

Name	Type
4S-GC	MM
4S-MMGM	MT-MM-GM
DS-FC	FC
DS-GM	GM

Wait until the session reaches which state?


State:

How long should the action wait before timing out?

Time (minutes):

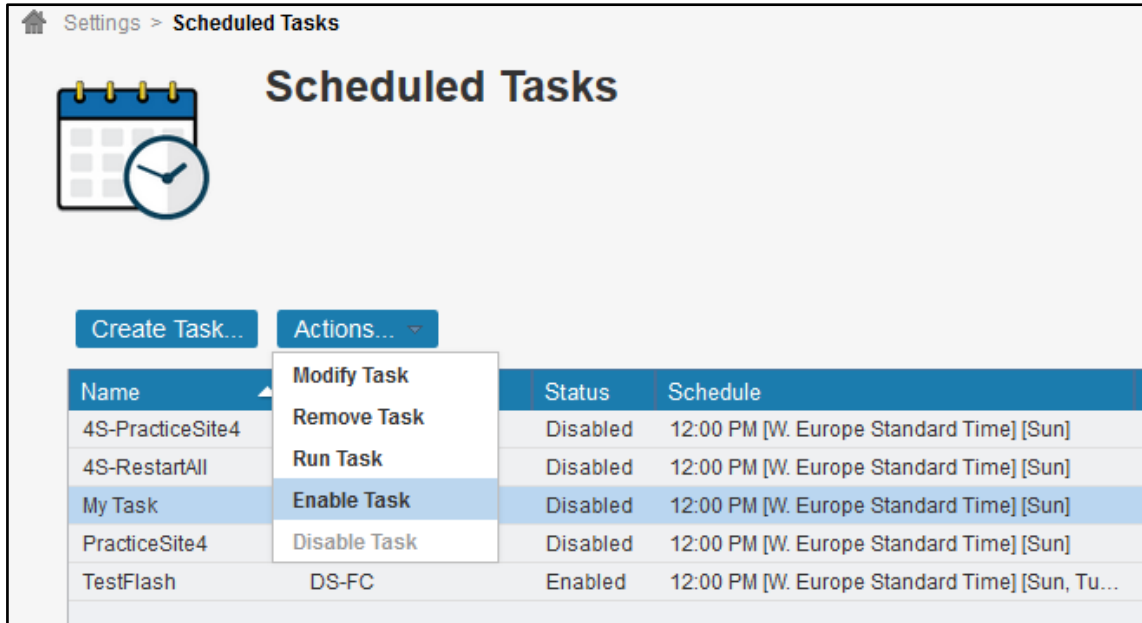
Once you are finished with your modifications, click on **Next** to review the task summary and **Finish** to save your changes. You will get a final confirmation that your changes have been saved.

Modify Task

 IWNR22181
 [Mar 26, 2018 4:37:10 PM] The scheduled task My Task was modified successfully.

3.4 Manage Scheduled Tasks

You can find all defined tasks in the Scheduled Tasks GUI panel. This can be found under **Settings -> Scheduled Tasks**. To manage a task, **select the task** and click on **Actions...**



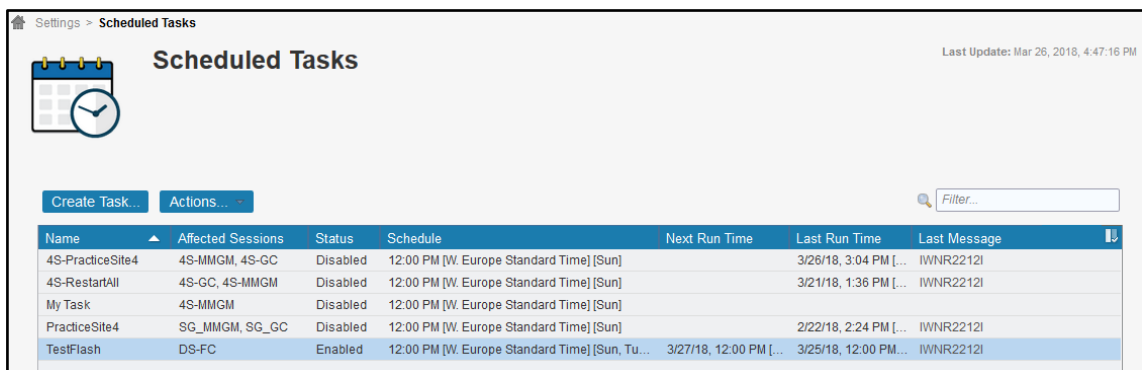
Following Actions are possible for a task:

- **Modify Task** : Make changes to the task, schedule, actions within the task
- **Remove Task** : Delete the task permanently
- **Run Task** : Execute the task on demand
- **Enable Task** : Enable the defined schedule for the task for automatic execution
- **Disable Task** : Disable the defined schedule for the task (No automatic execution)

Starting with CSM 6.2.7, when you enable a task with an hourly schedule, you can also select if the task is enabled Now, or define a day and time when the regular task should be enabled in the future.

3.4.1 Scheduled execution

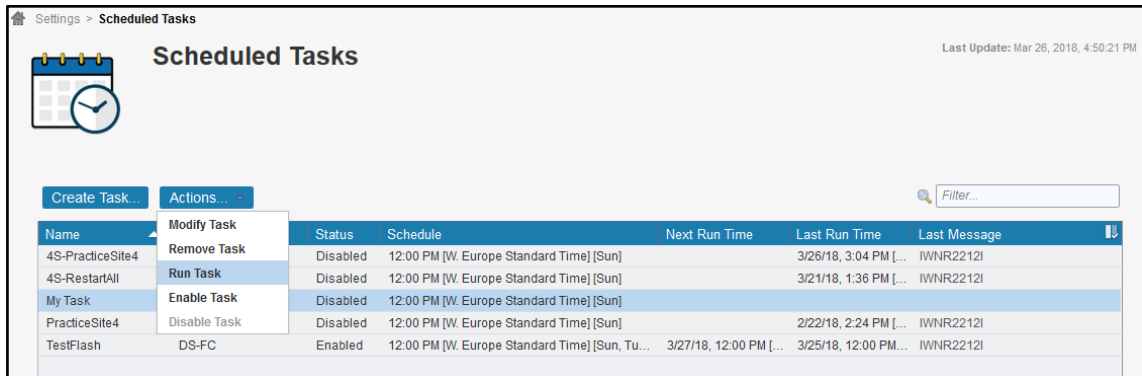
In order to determine whether there are tasks that will be executed automatically, you can look for an *Enabled* status in the table. Enabled tasks will also show a timestamp in the *Next Run Time* column.



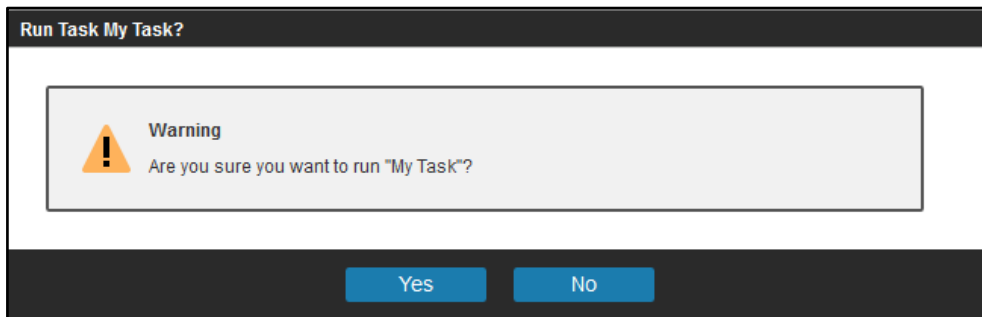
The table also shows when the scheduled tasks have been run the last time and what the last message was. Click on the message link to get more details of the message.

3.4.2 On demand execution

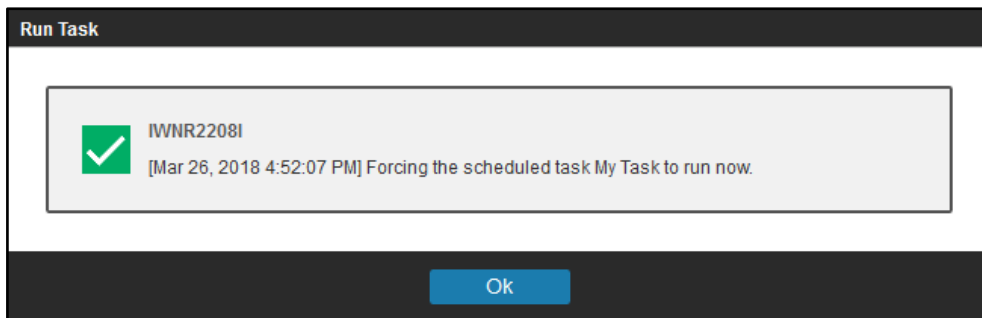
In order to run any task on demand, **select the task**, click **Actions...Run Task**.



You need to confirm that you really want to run the selected task, click **Yes**.



Next you will see a confirmation that the task was started.



Click **OK** to close the information popup and return to the Scheduled Tasks panel.

3.4.3 Task monitoring

Running tasks can be identified in the Scheduled Tasks GUI panel. The status column shows a **Running** status while a task is being executed.

Settings > Scheduled Tasks

Scheduled Tasks Last Update: Mar 26, 2018, 4:55:15 PM

Create Task... Actions...

Filter...

Name	Affected Sessions	Status	Schedule	Next Run Time	Last Run Time	Last Message
4S-PracticeSite4	4S-MMGM, 4S-GC	Disabled	12:00 PM [W. Europe Standard Time] [Sun]		3/26/18, 3:04 PM [...]	IWNR2212I
4S-RestartAll	4S-GC, 4S-MMGM	Disabled	12:00 PM [W. Europe Standard Time] [Sun]		3/21/18, 1:36 PM [...]	IWNR2212I
My Task	4S-MMGM	Running	12:00 PM [W. Europe Standard Time] [Sun]		3/26/18, 4:54 PM [...]	IWNR1026I
PracticeSite4	SG_MMGM, SG_GC	Disabled	12:00 PM [W. Europe Standard Time] [Sun]		2/22/18, 2:24 PM [...]	IWNR2212I
TestFlash	DS-FC	Enabled	12:00 PM [W. Europe Standard Time] [Sun, Tu...]	3/27/18, 12:00 PM [...]	3/25/18, 12:00 PM [...]	IWNR2212I

While the task is running, you can follow the detailed progress in the CSM Console log. Click on **Console** to open the console log panel. In the upper right corner, click on the small icon with the arrow in the window to open the console log in a new window and keep it open while continuing navigation in the CSM GUI.

Console 26, 2018, 5:00:21

- ✓ Mar 26, 2018 4:54:59 PM : Server : IWNR2211I : The scheduled task My Task has started running.
- ✓ Mar 26, 2018 4:54:59 PM : Server : IWNR1028I : The Start H1->H2 H1->H3 command in the 4S-MMGM session was issued.
- ✓ Mar 26, 2018 4:54:59 PM : Server : IWNR6016I : Configuring paths for role pair H2-H3...
- ✓ Mar 26, 2018 4:55:10 PM : Server : IWNR6016I : Configuring paths for role pair H2-H3...
- ✓ Mar 26, 2018 4:55:10 PM : Server : IWNR6004I : Recovering all pairs in role pair H2-H3 ...
- ✓ Mar 26, 2018 4:55:10 PM : Server : IWNR6000I : Starting all pairs in role pair H1-H2 ...
- ✓ Mar 26, 2018 4:55:10 PM : Server : IWNR6006I : Waiting for all pairs in role pair H1-H2 to reach a state of Prepared. See help for list of actions if transition is taking too long...
- ✓ Mar 26, 2018 4:55:10 PM : Server : IWNR6000I : Starting all pairs in role pair H1-H3 ...
- ✓ Mar 26, 2018 4:55:10 PM : Server : IWNR6008I : Waiting for all pairs in role pair H1-H3 to complete their first phase in the Global Copy synchronization or resynchronization ...
- ✓ Mar 26, 2018 4:55:10 PM : Server : IWNR1950I : Session 4S-MMGM changed from the Suspended state to the Preparing state.
- ✓ Mar 26, 2018 4:55:10 PM : Server : IWNR1960I : Session 4S-MMGM has changed from Severe status to Warning status.
- ✓ Mar 26, 2018 4:55:10 PM : Server : IWNR1026I : The Start H1->H2 H1->H3 command in the 4S-MMGM session completed.
- ✓ Mar 26, 2018 4:55:11 PM : Server : IWNR6000I : Starting all pairs in role pair H1-J3 ...
- ✓ Mar 26, 2018 4:55:19 PM : Server : IWNR1041I : The command start was successfully issued to all pairs under role pair H1-J3 for session 4S-MMGM.
- ✓ Mar 26, 2018 4:55:22 PM : Server : IWNR1950I : Session 4S-MMGM changed from the Preparing state to the Prepared state.
- ✓ Mar 26, 2018 4:55:22 PM : Server : IWNR1960I : Session 4S-MMGM has changed from Warning status to Normal status.
- ✓ Mar 26, 2018 4:55:23 PM : Server : IWNR2220I : The scheduled task My Task in step 2 found session 4S-MMGM moved to state Prepared in 0 minutes.
- ✓ Mar 26, 2018 4:55:23 PM : Server : IWNR2212I : The scheduled task My Task has finished running.

