PIPESERV 1.2.4
A Pipeline Server Facility

14 May 2002

Finn Skovgaard

E-mail: finn@skovgaard.org
URL: http://skovgaard.org/

Acknowledgments

The scheduling capabilities of PIPESERV are inspired by TODEVENT, created by Patrick Hallows.

John Hartmann created CMS Pipelines. Without it, there would have been no PIPESERV. He also patiently answered an endless row of pipeline questions during the years, thereby significantly improving the design of PIPESERV. Without John's 24 hour support correcting weird bugs in obscure parts of CMS Pipelines used by PIPESERV, it would never have got off the ground.

During my 1.5 years of absence from IBM, Steve Hayes of IBM UK willingly took PIPESERV under his wings and completed the timing algorithms, and he significantly improved the robustness of the code.

During the same period, Michel Casabona of IBM France pointed out various bugs, fixed them, and generally improved the code.

Hilar Hittinger of IBM Germany has done extensive testing of PIPESERV.

The vast majority of PIPESERV is the work of Finn Skovgaard.

Edition 1.2.6 (14 May 2002)

This edition applies to version 1.2.4 and later of **PIPESERV - A Pipeline Server Facility**.

Contents

Preface

What is PIPESERV?

PIPESERV is a pipeline stage that can run under VM.

It is a virtual machine driver with facilities like command authorisation, IUCV connection, timer interrupts and easy to use information about reader files.

The fact that PIPESERV runs as a pipeline stage makes it differ from other servers like PROP, TODEVENT, HMF, APBOX and VMSERVE, that traditionally run as host commands.

PIPESERV may run without being connected to any other pipeline stages. The user has the option of connecting to one or more input or output streams to make use of some more features of PIPESERV.

As PIPESERV runs as a stage, the user may run other pipeline stages in parallel with PIPESERV

At any time, the console of the virtual machine is available for command execution.

Syntax Notation and Typography

The syntax diagrams follow the standard used by the CMS Pipelines Reference Manual.

References to built-in programs from CMS Pipelines, PIPESERV commands, and PIPESERV input and output streams are written in lowercase *italics*.

Other CMS, pipeline or PIPESERV keywords are written in SMALL UPPERCASE TYPE.

Command examples and samples are written in `monospace Gothic type`.

Examples

Examples like sample terminal sessions and program source files are set in `monospace Gothic type`. The first position of a line of a sample terminal session indicates whether the line was typed on the terminal (blank) or was a response (►).

Summary of changes

Edition 1.2.6, 3 May 2002

- Version 1.2.4 sublevel 0 has been described. This level introduces support for the new IUCV stages in CMS Pipelines, and it updates the synchronous IUCV interface to use supported Pipeline stages, thus making the requirement for PIPSYSF MODULE obsolete.

Edition 1.2.5, 12 April 2002

- Version 1.2.3 sublevel 1 has been described.
- The chapter describing the PIPESERV change history has been taken out of this document and placed in a flat file named PIPESERV CHANGES.

Edition 1.2.4, 3 August 2000

- Version 1.2.3 has been described.
- Clarification of the use of the *cmdt* and *rc* streams.

Edition 1.2.3, 11 May 2000

- The indicator for reader files in nohold was incorrectly stated in “rdr and irdr file output streams” on page 46.
- Version 1.2.2 sublevels 1, 2 and 3 have been described.

Edition 1.2.2, 21 May 1999

- Version 1.2.2 has been described.

Edition 1.2.1, 15 April 1999

- The new features of Version 1.1.1 sublevel 2-5 and Version 1.2.1 sublevel 0-1 have been described.
- The author's details have been updated.
- The list of known problems has been amended.
- An error in the requirements for the synchronous command IUCV interface has been corrected.
- Spelling has been updated to conform better to that used on the British isles.
- A note has been added regarding NORDYMSG and fullscreen.

Edition 1.1.1, 4 April 1997

- The new features of Version 1.1.1 have been described.
- Clarification of PIPSYSF activation.
- Recommendation to use a separate filemode for logging to eliminate any risk of data loss.
- The use of case has been clarified for the synchronous IUCV interface.
- REXX level requirement has been documented.

- VM/ESA Release 2 requirement has been documented.
- It has been documented where to find the PIPESERV package and CMS Pipelines.

Edition 1.1, 22 November 1996

- FORMAT keyword of the *timer* command now documented, as is field names for the timer output record.
- More advice on using serial connections to PIPESERV.