Open in new window

Close

There are separate messages that mark the start and the completion of a task:

```
normal Mar 26, 2018 4:54:59 PM : Server : IWNR2211I : The scheduled task My Task has started running.
.....
normal Mar 26, 2018 4:55:23 PM : Server : IWNR2212I : The scheduled task My Task has finished running.
```

When you look for task messages, you should be aware that CSM currently does not log specific step messages for Command Actions. You will see just the standard command output the same way as running the command in the session itself. However, you will see that the command was issued by the Server instead of a CSM user:

```
normal Mar 26, 2018 4:54:59 PM : Server : IWNR1028I : The Start H1->H2 H1->H3 command in the 4S-MMGM session was issued.
```

The Wait for State actions however log individual step messages once completed:

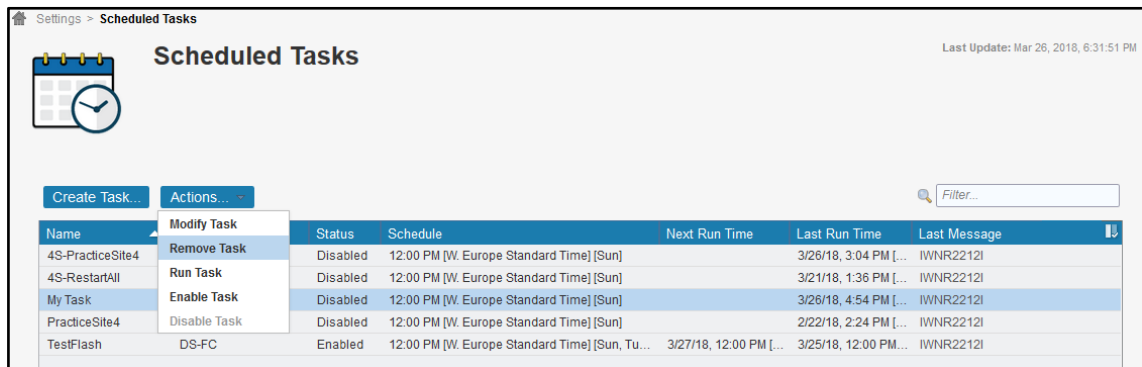
```
normal Mar 26, 2018 4:55:23 PM : Server : IWNR2220I : The scheduled task My Task in step 2 found session 4S-MMGM moved to state Prepared in 0 minutes.
```

Following is the complete console log for the execution of the example task we created previously. The task was executed while the session 4S-MMGM was in state Suspended with no changes to the primary volumes. Therefore it moved to Prepared state very quickly. Each task step entry is marked in blue:

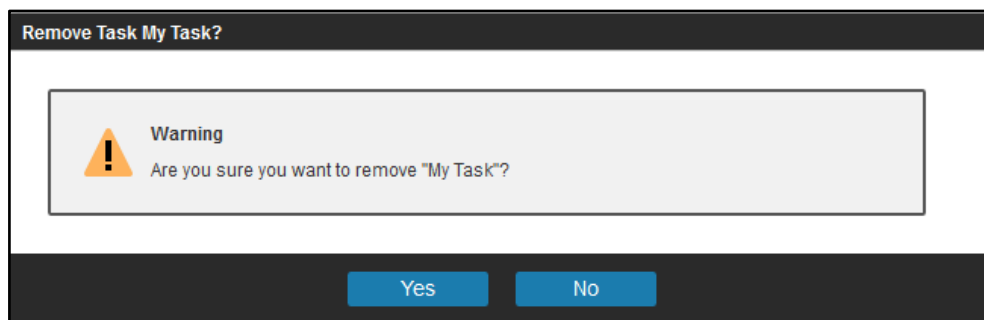
```
normal Mar 26, 2018 4:54:59 PM : csmadmin : IWNR22081 : Forcing the scheduled task My Task to run now.
normal Mar 26, 2018 4:54:59 PM : Server : IWNR22111 : The scheduled task My Task has started running.
normal Mar 26, 2018 4:54:59 PM : Server : IWNR10281 : The Start H1->H2 H1->H3 command in the 4S-MMGM session was issued.
normal Mar 26, 2018 4:54:59 PM : Server : IWNR60161 : Configuring paths for role pair H2-H3...
normal Mar 26, 2018 4:55:10 PM : Server : IWNR60161 : Configuring paths for role pair H2-H3...
normal Mar 26, 2018 4:55:10 PM : Server : IWNR60041 : Recovering all pairs in role pair H2-H3 ...
normal Mar 26, 2018 4:55:10 PM : Server : IWNR60001 : Starting all pairs in role pair H1-H2 ...
normal Mar 26, 2018 4:55:10 PM : Server : IWNR60061 : Waiting for all pairs in role pair H1-H2 to reach a state of Prepared. See help for
list of actions if transition is taking too long...
normal Mar 26, 2018 4:55:10 PM : Server : IWNR60001 : Starting all pairs in role pair H1-H3 ...
normal Mar 26, 2018 4:55:10 PM : Server : IWNR60081 : Waiting for all pairs in role pair H1-H3 to complete their first phase in the Global
Copy synchronization or resynchronization ...
normal Mar 26, 2018 4:55:10 PM : Server : IWNR19501 : Session 4S-MMGM changed from the Suspended state to the Preparing state.
normal Mar 26, 2018 4:55:10 PM : Server : IWNR19601 : Session 4S-MMGM has changed from Severe status to Warning status.
normal Mar 26, 2018 4:55:10 PM : Server : IWNR10261 : The Start H1->H2 H1->H3 command in the 4S-MMGM session completed.
normal Mar 26, 2018 4:55:11 PM : Server : IWNR60001 : Starting all pairs in role pair H1-J3 ...
normal Mar 26, 2018 4:55:19 PM : Server : IWNR10411 : The command start was successfully issued to all pairs under role pair H1-J3
for session 4S-MMGM.
normal Mar 26, 2018 4:55:22 PM : Server : IWNR19501 : Session 4S-MMGM changed from the Preparing state to the Prepared state.
normal Mar 26, 2018 4:55:22 PM : Server : IWNR19601 : Session 4S-MMGM has changed from Warning status to Normal status.
normal Mar 26, 2018 4:55:23 PM : Server : IWNR22201 : The scheduled task My Task in step 2 found session 4S-MMGM moved to state
Prepared in 0 minutes.
normal Mar 26, 2018 4:55:23 PM : Server : IWNR22121 : The scheduled task My Task has finished running.
```

3.5 Remove Scheduled Tasks

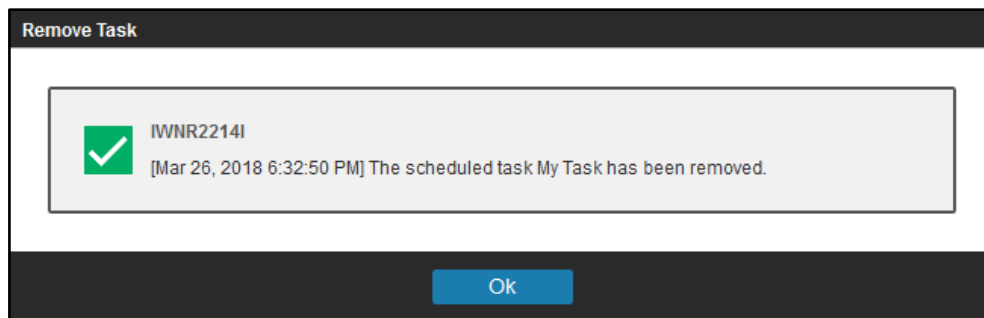
A task can also be permanently deleted in the Scheduled Tasks GUI panel. To delete a task, **select the task**, click on **Actions...Remove Task**.



You need to confirm that you want to remove the selected Task. Click **Yes** to proceed.



Next you will see a confirmation that the task was removed.



Click **OK** to close the information popup and return to the Scheduled Tasks panel. The deleted task disappeared from the Scheduled Tasks table.

3.6 Advanced Scheduled Task example

Similar to the automation script example that is described in this paper, we can automate advanced command sequences via a scheduled task. The described task in this example will demonstrate how a consistent practice copy on 4th site can be created in a 4 site replication topology. It is the same sequence of a CSM MM-GM with site 4 replication session as described in '2.5 Script example'.

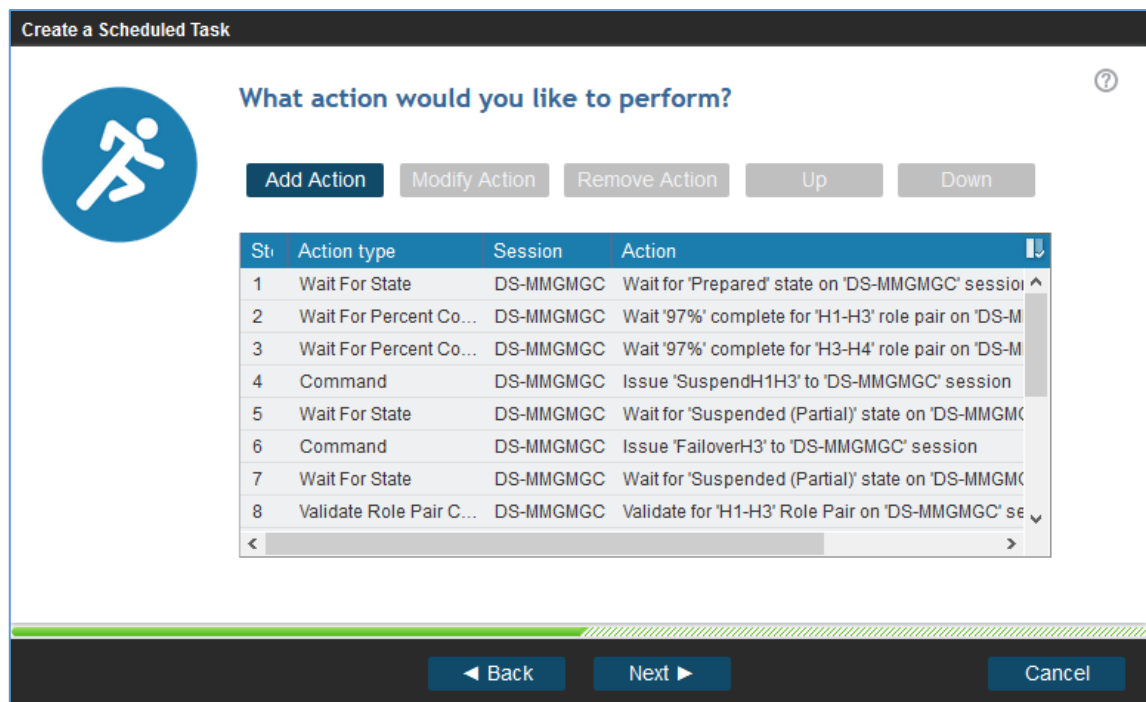
3.6.1 Task and Actions definition

Following table shows the required actions which need to be defined for such a task. The 4 action steps marked in blue (2,3,8 and 15) are new action types and require CSM 6.2.7 or higher. The blue timeout and progress values are parameters that should be adjusted to your requirements.

Note: The timeout of 0 minutes for the first 3 actions should remain unchanged for initial validation steps in the sequence to ensure that the session commands will be triggered only when the session is in a specific condition.

Step	Action	Command/State/Parms	Timeout	Comment
1	Wait for State	Prepared	0 min	
2	Wait for percent complete	97% of H1-H3	0 min	Ensure GM has sufficient progress to limit GM RPO impact Requires CSM 6.2.7
3	Wait for percent complete	97% of H3-H4	0 min	Ensure GC has sufficient progress to limit GM RPO impact Requires CSM 6.2.7
4	Command	SuspendH1H3		will transition to <i>Suspending</i>
5	Wait for State	Suspended (Partial)	3 min	
6	Command	FailoverH3		will transition to <i>Failing Over</i>
7	Wait for State	Suspended (Partial)	3 min	
8	Validate role pair consistency	H1-H3		Ensure H3 is consistent Requires CSM 6.2.7
9	Command	SuspendH3-H4	3 min	will transition to <i>Draining</i>
10	Wait for State	Suspended (Partial)	3 min	
11	Command	StartGMH1-H3		Start GM first to limit RPO impact, will transition to <i>Preparing</i>
12	Wait for State	Prepared	5 min	
13	Command	FailoverH4		will transition to <i>Failing Over</i>
14	Wait for State	Prepared	1 min	
15	Validate role pair consistency	H3-H4		Ensure H4 is consistent Requires CSM 6.2.7

Following is the task overview in the GUI:



3.6.2 Task execution

When the task is run, you can follow the progress in the CSM Console log. Following is the output produced by this task. Note that the command actions do not log step specific messages to the console, but just the standard output as running the command in the session itself. The Wait for State actions log individual step messages. Each step entry is marked in blue:

```
normal Jan 16, 2020 1:53:04 PM : csmadmin : IWNR2208I : Forcing the scheduled task MMGMGC-PracticeD to run now.
normal Jan 16, 2020 1:53:04 PM : Server : IWNR2211I : The scheduled task MMGMGC-PracticeD has started running.
normal Jan 16, 2020 1:53:04 PM : Server : IWNR2220I : The scheduled task MMGMGC-PracticeD in step 1 found session DS-
MMGMGC moved to state Prepared in 0 minutes.
normal Jan 16, 2020 1:53:04 PM : Server : IWNR2224I : The scheduled task MMGMGC-PracticeD in step 2 found session DS-
MMGMGC and role pair H1-H3 moved to 97 percent complete in 0 minutes.
normal Jan 16, 2020 1:53:04 PM : Server : IWNR2224I : The scheduled task MMGMGC-PracticeD in step 3 found session DS-
MMGMGC and role pair H3-H4 moved to 97 percent complete in 0 minutes.
normal Jan 16, 2020 1:53:04 PM : Server : IWNR1028I : The SuspendH1H3 command in the DS-MMGMGC session was issued.
normal Jan 16, 2020 1:53:04 PM : Server : IWNR6005I : Pausing all pairs in role pair H1-J3 ...
normal Jan 16, 2020 1:53:04 PM : Server : IWNR6006I : Waiting for all pairs in role pair H1-J3 to reach a state of Suspended. See help
for list of actions if transition is taking too long...
normal Jan 16, 2020 1:53:04 PM : Server : IWNR1950I : Session DS-MMGMGC changed from the Prepared state to the Suspending
state.
warning Jan 16, 2020 1:53:04 PM : Server : IWNR1959W : Session DS-MMGMGC has changed from Normal status to Warning status.
normal Jan 16, 2020 1:53:04 PM : Server : IWNR1026I : The SuspendH1H3 command in the DS-MMGMGC session completed.
normal Jan 16, 2020 1:53:05 PM : Server : IWNR1950I : Session DS-MMGMGC changed from the Suspending state to the Suspended
(Partial) state.
warning Jan 16, 2020 1:53:05 PM : Server : IWNR1959W : Session DS-MMGMGC has changed from Warning status to Severe status.
normal Jan 16, 2020 1:53:05 PM : Server : IWNR2220I : The scheduled task MMGMGC-PracticeD in step 5 found session DS-
MMGMGC moved to state Suspended (Partial) in 0 minutes.
normal Jan 16, 2020 1:53:05 PM : Server : IWNR1028I : The FailoverH3 command in the DS-MMGMGC session was issued.
normal Jan 16, 2020 1:53:05 PM : Server : IWNR6004I : Recovering all pairs in role pair H1-J3 ...
normal Jan 16, 2020 1:53:07 PM : Server : IWNR1950I : Session DS-MMGMGC changed from the Suspended (Partial) state to the
Failing Over state.
normal Jan 16, 2020 1:53:07 PM : Server : IWNR1960I : Session DS-MMGMGC has changed from Severe status to Normal status.
normal Jan 16, 2020 1:53:07 PM : Server : IWNR6012I : Flashing all pairs in role pair H3-J3 ...
normal Jan 16, 2020 1:53:07 PM : Server : IWNR1041I : The command flash was successfully issued to all pairs under role pair H3-J3
for session DS-MMGMGC.
```

normal Jan 16, 2020 1:53:07 PM : Server : IWNR1950I : Session DS-MMGMGC changed from the Failing Over state to the Suspended (Partial) state.

warning Jan 16, 2020 1:53:07 PM : Server : IWNR1959W : Session DS-MMGMGC has changed from Normal status to Severe status.

normal Jan 16, 2020 1:53:07 PM : Server : IWNR1026I : The FailoverH3 command in the DS-MMGMGC session completed.

normal Jan 16, 2020 1:53:07 PM : Server : IWNR2220I : The scheduled task MMGMGC-PracticeD in step 7 found session DS-MMGMGC moved to state Suspended (Partial) in 0 minutes.

normal Jan 16, 2020 1:53:07 PM : Server : IWNR2227I : The scheduled task MMGMGC-PracticeD in step 8 found role pair H1-H3 in session DS-MMGMGC was recoverable.

normal Jan 16, 2020 1:53:07 PM : Server : IWNR1028I : The SuspendH3H4 command in the DS-MMGMGC session was issued.

normal Jan 16, 2020 1:53:07 PM : Server : IWNR6015I : Waiting until all pairs in role pair H3-H4 have copied 100% of their data.

normal Jan 16, 2020 1:53:07 PM : csmadmin : IWNR1950I : Session DS-MMGMGC changed from the Suspended (Partial) state to the Draining state.

normal Jan 16, 2020 1:53:07 PM : csmadmin : IWNR1960I : Session DS-MMGMGC has changed from Severe status to Warning status.

normal Jan 16, 2020 1:53:07 PM : Server : IWNR1026I : The SuspendH3H4 command in the DS-MMGMGC session completed.

normal Jan 16, 2020 1:53:07 PM : Server : IWNR6002I : Suspending all pairs in role pair H3-H4 ...

normal Jan 16, 2020 1:53:08 PM : Server : IWNR1041I : The command freeze was successfully issued to all pairs under role pair H3-H4 for session DS-MMGMGC.

normal Jan 16, 2020 1:53:08 PM : Server : IWNR6020I : Releasing I/O for all pairs in role pair H3-H4...

normal Jan 16, 2020 1:53:08 PM : Server : IWNR1041I : The command thaw was successfully issued to all pairs under role pair H3-H4 for session DS-MMGMGC.

normal Jan 16, 2020 1:53:08 PM : Server : IWNR6019I : Verifying the consistency of role pair H3-H4...

normal Jan 16, 2020 1:53:08 PM : Server : IWNR1041I : The command checkConsistency was successfully issued to all pairs under role pair H3-H4 for session DS-MMGMGC.

normal Jan 16, 2020 1:53:08 PM : csmadmin : IWNR1950I : Session DS-MMGMGC changed from the Draining state to the Suspended (Partial) state.

warning Jan 16, 2020 1:53:08 PM : csmadmin : IWNR1959W : Session DS-MMGMGC has changed from Warning status to Severe status.

normal Jan 16, 2020 1:53:09 PM : Server : IWNR2220I : The scheduled task MMGMGC-PracticeD in step 10 found session DS-MMGMGC moved to state Suspended (Partial) in 0 minutes.

normal Jan 16, 2020 1:53:09 PM : Server : IWNR1028I : The StartGM H1->H3 command in the DS-MMGMGC session was issued.

normal Jan 16, 2020 1:53:09 PM : Server : IWNR6003I : Terminating all pairs in role pair H1-J1 ...

normal Jan 16, 2020 1:53:09 PM : Server : IWNR6004I : Recovering all pairs in role pair H2-H3 ...

normal Jan 16, 2020 1:53:09 PM : Server : IWNR6004I : Recovering all pairs in role pair H2-J3 ...

normal Jan 16, 2020 1:53:10 PM : Server : IWNR6000I : Starting all pairs in role pair H1-H3 ...

normal Jan 16, 2020 1:53:12 PM : csmadmin : IWNR1950I : Session DS-MMGMGC changed from the Suspended (Partial) state to the Preparing state.

warning Jan 16, 2020 1:54:08 PM : Server : IWNR6030W : The pairs in role pair H3-H4 are not replicating the latest data from role pair H1-H3.

normal Jan 16, 2020 1:54:08 PM : Server : IWNR1960I : Session DS-MMGMGC has changed from Severe status to Warning status.

normal Jan 16, 2020 1:54:08 PM : Server : IWNR6008I : Waiting for all pairs in role pair H1-H3 to complete their first phase in the Global Copy synchronization or resynchronization ...

normal Jan 16, 2020 1:54:08 PM : Server : IWNR1026I : The StartGM H1->H3 command in the DS-MMGMGC session completed.

normal Jan 16, 2020 1:54:08 PM : Server : IWNR6000I : Starting all pairs in role pair H1-J3 ...

normal Jan 16, 2020 1:55:16 PM : Server : IWNR1041I : The command start was successfully issued to all pairs under role pair H1-J3 for session DS-MMGMGC.

normal Jan 16, 2020 1:55:20 PM : csmadmin : IWNR1950I : Session DS-MMGMGC changed from the Preparing state to the Prepared state.

normal Jan 16, 2020 1:55:21 PM : Server : IWNR2220I : The scheduled task MMGMGC-PracticeD in step 12 found session DS-MMGMGC moved to state Prepared in 1 minutes.

normal Jan 16, 2020 1:55:21 PM : Server : IWNR1028I : The FailoverH4 command in the DS-MMGMGC session was issued.

normal Jan 16, 2020 1:55:21 PM : Server : IWNR6004I : Recovering all pairs in role pair H3-H4 ...

warning Jan 16, 2020 1:55:21 PM : Server : IWNR6030W : The pairs in role pair H3-H4 are not replicating the latest data from role pair H1-H3.

normal Jan 16, 2020 1:55:21 PM : Server : IWNR1026I : The FailoverH4 command in the DS-MMGMGC session completed.

normal Jan 16, 2020 1:55:21 PM : Server : IWNR2220I : The scheduled task MMGMGC-PracticeD in step 14 found session DS-MMGMGC moved to state Prepared in 0 minutes.

normal Jan 16, 2020 1:55:21 PM : Server : IWNR2227I : The scheduled task MMGMGC-PracticeD in step 15 found role pair H3-H4 in session DS-MMGMGC was recoverable.

normal Jan 16, 2020 1:55:21 PM : Server : IWNR2212I : The scheduled task MMGMGC-PracticeD has finished running.

4 CSM session automation conclusion

We discussed various options how CSM session automation can be implemented. Some are easy to implement and good enough to perform simple automation tasks. Other options like CSMCLI scripting are more complex to implement but provide most flexibility to accomplish any complex session or even CSM server task.

If runtime performance plays a significant role for your automation, you should evaluate whether the necessary actions can be implemented as a CSM Scheduled Task (requires CSM 6.2.1 or later). The CSM task actions are controlled by the CSM server itself and wait states are event driven which avoids unnecessary wait times. If we compare the overall runtimes from the CSM task described in '3.6.2 Task execution' and the Rexx script runtime show in '7 Appendix: Output of REXX Script example' for the same session configuration, we can see following:

- CSM task run time : ~ 137 seconds
- Rexx script commands run time : ~ 186 seconds

The additional script run time is the overhead we have for launching the CSMCLI framework to execute the commands and the additional delays we accumulate in the intervals while waiting for specific states.

On the other hand, the Rexx script can handle conditional based execution and as such automatically limit the impact that might be caused if the creation of the practice copy should fail at any step.

An optimized approach might be to automate CSM task execution via Rexx script in order to use best runtime performance for a CSM session command sequence, but still being able to handle recovery steps via CSMCLI commands or other CSM tasks in case the core task execution will fail.

At the end, your automation requirements will determine the most suitable implementation option for your unique environment and hopefully this paper will give you the necessary guidelines and examples to implement your CSM session automation easily.

5 References

5.1 CSM and DS8000 Copy Services

1. IBM Copy Services Manager Knowledge Center
https://www.ibm.com/support/knowledgecenter/SSESK4/csm_kcwelcome.html
2. Redbook: IBM Copy Services Manager Implementation Guide, SG24-8375-00
<http://www.redbooks.ibm.com/abstracts/sg248375.html?Open>
3. Redpaper: DS8000 4-Site Replication with IBM Copy Services Manager, REDP-5517-00
<http://www.redbooks.ibm.com/redpieces/abstracts/redp5517.html?Open>
4. Redbook: IBM DS8000 Copy Services, SG24-8367-00
<http://www.redbooks.ibm.com/abstracts/sg248367.html?Open>
5. Redpaper: IBM DS8880 Integrated Copy Services Manager and LDAP Client on the HMC, REDP-5356-00
<http://www.redbooks.ibm.com/abstracts/redp5356.html?Open>

5.2 Rexx Scripting

1. z/OS 2.3 TSO/E REXX User's Guide
https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.ikjc300/abstract.htm
2. Regina Rexx, portable Rexx Interpreter
<https://regina-rexx.sourceforge.io/>
3. Brexx, lightweight open source implementation of Rexx
<https://sourceforge.net/projects/brexx/>
4. Open Object Rexx
<http://www.oorexx.org/about.html>
5. Free Resources for Rexx Programmers
<http://www.rexxinfo.org/>