Edition 1.0, 14 November 1996

- Due to a bug in the *gate* stage of CMS Pipelines, a higher level of Pipelines is now required. Using earlier levels of Pipelines may cause premature termination of PIPESERV on EOF on input streams.
- The new features of Version 1.1.0Y and 1.1.0Z have been described.
- The documentation has been tidied up, clarified, and extended a few places.

Edition 1.0 draft, 28 September 1994

- The new features of Version 1.1.0 have been described.
- The discussion about immediate commands has been updated to describe the use of immediate commands in CONSOLE mode.
- The syntax typography for input and output streams has been changed from SMALL UPPERCASE TYPE to lowercase *italics*, because pipeline stream identifiers are case sensitive.

Edition 0.19, 15 July 1994

- The new features of Version 0.2.2 have been described.

Edition 0.18, 27 May 1994

- The new features of Version 0.2.1 have been described.
- SUSPEND command added to the real-life account records example.

Edition 0.17, 20 May 1994

- The new features of Version 0.1.11 have been described.
- New examples have been included to illustrate how to schedule commands for later execution and how to manage account record collection in separate files for each day.
- The program logic part has been extended.

Known problems

Timer weekday ignored

Regardless if a weekday or a specific date is specified on a *timer* or *reminder* command, the timer will expire on any day. For timer requests being written to the *tmr* output stream, the circumvention is to connect a REXX stage to the stream, which checks the weekday or date. Refer to “tmr output stream” on page 47, `DATEWDAY` or `DATEDATE` field.

Features overview

Basic features

Host command shell	Host commands entered on the console are executed.
Initial stack	Records in the CMS program stack will be read at PIPESERV initialisation and executed as host or PIPESERV commands.
PQUERY	The *pquery* command lets you review the current settings.
Interactive REXX	You can interactively execute REXX statements in either a protected environment or in the PIPESERV environment.
Subcommand environment	You can address a different subcommand environment than the default.
Dynamic pipeline addition	During PIPESERV execution, you may dynamically add new pipelines to run in parallel with PIPESERV.
Timer	PIPESERV can be instructed to execute a command, add a pipeline, write a record to an application pipeline or display a reminder at specific times, either once or repeatedly. The *timer* command has a variety of options, allowing you to limit the scope of the timer to for example certain weekdays and certain time ranges. Timer requests can be queried and deleted.
CP *MSG IUCV connection	MSG, SMSG, IMSG, EMSG, CPCONIO, and SCIF is automatically set to IUCV, trapped and logged by PIPESERV.
Reader file information	PIPESERV will query incoming reader files, log and display information on the console.
Logging	Optional logging and automatic housekeeping of log files. Pipelines using PIPESERV may create records for inclusion in the log file.

Optional features

Restart	PIPESERV can be stopped and restarted with the *restart* command. This is useful if you have updated some of the active pipeline programs and want to execute the new versions.
Logical lineend character	As a supplement to the CP LINEND character, you may define a logical lineend character for PIPESERV in PIPESERV CONFIG.

Command authorisation	You may authorise other userids on any RSCS node to execute host commands and PIPESERV commands. The authorisation is based on a control file that may be dynamically updated and reloaded.
Optional console output	Console output may be suppressed if desired. This may be useful if PIPESERV is running in a disconnected server.
Optional ready message	The ready message displayed after each command may be suppressed.
Initial command file	The user may create a PIPESERV INITCMD file containing commands for execution at PIPESERV initialisation. This is particularly useful for specification of permanent timer requests.
Reload command	The *rlddata* command reloads the optional authorisation and initial command files.

Optional programming features

Reader file information records	Information records for incoming reader files are available to your own pipeline stages on the *rdr* output stream if connected.
Initial reader file information	PIPESERV will optionally query existing reader files and write information about them to the *irdr* output stream, if connected, when PIPESERV is started.
Warning records	WNG records are trapped and logged via the CP *MSG IUCV interface if you connect to the *wng* output stream.
Log interface	In addition to the log records generated by PIPESERV, you may append your own records to the log.
Command input stream	If your program connects to the *cmd* input stream, any command presented will be executed as if it was entered on the console.
Alternate command input stream	If your program connects to the *cmdt* input stream, any command presented will be executed as if it was entered on the console, but the command output will be directed to the *cmdt* output stream if connected. if the *cmdt* output stream is not connected, then the command output is written on the console.
Console output stream	Commands entered on the console are available to your program if you connect to the *cons* output stream.
Command output stream	Commands entered via the *cmd* input stream are available to your program if you connect to the *cmd* output stream.
Alternate command output stream	Commands entered via the *cmdt* input stream, together with their console output, are available to your program if you connect to the *cmdt* output stream.