6 Appendix: REXX Script example

This example script contains the various CSMCLI scripting concepts as discussed in this paper. It can be reused as framework for your own scripting. The included procedures have been coded in a way to be commonly reusable for other scenarios as well.

```
0001 /* REXX */
0002
0003 /*****
0004 /* IBM Copy Services Manager script to create practice copy on 4th site: */
0005 /* ----- */
0006 /* It uses the CSMCLI to create a consistent practice copy on 4th site of a */
0007 /* four site replication topology with a MM-GM-GC 4-site session. */
0008 /* */
0009 /* These process steps are used: */
0010 /* ----- */
0011 /* 0) Check for Prepared state of H1-J3 pairs in session and Preparing */
0012 /* State of H3-H4 pairs in cascaded GC role pair with progress >= xx %) */
0013 /* 1) Suspend H1-H3 (GM leg) of session and wait until suspended */
0014 /* 2) FailoverH3 of session and wait until completed */
0015 /* Wait for previous Suspended (Partially) state and check H1-H3 */
0016 /* is recoverable and all H1-H3 pairs are Target Available */
0017 /* 3) Suspend cascaded GC of H3-H4 and wait until Suspended */
0018 /* 4) StartGM H1-H3 (GM leg) of session to minimize GM RPO impact */
0019 /* Wait for Prepared state of all H1-H3 pairs */
0020 /* 5) Failover cascaded GC of H3-H4 and wait until Target Available */
0021 /* Check that H3-H4 is recoverable */
0022 /* If there are errors in step 1-5 or in an optional task that is executed, */
0023 /* the script will try to restore original GM and cascaded GC replication. */
0024 /* */
0025 /* Optional Input Parameters: */
0026 /* ----- */
0027 /* acsm=addr: Hostname or IP address of CSM server having the Active role. */
0028 /* This will overwrite the defined 'actcsm' value of the script. */
0029 /* sess=name: Name of the 4-site session to be used. */
0030 /* This will overwrite the defined 'deffsess' value of the script. */
0031 /* Name is case sensitive and single/double quotes must be used */
0032 /* if it contains spaces. Either session name is required. */
0033 /* task=name: Name of the 4-site session scheduled task to be used instead */
0034 /* of script steps 1-5 (Optional). Script will then run the task, */
0035 /* monitor its completion, and restore replication in case of task */
0036 /* error. Name is case sensitive and single/double quotes must be */
0037 /* used if it contains spaces. */
0038 /* pchk=off : This will disable the Pre-Checks of the script (step 0). It */
0039 /* can be used if proper pre-checks are included in a given task. */
0040 /* debug=lvl : This will set the debug level of the script. It can be used to */
0041 /* increase output details in case of unexpected errors. */
0042 /* Supported levels are 0 (default), 2 and 9 */
0043 /* */
0044 /* The script has following overall return codes: */
0045 /* ----- */
0046 /* 0 :Practice Copy was created and GM is back in Prepared state */
0047 /* 4 :Practice Copy was created, but GM could not be restarted within timeout*/
0048 /* 8 :Practice Copy creation failed, but previous replication was restarted */
0049 /* 12:Precheck error or practice copy as well as replication restart failed */
0050 /* 16:System environment for script cannot be established or missing parms */
0051 /* */
0052 /* Copyright IBM 2019, Author: Thomas Luther */
0053 /*****
0054
0055 /* Modify environment for script */
0056 pricsm = "" /* Primary CSM server IP/Name */
0057 seccsm = "" /* Secondary CSM server IP/Name */
```

```

0058 actcsm = pricsm                /* CSM server with Active role */
0059 stdcsm = seccsm                /* CSM server with Standby role */
0060
0061 /* Define def. parameters, extra single quotes are mandatory if space in name*/
0062 defsess = "'"                  /* Name of active MM-GM-GC session */
0063 deftask = "'"                  /* Name of task to run alternatively */
0064
0065 /* Modify scenario parameters as required */
0066 gcprog = 97                    /* Min. prog % of GC rolepair at start */
0067 gmsuspto = 300                 /* max sec. for GM Suspend completion */
0068 gcsuspto = 120                 /* max sec. for GC Suspend completion */
0069 frerto = 120                   /* max sec. for FRR completion */
0070 mmrecto = 60                   /* max sec. for MM Recovery completion */
0071 gmstartto = 300                /* max sec. for GM Restart completion */
0072 gcstartto = 60                 /* max sec. for GC Restart completion */
0073 gcstopto = 120                 /* max sec. for GC Stop (err. recovery)*/
0074 taskto = 600                  /* max sec. for task duration */
0075
0076 /* Mofify environment for CSMCLI calls */
0077 env.0 = 2                       /* # of entries in env. */
0078 env.1 = "HOME=/u/username"      /* Home for auth file */
0079 env.2 = "PATH=/opt/IBM/CSM/CLI/" /* Path to csmcli.sh */
0080 cliex = "csmcli.sh -noinfo"     /* default executable */
0081 dlmch = ";"                     /* Delimiter char for output */
0082 /* Note: This script does not include CSMCLI username or password. It relies */
0083 /* that the CSMCLI authentication properties file is setup in the CSM-CLI */
0084 /* subfolder of the specified HOME folder: */
0085 /* <HOME>/csm-cli/csmcli-auth.properties */
0086
0087 /* Set Debug level for additional output */
0088 debug = 0                       /* Set >0 for more debug output */
0089                                     /* 2: print more output of procedures */
0090                                     /* 9: print also CSMCLI call details */
0091
0092
0093 /* Define output formatting */
0094 tab = " "                        /* Tab to structure sub output */
0095 line = left("-",79,"-")
0096
0097 /* Define global Session variables to keep latest info from last query */
0098 GSname = "" /* Session Name */
0099 GSstate= "" /* Session State */
0100 GSrp = "" /* Specific Role Pair if applicable */
0101 GShost = "" /* Session active Host */
0102 GSrecov= "" /* Recoverability of Session/Rolepair */
0103 GSprog = "" /* Progress of Rolepair */
0104 GScpset= "" /* Qty of Copy Sets in Session */
0105
0106 /* Get Session parameters */
0107 parse arg parms
0108
0109 /* extract parms and assign values*/
0110 s4sess = ""
0111 s4task = ""
0112 s4pch = ""
0113 prest = parms
0114 do while prest <> ""
0115     parse var prest parm '=' prest
0116     if prest <> "" then do
0117         /* extract word before = */
0118         parm = translate(word(parm,words(parm)))
0119         qchar = left(word(prest,1),1)
0120         if qchar = "'" | qchar = '"' then do
0121             /* lookup matching quote char for full value*/
0122             parse var prest (qchar) pval (qchar) prest

```

```

0123     end
0124     else do
0125         pval = word(prest,1)
0126     end
0127     if parm = "SESS" & pval <> "" then do
0128         s4sess = pval
0129     end
0130     else if parm = "TASK" & pval <> "" then do
0131         s4task = pval
0132     end
0133     else if parm = "PCHK" & translate(pval) = "OFF" then do
0134         s4pchk = "DISABLED"
0135     end
0136     else if parm = "ACSM" & pval <> "" then do
0137         actcsm = pval
0138     end
0139     else if parm = "DEBUG" & pval <> "" then do
0140         if datatype(pval,'W') then do
0141             if pval > 0 then debug = pval
0142         end
0143     end
0144 end
0145 end
0146
0147 /* Set defaults if no parameter defined */
0148 if s4pchk = "" then s4pchk = "ENABLED" /* Precheck enabled per default */
0149 if s4sess = "" then s4sess = strip(defsess,"B","")
0150 if s4task = "" then s4task = strip(deftask,"B","")
0151 s4sess = ""s4sess""
0152 s4task = ""s4task""
0153 cliex = cliex "-server" actcsm /* append active server to exec */
0154 totrc = 0
0155
0156 /* Get Operating System and print used parameters */
0157 parse upper source osfull .
0158
0159 runtime = time('E')
0160 say line
0161 say LOGI("Creating Practice Copy on Site 4 (D volumes):")
0162 say LOGI("4-site Session:" s4sess)
0163 say LOGI("Session task   :" s4task)
0164 say LOGI("Pre-Checks    :" s4pchk)
0165 say LOGI("CSM Server     :" actcsm)
0166 say LOGI("Local O/S     :" osfull)
0167 say LOGI("Debug Level   :" debug)
0168 say line
0169 say
0170
0171 /* Verify required parameters for execution are given */
0172 if s4sess = "" then do
0173     say LOGI("ERROR: Missing 4-site Session name.")
0174     totrc = 16
0175 end
0176 if actcsm = "" then do
0177     say LOGI("ERROR: Missing CSM server IP/hostname.")
0178     totrc = 16
0179 end
0180 if totrc <> 0 then do
0181     say LOGI("Total RC      =" totrc)
0182     say line
0183     return totrc
0184 end
0185
0186 /* Verify and Prepare System environment */
0187 totrc = PREPAREENV()

```

```

0188 if totrc <> 0 then do
0189     say LOGI("ERROR: Failed to prepare system environment for:" osfull)
0190     say LOGI(totrc)
0191     totrc = 16
0192     say LOGI("Total RC      =" totrc)
0193     say line
0194     return totrc
0195 end
0196
0197
0198 /*****
0199 /* Start the sequence */
0200 /*****
0201
0202 /* Run precheck first if enabled */
0203 if s4pchk = "DISABLED" then do
0204     totrc = 0
0205 end
0206 else do
0207     totrc = PRECHECK()
0208 end
0209
0210 /* If precheck failed, exit */
0211 if totrc <> 0 then do
0212     say LOGI("Precheck ERROR:" totrc)
0213     totrc = 12
0214     say LOGI("Total RC      =" totrc)
0215     say line
0216     return totrc
0217 end
0218
0219
0220 /* Create Practice copy on D */
0221 say line
0222 say
0223 /* Run given task if specified, otherwise use script */
0224 if s4task <> "" then do
0225     totrc = RUNTASK()
0226     if totrc <> 0 & GTid = "" then do
0227         /* Error but Task was not started yet, exit without repl restart */
0228         say LOGI("Task start ERROR:" totrc)
0229         totrc = 12
0230         say LOGI("Total RC      =" totrc)
0231         say line
0232         return totrc
0233     end
0234 end
0235 else do
0236     totrc = CREATEPRACTICED()
0237 end
0238 if totrc <> 0 then say LOGI("ERROR:" totrc)
0239 say line
0240 say
0241
0242 /* Restart replication */
0243 enabrc = totrc
0244 if totrc = 0 & s4task = "" then do
0245     say line
0246     /* Practice data OK, restart only GM */
0247     say LOGI("Successfully created practice Copy on Site 4 (D volumes)")
0248     say LOGI("Restarting Global Mirror to restore site protection...")
0249     say line
0250     say
0251     totrc = RESTARTGM()
0252     if totrc <> 0 then say LOGI("ERROR:" totrc)

```



```

0253 say line
0254 say
0255 say line
0256 if totrc <> 0 then do
0257     say LOGI("WARNING: Failed to restart Global Mirror.")
0258     totrc = 4
0259 end
0260 else do
0261     say LOGI("Global Mirror returned to Prepared State.")
0262 end
0263 /* Enable practice copy on H4 */
0264 say LOGI("Enabling practice Copy on Site 4 (D volumes)...")
0265 say line
0266 say
0267 enabrc = ENABLEPRACTICED()
0268 if enabrc <> 0 then do
0269     say LOGI("ERROR:" enabrc)
0270     say LOGI("Failed to enable practice Copy on Site 4.")
0271 end
0272 say line
0273 say
0274 end
0275
0276 /* Recover replication states if error occurred */
0277 say line
0278 if enabrc = 0 then do
0279     if s4task <> "" then do
0280         say LOGI("Successfully completd task to create practice Copy on Site 4" ,
0281             "(D volumes)")
0282     end
0283     else do
0284         say LOGI("Successfully enabled practice Copy on Site 4.")
0285     end
0286     say LOGI("Systems can be IPLed on Site 4 for testing.")
0287 end
0288 else do
0289     /* Script or task failed, try to restart GM and GC */
0290     if s4task <> "" then do
0291         say LOGI("Failed to complete task for practice Copy on Site 4 (D volumes)")
0292     end
0293     else do
0294         say LOGI("Failed to enable a practice Copy on Site 4 (D volumes)")
0295     end
0296     say LOGI("Restarting Global Mirror and cascaded Global Copy to restore")
0297     say LOGI("original states of session...")
0298     say line
0299     say
0300     totrc = RESTARTSESSIONS()
0301     say line
0302     if totrc <> 0 then do
0303         say LOGI("Practice Copy on D volumes failed, and failed to restart")
0304         say LOGI("original replication. Please verify state of Session" s4sess)
0305         say LOGI("and restart replication manually.")
0306         totrc = 12
0307     end
0308     else do
0309         say LOGI("Practice Copy on D volumes failed, but original replication")
0310         say LOGI("in 4-site session" s4sess "was restarted successfully.")
0311         totrc = 8
0312     end
0313 end
0314
0315 say line
0316 say
0317 say line

```

```

0318 say LOGI("Total Runtime:" GETRUNTIME(runtime))
0319 say LOGI("Total RC      =" totrc)
0320 say line
0321 return totrc
0322
0323
0324 /*-----*/
0325 /*- SUBROUTINE                                     -*/
0326 /*-                                               -*/
0327 /*- Precheck routine to validate everything is OK for CSMCLI sequence. -*/
0328 /*- It performs Step 0 of the overall sequence.         -*/
0329 /*- Return codes:                                       -*/
0330 /*- 0           : All checks are OK                     -*/
0331 /*- ErrMsg: Message describing the problem              -*/
0332 /*-----*/
0333 PRECHECK:
0334 /* Step 0: Check MT session H1-J3 prepared, GC session H1-H2 is preparing */
0335 /* Step 0: Check GC session progress least min defined progress for start */
0336 step = 0
0337 steptime = time('E')
0338 sess = s4sess
0339 state = "Prepared"
0340 rolep = "H1-J3"
0341 say LOGS(step,"Check for" state "state of" rolep "pairs in session" sess"..")
0342 cmdrc = CHKPAIRSTATE(sess,rolep,state)
0343 if cmdrc <> 0 then do
0344     say LOGS(step,"Error while checking" state "state of" rolep "pairs in" sess)
0345     return cmdrc
0346 end
0347 say LOGS(step,"All pairs of" rolep "in" sess "are" state)
0348 state = "Preparing"
0349 rolep = "H3-H4"
0350 minprog = gcprog
0351 say LOGS(step,"Check for" state "state of" rolep "pairs in session" sess"..")
0352 cmdrc = CHKPAIRSTATE(sess,rolep,state)
0353 if cmdrc <> 0 then do
0354     say LOGS(step,"Error while checking" state "state of" rolep "pairs in" sess)
0355     return cmdrc
0356 end
0357 say LOGS(step,"Check" rolep "Progress >=" minprog "% in session" sess"..")
0358 cmdrc = CHKRP(sess,rolep,,minprog)
0359 if cmdrc <> 0 then do
0360     say LOGS(step,"Error while checking" rolep "progress in" sess)
0361     return cmdrc
0362 end
0363 say LOGS(step,"Progress of" rolep "in" sess "is >=" minprog "%")
0364 say LOGS(step,"Pre-Check completed successfully")
0365 say LOGS(step,"Runtime" GETRUNTIME(steptime))
0366 return 0
0367
0368 /*-----*/
0369 /*- SUBROUTINE                                     -*/
0370 /*-                                               -*/
0371 /*- Sequence to create a practice copy on D volumes -*/
0372 /*- It performs Step 1-3 of the overall sequence.     -*/
0373 /*- Return codes:                                       -*/
0374 /*- 0           : All steps executed successfully     -*/
0375 /*- ErrMsg: Message describing the problem              -*/
0376 /*-----*/
0377 CREATEPRACTICED:
0378 /* Step 1: Suspend GM leg of MT session and wait for completion */
0379 step = 1
0380 steptime = time('E')
0381 sess = s4sess
0382 command = "SuspendH1H3"

```

```

0383 say LOGS(step,command "of 4-site session" sess"..")
0384 cmd = cliex "cmdsess -quiet -action" command sess
0385 cmdrc = CLI(cmd)
0386 if cmdrc <> 0 then do
0387     say LOGS(step,"Error during" command "of 4-site session" sess)
0388     return cmdrc
0389 end
0390 say LOGS(step,"Successfully issued" command "to 4-site session" sess)
0391 validstates = "Suspended, Suspended (Partial)"
0392 timeout = gmsuspto /* max wait time for GM suspend completion */
0393 say LOGS(step,"Waiting for suspend completion of 4-site session" sess ,
0394     "(max." timeout "s)...")
0395 cmdrc = CHKSESSSTATE(sess,validstates,timeout)
0396 if cmdrc <> 0 then do
0397     say LOGS(step,"Error while waiting for session" sess "to reach valid state")
0398     return cmdrc
0399 end
0400 say LOGS(step,"Successfully suspended H1-H3 of 4-site session" sess)
0401 say LOGS(step,"Runtime" GETRUNTIME(step,time))
0402 say line
0403 say
0404
0405 /* Step 2: Failover GM leg of MT session and wait until H3 is consistent */
0406 step = 2
0407 steptime = time('E')
0408 sess = s4sess
0409 /* Get last session state from Globals from previos suspend check */
0410 oldstate = "???"
0411 say LOGS(step,"Determine state of 4-site session" sess "from previous query")
0412 if debug >= 2 then say LOGD(2,"GSname="GSname", GSrp="GSrp", GSstate="GSstate")
0413 if GSname = sess & GSrp = "" & GSstate <> "" then oldstate = GSstate
0414 say LOGS(step,sess "session state is:" oldstate)
0415 if oldstate = "???" then do
0416     say LOGS(step,"Could not determine last state from previous query of " sess)
0417     return 8
0418 end
0419
0420 command = "FailoverH3"
0421 say LOGS(step,command "of 4-site session" sess"..")
0422 cmd = cliex "cmdsess -quiet -action" command sess
0423 cmdrc = CLI(cmd)
0424 if cmdrc <> 0 then do
0425     say LOGS(step,"Error during" command "of 4-site session" sess)
0426     return cmdrc
0427 end
0428 say LOGS(step,"Successfully issued" command "to 4-site session" sess)
0429 validstates = oldstate /* wait until previous state reached */
0430 timeout = frtto /* max wait time for FRR completion */
0431 say LOGS(step,"Waiting for failover completion of 4-site session" sess ,
0432     "(max." timeout "s)...")
0433 cmdrc = CHKSESSSTATE(sess,validstates,timeout)
0434 if cmdrc <> 0 then do
0435     say LOGS(step,"Error while waiting for session" sess ,
0436         "to become:" validstates)
0437     return cmdrc
0438 end
0439
0440 state = "Target Available"
0441 rolep = "H1-H3"
0442 say LOGS(step,"Check" state "state of" rolep "pairs in" sess"..")
0443 cmdrc = CHKPAIRSTATE(sess,rolep,state)
0444 if cmdrc <> 0 then do
0445     say LOGS(step,"Error while checking" state "state of" rolep "in" sess)
0446     return cmdrc
0447 end

```

```

0448 say LOGS(step,"All pairs of" rolep "in" sess "are" state)
0449
0450 rolep = "H1-H3"
0451 say LOGS(step,"Check" rolep "is recoverable in session" sess"...")
0452 cmdrc = CHKRP(sess,rolep,"YES")
0453 if cmdrc <> 0 then do
0454     say LOGS(step,"Error while checking" rolep ,
0455             "is recoverable in session" sess)
0456     return cmdrc
0457 end
0458 say LOGS(step,rolep "in" sess "is recoverable")
0459 say LOGS(step,"H3 volumes are consistent in 4-site session" sess)
0460 say LOGS(step,"Runtime" GETRUNTIME(stepTime))
0461 say line
0462 say
0463
0464 /* Step 3: Suspend cascaded GC rolepair and wait for completion */
0465 step = 3
0466 steptime = time('E')
0467 sess = s4sess
0468 command = "SuspendH3H4"
0469 say LOGS(step,command "of 4-site session" sess"...")
0470 cmd = cliex "cmdsess -quiet -action" command sess
0471 cmdrc = CLI(cmd)
0472 if cmdrc <> 0 then do
0473     say LOGS(step,"Error during" command "of 4-site session" sess)
0474     return cmdrc
0475 end
0476 say LOGS(step,"Successfully issued" command "to 4-site session" sess)
0477 validstates = "Suspended, Suspended (Partial)"
0478 timeout = gcsuspto /* max wait time for GC suspend completion */
0479 say LOGS(step,"Waiting for suspend completion of 4-site session" sess ,
0480         "(max." timeout "s)...")
0481 cmdrc = CHKSESSSTATE(sess,validstates,timeout)
0482 if cmdrc <> 0 then do
0483     say LOGS(step,"Error while waiting for session" sess ,
0484             "to become:" validstates)
0485     return cmdrc
0486 end
0487 say LOGS(step,"Successfully suspended GC of 4-site session" sess)
0488 say LOGS(step,"Runtime" GETRUNTIME(stepTime))
0489 return 0
0490
0491 /*-----*/
0492 /*- SUBROUTINE -*/
0493 /*- -*/
0494 /*- Restart Global Mirror of 4-site session -*/
0495 /*- It performs Step 4 of the overall sequence. -*/
0496 /*- Return codes: -*/
0497 /*- 0 : All steps executed successfully -*/
0498 /*- ErrMsg: Message describing the problem -*/
0499 /*-----*/
0500 RESTARTGM:
0501 /* Step 4: StartGMH1-H3 of 4-site session and wait until H1-J3 prepared */
0502 step = 4
0503 steptime = time('E')
0504 sess = s4sess
0505 command = "StartGM_H1:H3"
0506 say LOGS(step,command "of 4-site session" sess"...")
0507 cmd = cliex "cmdsess -quiet -action" command sess
0508 cmdrc = CLI(cmd)
0509 if cmdrc <> 0 then do
0510     say LOGS(step,"Error during start of H1-H3 of 4-site session" sess)
0511     return cmdrc
0512 end

```

```

0513 say LOGS(step,"Successfully issued" command "to 4-site session" sess)
0514
0515 state = "Prepared"
0516 rolep = "H1-J3"
0517 timeout = gmstartto /* max wait time for GM Restart completion */
0518 say LOGS(step,"Waiting for" state "state of" rolep "pairs in session" sess ,
0519         "(max." timeout "s)...")
0520 cmdrc = CHKPAIRSTATE(sess,rolep,state,timeout)
0521 if cmdrc <> 0 then do
0522     say LOGS(step,"Error while checking" state "state of" rolep ,
0523             "pairs in" sess)
0524     return cmdrc
0525 end
0526 say LOGS(step,"All pairs of" rolep "in" sess "are" state)
0527 say LOGS(step,"Successfully restarted H1-H3 of 4-site session" sess)
0528 say LOGS(step,"Runtime" GETRUNTIME(steptime))
0529 return 0
0530
0531 /*-----*/
0532 /*- SUBROUTINE -*/
0533 /*- -*/
0534 /*- Sequence to enable a practice copy on D volumes (Failover) -*/
0535 /*- It performs Step 5 of the overall sequence. -*/
0536 /*- Return codes: -*/
0537 /*- 0 : All steps executed successfully -*/
0538 /*- ErrMsg: Message describing the problem -*/
0539 /*-----*/
0540 ENABLEPRACTICED:
0541 /* Step 5: Failover cascaded GC rolepair and wait until completed */
0542 step = 5
0543 steptime = time('E')
0544 sess = s4sess
0545 command = "FailoverH4"
0546 say LOGS(step,command "of 4-site session" sess)
0547 cmd = cliex "cmdsess -quiet -action" command sess
0548 cmdrc = CLI(cmd)
0549 if cmdrc <> 0 then do
0550     say LOGS(step,"Error during" command "of 4-site session" sess)
0551     return cmdrc
0552 end
0553 say LOGS(step,"Successfully issued" command "to 4-site session" sess)
0554
0555 state = "Target Available"
0556 rolep = "H3-H4"
0557 timeout = mmrecto /* max wait time for MM Recovery completion */
0558 say LOGS(step,"Check" state "state of" rolep "pairs in" sess"...")
0559 cmdrc = CHKPAIRSTATE(sess,rolep,state,timeout)
0560 if cmdrc <> 0 then do
0561     say LOGS(step,"Error while checking" state "state of" rolep "in" sess)
0562     return cmdrc
0563 end
0564 say LOGS(step,"All pairs of" rolep "in" sess "are" state)
0565
0566 rolep = "H3-H4"
0567 say LOGS(step,"Check" rolep "is recoverable in session" sess"...")
0568 cmdrc = CHKRP(sess,rolep,"YES")
0569 if cmdrc <> 0 then do
0570     say LOGS(step,"Error while checking" rolep ,
0571             "is recoverable in session" sess)
0572     return cmdrc
0573 end
0574 say LOGS(step,rolep "in" sess "is recoverable")
0575 say LOGS(step,"Successfully recovered H4 volumes in 4-site session" sess)
0576 say LOGS(step,"Runtime" GETRUNTIME(steptime))
0577 return 0

```

```

0578
0579 /*-----*/
0580 /*- SUBROUTINE -*/
0581 /*- -*/
0582 /*- Restart 4-site session in case there was an error in the sequence -*/
0583 /*- It performs a check whether GC or GM rolepair is in a state that might -*/
0584 /*- require a Stop first (step R0). Then it performs a restart of -*/
0585 /*- cascaded GC (step R1) and a restart of GM (step R2) in 4-site session -*/
0586 /*- Return codes: -*/
0587 /*- 0 : All steps executed successfully -*/
0588 /*- ErrMsg: Message describing the problem -*/
0589 /*-----*/
0590 RESTARTSESSIONS:
0591 /* Step R0: Check whether Session might require a stop command */
0592 step = "R0"
0593 steptime = time('E')
0594 sess = s4sess
0595 myerror = 0
0596 say LOGS(step,"Check if state of 4-site session" sess "requires a Stop" ,
0597 "command for the GC or GM rolepair...")
0598 validstates = "Draining,Suspending"
0599 cmdrc = CHKSESSSTATE(sess,validstates)
0600 if cmdrc <> 0 then do
0601 /* Good state which does not require a Stop */
0602 say LOGS(step,"Session state" GSstate "does not require a Stop command")
0603 end
0604 else do
0605 /* Bad state which requires a Stop first */
0606 /* Ignore errors in attempt to recover unexpected state of the pairs */
0607 if GSstate = "DRAINING" then do
0608 rolep = "H3-H4"
0609 command = "Stoph3h4"
0610 state = "Suspended"
0611 end
0612 else if GSstate = "SUSPENDING" then do
0613 rolep = "H1-H3"
0614 command = "Stoph1h3"
0615 state = "Suspended"
0616 end
0617 say LOGS(step,"Session state" GSstate "requires a Stop" ,
0618 "command for rolepair" rolep)
0619 say LOGS(step,command "of 4-site session" sess"...")
0620 cmd = cliex "cmdsess -quiet -action" command sess
0621 cmdrc = CLI(cmd)
0622 if cmdrc <> 0 then do
0623 say LOGS(step,"Error during" command "of 4-site session" sess)
0624 say LOGI("ERROR:" cmdrc)
0625 end
0626 else do
0627 say LOGS(step,"Successfully issued" command "of 4-site session" sess)
0628 timeout = gcstopto /* max wait time for GC stop completion */
0629 say LOGS(step,"Waiting for stop completion of 4-site session" sess ,
0630 "(max." timeout "s)...")
0631 cmdrc = CHKPAIRSTATE(sess,rolep,state,timeout)
0632 if cmdrc <> 0 then do
0633 say LOGS(step,"Error while waiting for rolepair" rolep ,
0634 "of 4-site session" sess "to become:" state)
0635 end
0636 else do
0637 say LOGS(step,"Successfully stopped" rolep "of 4-site session" sess)
0638 end
0639 end
0640 end
0641 say LOGS(step,"Runtime" GETRUNTIME(steptime))
0642 say line