Return code output stream	Return codes from all executed commands are available on the *rc* output stream, if connected.
PIPESERV INITCMD output stream	Commands read from the PIPESERV INITCMD file are available to your program if you connect to the *icmd* output stream.
Stack output stream	The CMS stack contents are available to your program if you connect to the *stac* output stream.
CP *MSG output streams	Any CP *MSG IUCV record, except control records for internal PIPESERV use, is available to your program if you connect to the relevant output stream.
Synchronous IUCV interface	Another userid on the same host, including the userid running PIPESERV, may receive the command output from PIPESERV in a pipeline. This allows applications to verify the execution of commands in PIPESERV, and it allows REXX stages executing in the userid running PIPESERV to execute commands in PIPESERV without connecting to any PIPESERV streams.
Command pre-processing exits	A number of exits is available to allow modification and filtering of incoming commands. Commands targeted for an application can be routed to the application instead of the PIPESERV command processor.

Overview

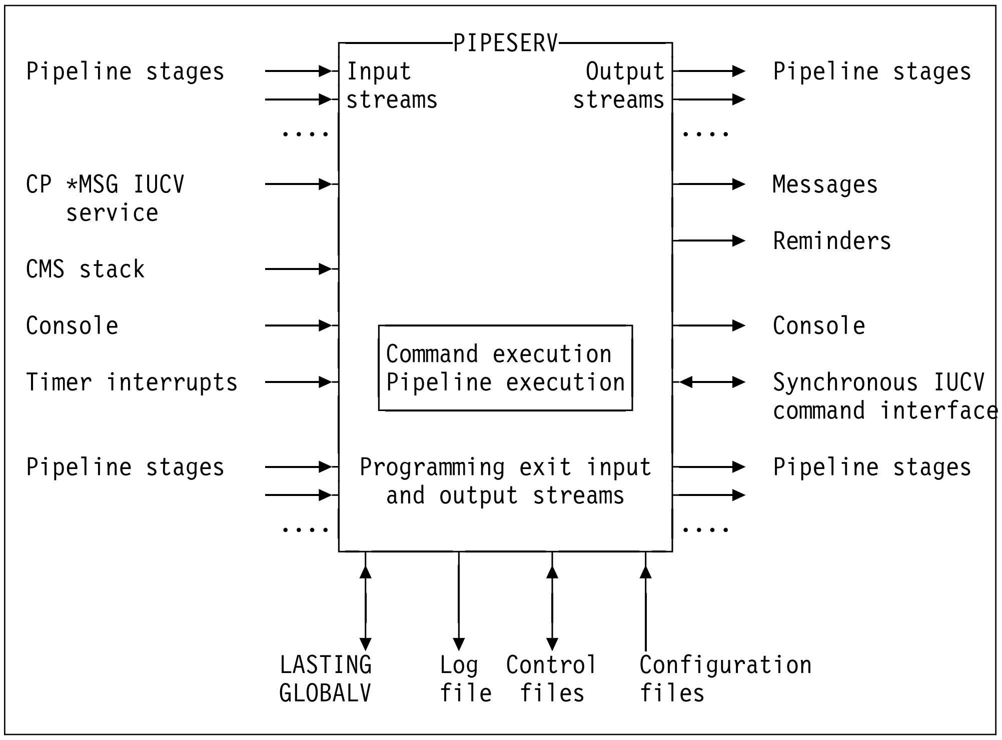


Figure 1. Overview

Planning and Installation

Requirements

- VM/ESA (Version 1) Release 2 or later.
- REXX level 3.48 or higher.
- CMS Pipelines level 110B0004 or later. Use command `pipe query level` to check your level. This level is required for the IUCV support.
- A R/W minidisk or SFS directory accessed as A must be available for PIPESERV's control files and for the `LASTING GLOBALV` file. The control files are used to store timer requests. The space requirement depends on the number of pending timer requests and the length of the string part. For each request, a control record of 141 bytes will be stored in the control file, `PIPESERV savetmr`, including string parts up to 40 bytes per request. Remaining text parts will be stored in a separate, variable length file, `PIPESERV saveext`. The fileids are in lowercase to indicate that these files should not be updated manually. PIPESERV will read and write few variables in the GLOBALV group `PIPESERV`. If too little space is available, PIPESERV may terminate during initialisation. If space runs out during operation, PIPESERV continues, but *timer* and *reminder* commands may be rejected.
- Unless `LOGWRITE OFF` is specified in the PIPESERV configuration file, space must be available for the log files, `yyyymmdd PIPSRVLG`, on the filemode specified. If space runs out during operation, PIPESERV continues, and the log records are written to the virtual console, which will be spooled, instead of to the log file. At midnight, PIPESERV will first try to write log records to the log file. If that fails, it will again use the console. The spooled console records may later be manually appended to the log files if so desired, as they are in the same format.