```

```

0643 say
0644
0645 /* Step R1: StartGC of cascaded rolepair and wait for preparing state */
0646 step = "R1"
0647 steptime = time('E')
0648 sess = s4sess
0649 command = "StartGC_H3:H4"
0650 myerror = 0
0651 say LOGS(step,command "of 4-site session" sess"..")
0652 cmd = cliex "cmdsess -quiet -action" command sess
0653 cmdrc = CLI(cmd)
0654 if cmdrc <> 0 then do
0655     say LOGS(step,"Error during" command "of 4-site session" sess)
0656     say LOGI("ERROR:" cmdrc)
0657     myerror = cmdrc
0658     /* return cmdrc      Ignore error at this point, try to restart also GM */
0659 end
0660 else do
0661     say LOGS(step,"Successfully issued" command "of 4-site session" sess)
0662     /* Wait for preparing state if there are no errors on start */
0663     state = "Preparing"
0664     rolep = "H3-H4"
0665     timeout = gcstartto      /* max wait time for GC start (Preparing) */
0666     say LOGS(step,"Waiting for" state "state of" rolep "pairs in" ,
0667             "4-site session" sess "(max." timeout "s)..")
0668     cmdrc = CHKPAIRSTATE(sess,rolep,state,timeout)
0669     if cmdrc <> 0 then do
0670         say LOGS(step,"Error while checking" state "state of" rolep ,
0671             "pairs in" sess)
0672         say LOGI("ERROR:" cmdrc)
0673         myerror = cmdrc
0674         /* return cmdrc      Ignore error at this point, try to restart also GM */
0675     end
0676     else do
0677         say LOGS(step,"All pairs of" rolep "in" sess "are" state)
0678     end
0679 end
0680 say LOGS(step,"Runtime" GETRUNTIME(steptime))
0681 say line
0682 say
0683
0684 /* Step R2: StartGM_H1-H3 of 4-site session and wait until H1-J3 prepared */
0685 step = "R2"
0686 steptime = time('E')
0687 sess = s4sess
0688 command = "StartGM_H1:H3"
0689 say LOGS(step,command "of 4-site session" sess"..")
0690 cmd = cliex "cmdsess -quiet -action" command sess
0691 cmdrc = CLI(cmd)
0692 if cmdrc <> 0 then do
0693     say LOGS(step,"Error during" command "of 4-site session" sess)
0694     say LOGI("ERROR:" cmdrc)
0695     myerror = cmdrc
0696 end
0697 else do
0698     say LOGS(step,"Successfully issued" command "to 4-site session" sess)
0699
0700     state = "Prepared"
0701     rolep = "H1-J3"
0702     timeout = gmstartto      /* max wait time for GM Restart completion */
0703     say LOGS(step,"Waiting for" state "state of" rolep "pairs in session" sess ,
0704             "(max." timeout "s)..")
0705     cmdrc = CHKPAIRSTATE(sess,rolep,state,timeout)
0706     if cmdrc <> 0 then do
0707         say LOGS(step,"Error while checking" state "state of" rolep ,

```

```

0708             "pairs in" sess)
0709         say LOGI("ERROR:" cmdrc)
0710         myerror = cmdrc
0711     end
0712     else do
0713         say LOGS(step,"All pairs of" rolep "in" sess "are" state)
0714     end
0715 end
0716 say LOGS(step,"Runtime" GETRUNTIME(stepime))
0717 say line
0718 say
0719
0720 return myerror
0721
0722
0723 /*----- */
0724 /*- SUBROUTINE -*/
0725 /*- -*/
0726 /*- Check status of given taskname and run the task ID -*/
0727 /*- Monitor task status and wait for task completion. -*/
0728 /*- The task should contain step 1-5 of this script -*/
0729 /*- Return codes: -*/
0730 /*- 0 : Task found and completed successfully -*/
0731 /*- ErrMsg: Message describing the problem -*/
0732 /*----- */
0733 RUNTASK:
0734 /* Step T0: Prechecking for an inactive scheduled task */
0735 step = "T0"
0736 steptime = time('E')
0737 task = s4task
0738 taskid = ""
0739 tasksts = ""
0740 taskmsg = ""
0741 say LOGS(step,"Checking task" task"...")
0742 cmdrc = CHKTASK(task,"YES")
0743 if cmdrc <> 0 then do
0744     say LOGS(step,"Error while checking task" task)
0745     call RESETGLOBALS /* reset Globals to indicate nothing started */
0746     return cmdrc
0747 end
0748 taskid = GTid
0749 tasksts = GTstate
0750 say LOGS(step,"Found inactive task" taskid:"task")
0751 say LOGS(step,"Runtime" GETRUNTIME(stepime))
0752 say line
0753 say
0754
0755 /* Step T1: Run task ID */
0756 step = "T1"
0757 steptime = time('E')
0758 say LOGS(step,"Starting task" taskid:"task"...")
0759 cmd = cliex "runtask -quiet" taskid
0760 cmdrc = CLI(cmd)
0761 if cmdrc <> 0 then do
0762     say LOGS(step,"Error while starting task" taskid:"task")
0763     call RESETGLOBALS /* reset Globals to indicate nothing started */
0764     return cmdrc
0765 end
0766 say LOGS(step,"Successfully started task" taskid:"task")
0767 say LOGS(step,"Runtime" GETRUNTIME(stepime))
0768 say line
0769 say
0770
0771 /* Step T2: Wait for task ID completion */
0772 step = "T2"

```



```

0773     steptime = time('E')
0774     timeout = taskto                               /* max wait time for task completion */
0775     say LOGS(step,"Checking completion of task" taskid:"task"...")
0776     cmdrc = CHKTASK(task,,timeout)
0777     if cmdrc <> 0 then do
0778         say LOGS(step,"Error while checking completion of task" taskid:"task")
0779         return cmdrc
0780     end
0781     taskmsg = word(GTmsg,1)
0782     /* IWNR2212I [timestamp] The scheduled task VALUE_1 has finished running. */
0783     if taskmsg <> "IWNR2212I" then do
0784         cmdrc = "Task finished with unexpected message:" taskmsg
0785         return cmdrc
0786     end
0787     say LOGS(step,"Successfully finished task" taskid:"task")
0788     say LOGS(step,"Runtime" GETRUNTIME(steptime))
0789     return 0
0790
0791
0792 /*-----*/
0793 /*- SUBROUTINE                                     -*/
0794 /*-                                               -*/
0795 /*- Check all pairs in given session & role pair are in the given -*/
0796 /*- state. Optionally specify timeout in sec how long to wait for state. -*/
0797 /*- (It uses CSMCLI lspair -l -rolepair command) -*/
0798 /*- Eg: call CHKPAIRSTATE(session,rolepair,state(,timeout)) -*/
0799 /*-     session : String with Session name, use ' ' if it includes spaces -*/
0800 /*-     rolepair: String with rolepair to use for pair state check -*/
0801 /*-     state   : String with state to be validated -*/
0802 /*-     timeout : 0-3600 sec (optional, use to wait for given state) -*/
0803 /*- Return codes: -*/
0804 /*- 0       : All checks are OK -*/
0805 /*- ErrMsg: Message describing the problem -*/
0806 /*-----*/
0807 CHKPAIRSTATE:
0808     parse arg sess1, rp1, state1, to1
0809     rp1 = translate(rp1)
0810     state1 = translate(state1)
0811     mytime = time('E')
0812     if debug >= 2 then do
0813         say LOGD(2,"CHKPAIRSTATE parameters:" sess1","rp1","state1","to1")
0814     end
0815     call RESETGLOBALS                               /* reset Global variables */
0816     mysess = strip(sess1,,"")                       /* remove optional ' */
0817     if to1 = "" then to1 = 0                         /* Default to 0 */
0818     /* calculate delay based on timeout value */
0819     mydelay = GETDELAY(to1)
0820     if mydelay < 0 then do
0821         myrc = "Invalid timeout specified:" to1 "(Valid 0-3600)"
0822         return myrc
0823     end
0824     if debug >= 2 then say LOGD(2,"Specified timeout:" to1 "=> Delay" ,
0825         mydelay "sec")
0826     cmd = cliex "lspair -l -fmt delim -delim ""||dlmch||"" -rolepair" rp1 sess1
0827     do while 1
0828         myrc = CLI(cmd)
0829         if myrc = 0 then do
0830             /* Count pairs with required state and also all other found states */
0831             totalpairs = 0
0832             /* Predefine found states with given state */
0833             foundstates= translate(state1,'_',' ') /* Replace blank with _ */
0834             statescount.0 = 1 /* qty of found states, preset given state */
0835             statescount.1 = 0 /* pre-define found qty of given state = 0 */
0836             do i=1 to out.0
0837                 /* Parse output fields */

```

```

0838      /* Source Volume;Target Volume;Role Pair;State;Recoverable;Copying; */
0839      /* Progress;New;Copy Set;Timestamp;Last Result */
0840      parse upper var out.i p1 (dlmch) ,
0841          p2 (dlmch) ,
0842          p3 (dlmch) ,
0843          p4 (dlmch) , .
0844      if debug >= 2 then say LOGD(2,p1","p2","p3","p4)
0845      if p3 = rp1 then do
0846          totalpairs = totalpairs + 1
0847          paddedp4 = translate(p4,'_',' ') /* Replace blank with _ */
0848          idx = wordpos(paddedp4,foundstates) /* Check if state known */
0849          if idx > 0 then do
0850              /* increase count for found state */
0851              statescount.idx = statescount.idx + 1
0852          end
0853          else do
0854              /* add new found state */
0855              foundstates = foundstates paddedp4
0856              idx = statescount.0 + 1
0857              statescount.idx = 1
0858              statescount.0 = idx
0859          end
0860      end
0861  end
0862  say LOGI(tab sess1 rp1 "Pair state result:")
0863  say LOGI(tab format(totalpairs,4) "pairs total:")
0864  if totalpairs > 0 then do
0865      do j = statescount.0 to 1 by -1
0866          statej = translate(word(foundstates,j),' ','_')
0867          say LOGI(tab format(statescount.j,6) "pairs" statej)
0868      end
0869      if totalpairs = statescount.1 then do
0870          /* Update Global Session variables only if all pairs equal state */
0871          GSname = sess1 /* keep single quotes included in name */
0872          GSrp = p3
0873          GSstate= state1
0874          GScpset= totalpairs
0875          return 0 /* all conditions met */
0876      end
0877  end
0878  else do
0879      myrc = "No pairs found."
0880      return myrc
0881  end
0882  end
0883  else do
0884      if debug >= 2 then say LOGD(2,"ERROR: '"myrc"' while listing pairs")
0885      return myrc /* CSMCLI error */
0886  end
0887  /* apply delay prior next call if necessary */
0888  if debug >= 2 then do
0889      say LOGD(2,"Elapsed:" time('E') - mytime "sec")
0890      say LOGD(2,"mydelay:" mydelay "sec")
0891      say LOGD(2,"Timeout:" to1 "sec")
0892  end
0893  if time('E') - mytime + mydelay >= to1 then do
0894      myrc = "Invalid States found."
0895      if to1 > 0 then myrc = "Wait timeout of" to1 "sec exceeded."
0896      return myrc /* Invalid states */
0897  end
0898  else do
0899      say LOGI(tab "Need to wait for valid state:" state1)
0900      say LOGI(tab "Requering in" mydelay "sec...")
0901      call WAIT(mydelay)
0902  end

```

```

0903     end
0904     return 0
0905
0906 /*----- */
0907 /*- SUBROUTINE -*/
0908 /*- -*/
0909 /*- Check that session reached any of the provided states -*/
0910 /*- Optionally specify timeout in sec how long to wait for valid state. -*/
0911 /*- (It uses CSMCLI lssess -l command) -*/
0912 /*- Eg: call CHKSESSSTATE(session,states(,timeout)) -*/
0913 /*- session: String with Session name, use ' ' if it includes spaces -*/
0914 /*- states : String with comma separated valid states (Use empty string -*/
0915 /*- without timeout to update global variable with state info) -*/
0916 /*- timeout: 0-3600 sec (optional, use to wait for given state) -*/
0917 /*- Return codes: -*/
0918 /*- 0 : All checks are OK -*/
0919 /*- ErrMsg: Message describing the problem -*/
0920 /*----- */
0921 CHKSESSSTATE:
0922     states2 = "" /* preset to empty string */
0923     parse arg sess2, states2, to2
0924     mytime = time('E')
0925     if debug >= 2 then do
0926         say LOGD(2,"CHKSESSSTATE parameters:" sess2,"states2","to2)
0927     end
0928     call RESETGLOBALS /* reset Global variables */
0929     mysess = strip(sess2,,"") /* remove optional ' */
0930     if to2 = "" then to2 = 0 /* Default to 0 */
0931     /* calculate delay based on timeout value */
0932     mydelay = GETDELAY(to2)
0933     if mydelay < 0 then do
0934         myrc = "Invalid timeout specified:" to2 "(Valid 0-3600)"
0935         return myrc
0936     end
0937     if debug >= 2 then say LOGD(2,"Specified timeout:" to2 "> Delay" ,
0938         mydelay "sec")
0939     cmd = cliex "lssess -l -fmt delim -delim '||dlmch||'" sess2
0940     do while 1
0941         myrc = CLI(cmd)
0942         if myrc = 0 then do
0943             mystate = "???"
0944             do i=1 to out.0
0945                 /* Parse output fields */
0946                 /* Name;Status;State;Copy Type;Recoverable;Copying;Copy Sets; */
0947                 /* Error;Group */
0948                 parse var out.i p1 (dlmch) ,
0949                     p2 (dlmch) ,
0950                     p3 (dlmch) ,
0951                     p4 (dlmch) ,
0952                     p5 (dlmch) ,
0953                     p6 (dlmch) ,
0954                     p7 (dlmch) , .
0955                 if debug >= 2 then say LOGD(2,p1,"p2","p3","p4","p5","p6","p7)
0956                 if p1 = mysess then do
0957                     mystate = p3
0958                     /* Update Global Session variables */
0959                     GSname = sess2 /* keep single quotes included in name */
0960                     GSstate= translate(mystate)
0961                     GSrecov= translate(p5)
0962                     GScpset= p7
0963                 end
0964             end
0965             say LOGI(tab sess2 "session state is:" mystate)
0966             /* check if valid state reached */
0967             mystates = translate(states2)

```

```

0968     mystate = translate(mystate)
0969     if mystates = "" then return 0 /* quit immediately, no check required */
0970     do while mystates <> ""
0971         parse var mystates nextstate ',' mystates
0972         if strip(nextstate) = mystate then do
0973             return 0 /* all conditions met */
0974         end
0975     end
0976     end
0977     else do
0978         if debug >= 2 then say LOGD(2,"ERROR: ""myrc"" while listing session")
0979         return myrc /* CSMCLI error */
0980     end
0981     /* apply delay prior next call if necessary */
0982     if debug >= 2 then do
0983         say LOGD(2,"Valid states:" states2)
0984         say LOGD(2,"mydelay:" mydelay "sec")
0985         say LOGD(2,"Timeout:" to2 "sec")
0986     end
0987     if time('E') - mytime + mydelay >= to2 then do
0988         myrc = "Invalid state found."
0989         if to2 > 0 then myrc = "Wait timeout of" to2 "sec exceeded."
0990         return myrc /* Invalid states */
0991     end
0992     else do
0993         say LOGI(tab "Need to wait for valid state:" states2)
0994         say LOGI(tab "Requering in" mydelay "sec...")
0995         call WAIT(mydelay)
0996     end
0997     end
0998     return 0
0999
1000 /*-----*/
1001 /*- SUBROUTINE -*/
1002 /*- -*/
1003 /*- Check given role pair in session. Optionally check whether recoverable -*/
1004 /*- or whether progress exceeds a given percentage. -*/
1005 /*- Optionally specify timeout in sec how long to wait for required condit. -*/
1006 /*- (It uses CSMCLI lsrolepair -l command) -*/
1007 /*- Eg: call CHKRP(session,rolepair,(recoverable,minprogress),(timeout) -*/
1008 /*- session: String with Session name, use ' ' if it includes spaces -*/
1009 /*- rolepair: String with rolepair to use for pair state check -*/
1010 /*- recoverable: (optional) Specify "YES" to validate recoverability -*/
1011 /*- minprogress: 0-100 % (optional, min. Progress in % to be validated) -*/
1012 /*- timeout: 0-3600 sec (optional, use to wait for given state) -*/
1013 /*- Return codes: -*/
1014 /*- 0 : All checks are OK -*/
1015 /*- ErrMsg: Message describing the problem -*/
1016 /*-----*/
1017 CHKRP:
1018     parse arg sess3, rp3, recov3, minprog3, to3
1019     rp3 = translate(rp3)
1020     recov3 = translate(recov3)
1021     mytime = time('E')
1022     if debug >= 2 then do
1023         say LOGD(2,"CHKRP parameters:" sess3,"rp3","recov3","minprog3","to3")
1024     end
1025     call RESETGLOBALS /* reset Global variables */
1026     mysess = strip(sess3,,"") /* remove optional ' */
1027     if to3 = "" then to3 = 0 /* Default to 0 */
1028     /* calculate delay based on timeout value */
1029     mydelay = GETDELAY(to3)
1030     if mydelay < 0 then do
1031         myrc = "Invalid timeout specified:" to3 "(Valid 0-3600)"
1032     return myrc

```

```

1033 end
1034 if debug >= 2 then say LOGD(2,"Specified timeout:" to3 "=> Delay" ,
1035                               mydelay "sec")
1036 cmd = cliex "lsrolepairs -l -fmt delim -delim '||dlmch||'" sess3
1037 do while 1
1038   myrc = CLI(cmd)
1039   if myrc = 0 then do
1040     /* Check specified conditions */
1041     bad = 0
1042     do i=1 to out.0
1043       /* Parse output fields */
1044       /* Name;Recoverable;Error;Copying;Copy Type;Progress;Error Volumes; */
1045       /* Recoverable Pairs;Copying Pairs;Total Pairs;Recovery Time;CG Name*/
1046       parse upper var out.i p1 (dlmch) ,
1047                 p2 (dlmch) ,
1048                 p3 (dlmch) ,
1049                 p4 (dlmch) ,
1050                 p5 (dlmch) ,
1051                 p6 (dlmch) ,
1052                 p7 (dlmch) ,
1053                 p8 (dlmch) ,
1054                 p9 (dlmch) ,
1055                 p10 (dlmch) , .
1056       if debug >= 2 then say LOGD(2,p1", "p2", "p3", "p4", "p5", "p6)
1057       if p1 = rp3 then do
1058         say LOGI(tab sess3 rp3 "Rolepair status:")
1059         say LOGI(tab "Recoverable:" p2)
1060         say LOGI(tab "Progress % :" p6)
1061         /* Update Global Session variables */
1062         GSname = sess3 /* keep single quotes included in name */
1063         GSrp = p1
1064         GSrecov= p2
1065         GSprog = p6
1066         GScpset= p10
1067         if recov3 <> "" & recov3 <> p2 then do
1068           bad = bad + 1
1069           if debug >= 2 then say LOGD(2,"Recoverability not met.")
1070         end
1071         if datatype(p6) <> "NUM" then p6 = -1 /* Ensure Progress is NUM */
1072         if datatype(minprog3) = "NUM" then do
1073           if p6 < minprog3 then do
1074             bad = bad + 1
1075             if debug >= 2 then say LOGD(2,"Minimum Progress not met.")
1076           end
1077         end
1078       end
1079     end
1080     if bad = 0 then return 0 /* All conditions met */
1081   end
1082   else do
1083     if debug >= 2 then say LOGD(2,"ERROR: '"myrc"' while listing rolepair")
1084     return myrc /* CSMCLI error */
1085   end
1086   /* apply delay prior next call if necessary */
1087   if debug >= 2 then do
1088     say LOGD(2,"Elapsed:" time('E') - mytime "sec")
1089     say LOGD(2,"mydelay:" mydelay "sec")
1090     say LOGD(2,"Timeout:" to3 "sec")
1091   end
1092   if time('E') - mytime + mydelay >= to3 then do
1093     myrc = "Invalid rolepair conditions."
1094     if to3 > 0 then myrc = "Wait timeout of" to3 "sec exceeded."
1095     return myrc
1096   end
1097   else do

```

```

1098     say LOGI(tab "Need to wait for valid rolepair conditions.")
1099     say LOGI(tab "Requering in" mydelay "sec...")
1100     call WAIT(mydelay)
1101     end
1102 end
1103 return 0
1104
1105 /*----- */
1106 /*- SUBROUTINE -*/
1107 /*- -*/
1108 /*- Check if task name exists and what the status is. It will update global -*/
1109 /*- task variables with state and last error message. -*/
1110 /*- Optionally specify if check fails if task is active. -*/
1111 /*- Optionally specify timeout in sec how long to wait for task completion. -*/
1112 /*- (It uses CSMCLI lstask command) -*/
1113 /*- Eg: call CHKTASK(taskname(,vfyinactive),(,timeout)) -*/
1114 /*- taskname: String with task name, use ' ' if it includes spaces -*/
1115 /*- vfyinactive: (optional) Specify "YES" to fail check if active -*/
1116 /*- timeout : 0-3600 sec (optional, use to wait for task completion) -*/
1117 /*- Return codes: -*/
1118 /*- 0 : Task found and completed successfully -*/
1119 /*- ErrMsg: Message describing the problem -*/
1120 /*----- */
1121 CHKTASK:
1122     parse arg task1, vfyinact, to1
1123     mytime = time('E')
1124     mytaskid = ""
1125     if debug >= 2 then do
1126         say LOGD(2,"RUNTASK parameters:" task1,"vfyinact","to1)
1127     end
1128     call RESETGLOBALS /* reset Global variables */
1129     mytask = strip(task1,,"") /* remove optional ' */
1130     if to1 = "" then to1 = 0 /* Default to 0 */
1131     /* calculate delay based on timeout value */
1132     mydelay = GETDELAY(to1)
1133     if mydelay < 0 then do
1134         myrc = "Invalid timeout specified:" to1 "(Valid 0-3600)"
1135         return myrc
1136     end
1137     if debug >= 2 then say LOGD(2,"Specified timeout:" to1 "=> Delay" ,
1138         mydelay "sec")
1139
1140     /* Check task status for specified taskname */
1141     cmd = cliex "lstasks -l -fmt delim -delim '|'|d|lmch|'"
1142     do while 1
1143         myrc = CLI(cmd)
1144         if myrc = 0 then do
1145             do i=1 to out.0
1146                 /* Parse output fields */
1147                 /* ID;Name;Status;Schedule;Next Run Time;Last Run Time;Last Result */
1148                 parse var out.i p1 (d|lmch) ,
1149                     p2 (d|lmch) ,
1150                     p3 (d|lmch) ,
1151                     p4 (d|lmch) ,
1152                     p5 (d|lmch) ,
1153                     p6 (d|lmch) ,
1154                     p7 (d|lmch) , .
1155                 if debug >= 2 then say LOGD(2,p1,"p2","p3","p4","p5","p6","p7)
1156                 if p2 = mytask then do
1157                     GTname = task1 /* keep single quotes included in name */
1158                     GTid = p1
1159                     GTstate = p3
1160                     GTmsg = p7
1161                     parse var p6 GTlrt "." .
1162                     GTlrt = strip(GTlrt)

```

```

1163         leave
1164     end
1165 end
1166 if GTid <> "" then do
1167     say LOGI(tab "Task:"GTid " Status:"GTstate " Message:"GTmsg ,
1168             " LastRun:"GTrt)
1169     /* check if task still running */
1170     if translate(p3) <> "RUNNING" then do
1171         return 0 /* all conditions met */
1172     end
1173 end
1174 else do
1175     myrc = "Task not found:" task1
1176     return myrc
1177 end
1178 end
1179 else do
1180     if debug >= 2 then say LOGD(2,"ERROR: '"myrc"' while listing tasks")
1181     return myrc /* CSMCLI error */
1182 end
1183 /* apply delay prior next call if necessary */
1184 if debug >= 2 then do
1185     say LOGD(2,"Elapsed:" time('E') - mytime "sec")
1186     say LOGD(2,"mydelay:" mydelay "sec")
1187     say LOGD(2,"Timeout:" to1 "sec")
1188 end
1189 if time('E') - mytime + mydelay >= to1 then do
1190     if vfyinact = "YES" then do
1191         myrc = "Task is running."
1192     end
1193     if to1 > 0 then myrc = "Wait timeout of" to1 "sec exceeded."
1194     return myrc /* Invalid states */
1195 end
1196 else do
1197     say LOGI(tab "Need to wait for task completion:" p3)
1198     say LOGI(tab "Requerying in" mydelay "sec...")
1199     call WAIT(mydelay)
1200 end
1201 end
1202 return 0
1203
1204 /*-----*/
1205 /*- SUBROUTINE -*/
1206 /*- -*/
1207 /*- Call CSMCLI with specified cmd and verify RC & output streams. -*/
1208 /*- Any CSMCLI framework RC <> 0 will be passed back with more error details-*/
1209 /*- It means the command could not be sent to the server. -*/
1210 /*- If the output streams contain a CSMCLI Error message, the full message -*/
1211 /*- line will be returned. -*/
1212 /*- 0 will be returned if the command was executed without Error message. -*/
1213 /*- Eg: call CLI(command) -*/
1214 /*- command: full single shot csmcli string including executable -*/
1215 /*- Return codes: -*/
1216 /*- 0 : Command was executed without error -*/
1217 /*- ErrMsg: Message describing the problem -*/
1218 /*-----*/
1219 CLI:
1220 parse arg mycommand
1221 if debug >= 9 then say LOGD(9,"CLICMD:" mycommand)
1222 if os = "TSO" then do
1223     clirc = bpxwunix(mycommand,,out.,err.,env.)
1224 end
1225 else if os = "WIN" then do
1226     address SYSTEM mycommand WITH OUTPUT STEM out. ERROR STEM err.
1227     clirc = RC

```

```

1228 end
1229 else return "ERROR: Unknown O/S to run CSMCLI executable"
1230
1231 if clirc <> 0 then do
1232   clirc = "CSMCLI RC" clirc || ":"
1233   /* Add error info line from error or output stream */
1234   if err.0 > 0 then clirc = clirc strip(err.1)
1235   else if out.0 > 0 then clirc = clirc strip(out.1)
1236 end
1237 numlines = out.0
1238 if numlines > 0 then do
1239   /* parse only last 5 lines for error codes */
1240   if numlines > 4 then tail = numlines - 4
1241   else tail = 1
1242   do i=1 to tail
1243     if debug >= 9 then say LOGD(9,"CLIOUT:" strip(out.i))
1244     /* Catch Error message code in last 5 lines and return last code */
1245     if i >= tail then do
1246       if pos("IWN",out.i) > 0 then do
1247         outline = out.i
1248         do while outline <> ""
1249           parse var outline nextword outline
1250           if left(nextword,3) = "IWN" then do
1251             if right(nextword,1) = "E" then do
1252               if debug >=9 then say LOGD(9,"Found Error Msg:" nextword)
1253               clirc = strip(out.i) /* Return full line with msg */
1254             end
1255           end
1256         end
1257       end
1258     end
1259   end
1260 end
1261 numlines = err.0
1262 if numlines > 0 then do
1263   do i=1 to numlines
1264     if debug >= 9 then say LOGD(9,"CLIERR:" strip(err.i))
1265     /* Catch Error message code in any line and return last code */
1266     if pos("IWN",err.i) > 0 | pos("CMM",err.i) > 0 then do
1267       errline = err.i
1268       do while errline <> ""
1269         parse var errline nextword errline
1270         prefix = left(nextword,3)
1271         if prefix = "IWN" | prefix = "CMM" then do
1272           if right(nextword,1) = "E" then do
1273             if debug >=9 then say LOGD(9,"Found Error Msg:" nextword)
1274             clirc = strip(err.i) /* Return full line with msg */
1275           end
1276         end
1277       end
1278     end
1279   end
1280 end
1281 if debug >=9 then say LOGD(9,"RC:" clirc)
1282 return clirc
1283
1284 /*-----*/
1285 /*- SUBROUTINE -*/
1286 /*- -*/
1287 /*- Reset Global Session variables (e.g. prior new CSMCLI queries) -*/
1288 /*-----*/
1289 RESETGLOBALS:
1290 /* Reset global Session variables to keep latest info from last query */
1291 GSname = "" /* Session Name */
1292 GSstate= "" /* Session State */