  Note: It is recommended to use a separate minidisk for the log file, or to use an SFS directory, because having a constantly open file being updated in place on a minidisk imposes a slight risk of corrupting the whole minidisk. This will only happen under rare circumstances, and it is described in the CMS User's Guide.

Installation

Load the files from PIPESERV PACKAGE onto a minidisk.

PIPESERV can be downloaded from http://www.vm.ibm.com/download/ .

If you plan to use the PIPSRVRB utility, download PROMPT from http://www.vm.ibm.com/download/ and install it.

Install the level of CMS Pipelines required. It can be downloaded from http://vm.marist.edu/~pipeline/ .

Tailoring

PIPESERV CONFIG

Optionally edit the PIPESERV SAMPCONF file to suit your needs and file it as another fileid. If it is not filed as PIPESERV CONFIG, you must specify its name upon invocation of PIPESERV. Refer to “PIPESERV syntax” on page 13.

The file may be fixed or variable and may have any record length.

The records in the file must conform to the following standard:

Token 1: If the first non-blank character in a record is a "*", the record is treated as a comment. Otherwise, the first token (word) is taken as a keyword.

Token 2-*: For non-comment records, token 2-* contains parameters for the keyword.

Keyword and comment records may appear in any order and may be intermixed if desired.

Keywords	**Parameters**
KEEPLOG	Specifies how many days the log files are to be kept. Log files are named `yyyymmdd PIPSRVLG A6`. PIPESERV automatically erases the log files when expired. If there is more than one KEEPLOG record, the first record takes precedence.
MSGCMD	Specifies the CP message command to be used when sending messages to other userids (MSG or MSGNOH). The default is MSGNOH. Note that the MSGNOH command requires CP privilege class B if your system has not redefined it using the User Class Restricture facility of CP. If there is more than one MSGCMD record, the first record takes precedence.
MSGNOH	Specifies one or more blank-separated userids for which no message header is to be displayed on your console. As MSG and MSGNOH are presented to you as the same IUCV class (1), PIPESERV cannot separate MSGNOH from MSG. As many MSGNOH records as desired may be specified.
RESTART	Specifies a CMS command to be executed to restart PIPESERV when it has terminated following a *restart* command. If there is more than one RESTART record, the first record takes precedence.
LINEND	Specifies a logical lineend character to allow multiple logical commands to be entered to PIPESERV as one command string. Note that you must be authorised for each logical command, as the command string is split before the authorisation check.
LOGWRITE	Specifies if one log record at a time is written to the log file, if the log file is only updated when an in-storage I/O buffer is full or if no log file is to be written. Also specifies if the log file is to be closed for every record written. Specify SAFE to write one record at a time and close the log file for every record. Specify SLOW to write one record at a time without closing the log file. Specify FAST to write a complete buffer at a time without closing the log file. Specify OFF if no log file is to be written.

Note: You cannot use the *logx* pre-processing exit if you specify LOGWRITE OFF. SAFE is recommended for normal use, as it allows you to browse the log file and see the latest entries during PIPESERV operation. FAST is recommended for high-activity userids, because writing one record at a time may severely degrade performance.

LOGFM Specifies the filemode where the log file is to be written. Default is A. If space runs out during operation, PIPESERV continues, and the log records are written to the virtual console, which will be spooled, instead of to the log file. At midnight, PIPESERV will first try to write log records to the log file. If that fails, it will again use the console. The spooled console records may later be manually appended to the log files if so desired, as they are in the same format.

Note: It is recommended to use a separate minidisk for the log file, or to use an SFS directory, because having a constantly open file being updated in place on a minidisk imposes a slight risk of corrupting the whole minidisk. This will only happen under rare circumstances, and it is described in the CMS User's Guide.

```
MSGCMD MSGNOH
MSGNOH TODEVENT RSCS DIRMAINT PVM RACFVM RACMAINT
KEEPLOG 7
RESTART PIPE PIPESERV
LINEND ;
LOGWRITE SAFE
LOGFM A
```

Figure 2. Sample PIPESERV CONFIG file

PIPESERV AUTHUSER

Optionally edit the PIPESERV SAMPAUTH file to suit your needs and file it as PIPESERV AUTHUSER. This file controls which userids may execute commands in the userid running PIPESERV. If the file does not exist on any accessed filemode, no other userids will be authorised. You may edit the file while PIPESERV is running. When you issue the *rlddata* command, the file will be reloaded into PIPESERV.