```



```

1293  GSrp   = "" /* Specific Role Pair if applicable */
1294  GShost = "" /* Session active Host */
1295  GSrecov= "" /* Recoverability of Session/Rolepair */
1296  GSprog = "" /* Progress of Rolepair */
1297  GScpset= "" /* Qty of Copy Sets in Session */
1298  GTid   = "" /* Task ID */
1299  GTname = "" /* Task name */
1300  GTstate= "" /* Task Stauts */
1301  GTlrt  = "" /* Task last run time */
1302  GTmsg  = "" /* Task Message */
1303  return 0
1304
1305  /*-----*/
1306  /*- SUBROUTINE */
1307  /*- */
1308  /*- Prepare system environment for script execution. */
1309  /*- It verifies whether the platform is supported by the script and if so */
1310  /*- it prepares the environment for execution. */
1311  /*- Return codes: */
1312  /*- 0 : Preparation completed successfully */
1313  /*- ErrMsg: Message describing the problem */
1314  /*-----*/
1315  PREPAREENV:
1316  /* Get Operating System */
1317  parse upper source osfull .
1318  os = left(osfull,3)
1319  if os = "TSO" then do
1320  /* Verify if USS syscalls are possible */
1321  address tso
1322  if syscalls('ON') > 3 then do
1323  myrc = "ERROR: Unable to establish the USS SYSCALL environment"
1324  return myrc
1325  end
1326  end
1327  else if os = "WIN" then do
1328  /* initialize environment variables for CSMCLI */
1329  if debug >= 9 then do
1330  say line
1331  say LOGD(9,"Following environment variables have been defined for" ,
1332  "System Calls on" osfull)
1333  end
1334  do i=1 to env.0
1335  parse var env.i envname "=" envvalue
1336  if envname = "PATH" then do
1337  /* extend default system path for CSMCLI */
1338  envvalue = value(envname, 'ENVIRONMENT') || ";" || envvalue
1339  end
1340  /* set environment variable for rexx execution*/
1341  call value envname, envvalue, 'ENVIRONMENT'
1342  if debug >= 9 then do
1343  say LOGD(9,envname="||value(envname, 'ENVIRONMENT'))
1344  if i = env.0 then do
1345  say line
1346  say
1347  end
1348  end
1349  end
1350  end
1351  else do
1352  /* OS not supported */
1353  myrc = "ERROR: Unsupported Operating System found:" osfull
1354  return myrc
1355  end
1356  return 0
1357

```

```

1358 /*----- */
1359 /*- SUBROUTINE -*/
1360 /*- -*/
1361 /*- Create common prefix for messages -*/
1362 /*- Eg: LOGI(message) -*/
1363 /*- message: String to be formatted with prefix -*/
1364 /*----- */
1365 LOGI:
1366     parse arg mymsg
1367     /* Add timestamp as prefix to message */
1368     return time() || ":" mymsg
1369
1370 /*----- */
1371 /*- SUBROUTINE -*/
1372 /*- -*/
1373 /*- Create common prefix for Step messages -*/
1374 /*- Eg: LOGS(stepnum,message) -*/
1375 /*- stepnum: Step number to be used in prefix -*/
1376 /*- message: String to be formatted with prefix -*/
1377 /*----- */
1378 LOGS:
1379     parse arg mystep, mymsg
1380     /* Add timestamp and Step number as prefix to message */
1381     return time() || ": Step" mystep || ":" mymsg
1382
1383 /*----- */
1384 /*- SUBROUTINE -*/
1385 /*- -*/
1386 /*- Create common prefix for debug messages -*/
1387 /*- Eg: LOGD(dbglvl,message) -*/
1388 /*- dbgnum : Debug level to be used in prefix -*/
1389 /*- message: String to be formatted with prefix -*/
1390 /*----- */
1391 LOGD:
1392     parse arg lvl, mymsg
1393     /* Add timestamp and debug prefix to message */
1394     return time() || ": --debug("||lvl||"):" mymsg
1395
1396 /*----- */
1397 /*- SUBROUTINE -*/
1398 /*- -*/
1399 /*- Calculate runtime and format to mm:ss.s based on provided start time -*/
1400 /*- Eg: call GETRUNTIME(starttime) -*/
1401 /*- starttime: Start time saved with time('E') to use for calculation -*/
1402 /*----- */
1403 GETRUNTIME:
1404     parse arg mystarttime
1405     if datatype(mystarttime) = "NUM" then do
1406         myruntime = time('E') - mystarttime
1407         mymin = myruntime % 60
1408         mysec = right(format(myruntime // 60,,1),4,'0')
1409         return mymin || ":" || mysec "(min:sec)"
1410     end
1411     return "?:?? (min:sec)"
1412
1413 /*----- */
1414 /*- SUBROUTINE -*/
1415 /*- -*/
1416 /*- Wait x seconds, utilizing USS system call -*/
1417 /*- Eg: call WAIT(time) -*/
1418 /*- time: Number of seconds to wait -*/
1419 /*----- */
1420 WAIT:
1421     parse arg seconds
1422     if os = "TSO" then do

```

```

1423     address syscall
1424     'sleep (seconds)'
1425     address tso
1426     end
1427     else if os = "WIN" then do
1428         address SYSTEM "timeout" seconds
1429     end
1430
1431     return 0
1432
1433     /*-----*/
1434     /*- SUBROUTINE                                     -*/
1435     /*-                                               -*/
1436     /*- Get delay depending on given timeout. Returns -1 for invalid timeouts. -*/
1437     /*- Eg: call GETDELAY(timeout)                   -*/
1438     /*-      timeout: Overall timeout to calculate appropriate delay (0-3600 sec)-*/
1439     /*-----*/
1440     GETDELAY:
1441         parse arg myto
1442         if datatype(myto) <> "NUM" then return -1
1443         if myto > 0 then do
1444             if      myto <= 30  then return 5           /* 5 s delay within 30 sec */
1445             else if myto <= 120 then return 10        /* 10 s delay within 2 min */
1446             else if myto <= 300 then return 20       /* 20 s delay within 5 min */
1447             else if myto <= 600 then return 30       /* 30 s delay within 10 min */
1448             else if myto <= 3600 then return 60      /* 60 s delay within 60 min */
1449             else return -1
1450         end
1451     return 0

```

7 Appendix: Output of REXX Script example

Following is an example output of the script with a debug level of 0 (No Debug details):

```
-----
14:33:33: Creating Practice Copy on Site 4 (D volumes):
14:33:33: 4-site Session: 'DS-MMGMGC'
14:33:33: Session task   : ''
14:33:33: Pre-Checks      : ENABLED
14:33:33: CSM Server       : 9.155.114.38
14:33:33: Local O/S        : WIN64
14:33:33: Debug Level     : 0
-----

14:33:33: Step 0: Check for Prepared state of H1-J3 pairs in session 'DS-
MMGMGC'...
14:33:40:          'DS-MMGMGC' H1-J3 Pair state result:
14:33:40:             16 pairs total:
14:33:40:             16 pairs PREPARED
14:33:40: Step 0: All pairs of H1-J3 in 'DS-MMGMGC' are Prepared
14:33:40: Step 0: Check for Preparing state of H3-H4 pairs in session 'DS-
MMGMGC'...
14:33:47:          'DS-MMGMGC' H3-H4 Pair state result:
14:33:47:             16 pairs total:
14:33:47:             16 pairs PREPARING
14:33:47: Step 0: Check H3-H4 Progress >= 97 % in session 'DS-MMGMGC'...
14:33:57:          'DS-MMGMGC' H3-H4 Rolepair status:
14:33:57:             Recoverable: NO
14:33:57:             Progress % : 100
14:33:57: Step 0: Progress of H3-H4 in 'DS-MMGMGC' is >= 97 %
14:33:57: Step 0: Pre-Check completed successfully
14:33:57: Step 0: Runtime 0:23.3 (min:sec)
-----

14:33:57: Step 1: SuspendH1H3 of 4-site session 'DS-MMGMGC'...
14:34:08: Step 1: Successfully issued SuspendH1H3 to 4-site session 'DS-MMGMGC'
14:34:08: Step 1: Waiting for suspend completion of 4-site session 'DS-MMGMGC'
(max. 300 s)...
14:34:18:          'DS-MMGMGC' session state is: Suspended (Partial)
14:34:18: Step 1: Successfully suspended H1-H3 of 4-site session 'DS-MMGMGC'
14:34:18: Step 1: Runtime 0:21.5 (min:sec)
-----

14:34:18: Step 2: Determine state of 4-site session 'DS-MMGMGC' from previous
query
14:34:18: Step 2: 'DS-MMGMGC' session state is: SUSPENDED (PARTIAL)
14:34:18: Step 2: FailoverH3 of 4-site session 'DS-MMGMGC'...
14:34:29: Step 2: Successfully issued FailoverH3 to 4-site session 'DS-MMGMGC'
14:34:29: Step 2: Waiting for failover completion of 4-site session 'DS-MMGMGC'
(max. 120 s)...
14:34:39:          'DS-MMGMGC' session state is: Suspended (Partial)
14:34:39: Step 2: Check Target Available state of H1-H3 pairs in 'DS-MMGMGC'...
14:34:49:          'DS-MMGMGC' H1-H3 Pair state result:
14:34:49:             16 pairs total:
14:34:49:             16 pairs TARGET AVAILABLE
14:34:49: Step 2: All pairs of H1-H3 in 'DS-MMGMGC' are Target Available
14:34:49: Step 2: Check H1-H3 is recoverable in session 'DS-MMGMGC'...
14:34:59:          'DS-MMGMGC' H1-H3 Rolepair status:
14:34:59:             Recoverable: YES
```

```

14:34:59:          Progress % : -
14:34:59: Step 2: H1-H3 in 'DS-MMGMGC' is recoverable
14:34:59: Step 2: H3 volumes are consistent in 4-site session 'DS-MMGMGC'
14:34:59: Step 2: Runtime 0:41.0 (min:sec)
-----

14:34:59: Step 3: SuspendH3H4 of 4-site session 'DS-MMGMGC'...
14:35:09: Step 3: Successfully issued SuspendH3H4 to 4-site session 'DS-MMGMGC'
14:35:09: Step 3: Waiting for suspend completion of 4-site session 'DS-MMGMGC'
(max. 120 s)...
14:35:19:          'DS-MMGMGC' session state is: Suspended (Partial)
14:35:19: Step 3: Successfully suspended GC of 4-site session 'DS-MMGMGC'
14:35:19: Step 3: Runtime 0:20.1 (min:sec)
-----

14:35:19: Successfully created practice Copy on Site 4 (D volumes)
14:35:19: Restarting Global Mirror to restore site protection...
-----

14:35:19: Step 4: StartGM_H1:H3 of 4-site session 'DS-MMGMGC'...
14:35:30: Step 4: Successfully issued StartGM_H1:H3 to 4-site session 'DS-MMGMGC'
14:35:30: Step 4: Waiting for Prepared state of H1-J3 pairs in session 'DS-
MMGMGC' (max. 300 s)...
14:35:40:          'DS-MMGMGC' H1-J3 Pair state result:
14:35:40:          16 pairs total:
14:35:40:          16 pairs PREPARING
14:35:40:          0 pairs PREPARED
14:35:40:          Need to wait for valid state: PREPARED
14:35:40:          Requerying in 20 sec...

Waiting for 0 seconds, press a key to continue ...
14:36:09:          'DS-MMGMGC' H1-J3 Pair state result:
14:36:09:          16 pairs total:
14:36:09:          16 pairs PREPARED
14:36:09: Step 4: All pairs of H1-J3 in 'DS-MMGMGC' are Prepared
14:36:09: Step 4: Successfully restarted H1-H3 of 4-site session 'DS-MMGMGC'
14:36:09: Step 4: Runtime 0:49.7 (min:sec)
-----

14:36:09: Global Mirror returned to Prepared State.
14:36:09: Enabling practice Copy on Site 4 (D volumes)...
-----

14:36:09: Step 5: FailoverH4 of 4-site session 'DS-MMGMGC'
14:36:19: Step 5: Successfully issued FailoverH4 to 4-site session 'DS-MMGMGC'
14:36:19: Step 5: Check Target Available state of H3-H4 pairs in 'DS-MMGMGC'...
14:36:29:          'DS-MMGMGC' H3-H4 Pair state result:
14:36:29:          16 pairs total:
14:36:29:          16 pairs TARGET AVAILABLE
14:36:29: Step 5: All pairs of H3-H4 in 'DS-MMGMGC' are Target Available
14:36:29: Step 5: Check H3-H4 is recoverable in session 'DS-MMGMGC'...
14:36:39:          'DS-MMGMGC' H3-H4 Rolepair status:
14:36:39:          Recoverable: YES
14:36:39:          Progress % : -
14:36:39: Step 5: H3-H4 in 'DS-MMGMGC' is recoverable
14:36:39: Step 5: Successfully recovered H4 volumes in 4-site session 'DS-MMGMGC'
14:36:39: Step 5: Runtime 0:30.6 (min:sec)

```


14:36:39: Successfully enabled practice Copy on Site 4.
14:36:39: Systems can be IPLed on Site 4 for testing.

14:36:39: Total Runtime: 3:06.1 (min:sec)
14:36:39: Total RC = 0