The file may be fixed or variable and may have any record length.

Comment records are identified by a * (asterisk) in column 1. They will not be processed.

Other records contain authorisations.

Format for authorisation records:

Token 1: Userid(s) to be authorised.

Token 2: RSCS nodeid(s) on which the userid(s) are authorised.

Token 3-*: The host commands or PIPESERV commands that may be executed. The first word may optionally be CMD, but this is redundant.

There is no need to authorise the userid running PIPESERV, as commands originating from this userid will bypass the authorisation check automatically.

All records are tokenised in accordance with the CMS standard, that is, all words are uppercased and truncated to 8 bytes by PIPESERV, and left and right parentheses are regarded as separate tokens.

Any token may be specified in any of the following ways:

- As a complete word that must match.
- As `*` (asterisk), meaning that any word matches.
- As a partial word, ending with an `*` (asterisk), meaning that the specified characters must match. For example: `Q*` will match `QUERY`. `LIST*` will match `LIST` and `LISTFILE`, but not `LOCATE`.

Even though the examples are aligned in columns, you may place the control words in any columns you like.

Note: If you specify userid as `*`, you cannot place it in column 1, as this would be regarded as a comment record. So to avoid mistakes, you may prefer to leave column 1 blank except for comments.

```
* Userid   Nodeid   Command prefix
* -------- -------- ----------------------------------------------------
*Allow any user on any node to use the REM command:
 *        *        REM
*Allow GOOFY on any node to use any command:
 GOOFY    *        *
*Allow anybody on any node starting with VMNODE to use the TMR command:
 *        VMNODE*  TMR
*Allow MICKEY on VMNODE3 to execute any command starting with Q:
 MICKEY   VMNODE3  Q*
*Allow MICKEY on VMNODE3 to execute QUERY DISK:
 MICKEY   VMNODE3  QUERY DISK
```

Figure 3. Sample PIPESERV AUTHUSER file

PIPESERV INITCMD

Optionally create a `PIPESERV INITCMD` file. Records in this file will be executed as host or PIPESERV commands before the CMS stack is emptied. This is particularly useful for specifying permanent timer requests. You may edit or create the file while PIPESERV is running. When you issue the *rlddata* command, the file will be reloaded into PIPESERV, and any pending permanent timer requests will be deleted. Note that only *timer* and *reminder* commands from this file are executed as a result of an *rlddata* command.

The file may be fixed or variable, and the record length is unlimited. Commands may begin in any column.

Getting started

To use PIPESERV in its simplest form, all you have to do is enter `pipe pipeserv` from the CMS command line. This will make all basic features (see “Basic features” on page 3) available to you. You may want to try some simple commands:

```
 pipe pipeserv
►Loading authorization file PIPESERV AUTHUSER A1 dated 2000-05-10 16:03:20
►
►                      PIPESERV Version 1.2.4 running.
►    (c) Copyright International Business Machines Corporation 1996, 2000.
►                            All Rights Reserved.
►
►Ready(); 2000-05-20 15:35:23 (SKOVGAF at GFORD1)
 listfile * blah
►DMSLST002E File not found
►Ready(28); 2000-05-20 15:35:40 (SKOVGAF at GFORD1)
 listfile * pipsrvlg *
►19970402 PIPSRVLG Z6
►19970312 PIPSRVLG Z6
►19970313 PIPSRVLG Z6
►19970314 PIPSRVLG Z6
►19970317 PIPSRVLG Z6
►19970318 PIPSRVLG Z6
►19970319 PIPSRVLG Z6
►19970320 PIPSRVLG Z6
►19970321 PIPSRVLG Z6
►19970322 PIPSRVLG Z6
►19970324 PIPSRVLG Z6
►19970325 PIPSRVLG Z6
►19970326 PIPSRVLG Z6
►19970327 PIPSRVLG Z6
►19970401 PIPSRVLG Z6
►Ready(0); 2000-05-20 15:35:50 (SKOVGAF at GFORD1)
 rem +::60 string I want a reminder after 60 seconds
►Timer request number 1 stored
►Ready(0); 2000-05-20 15:36:12 (SKOVGAF at GFORD1)
 rexx say 2**8
►256
►Ready(0); 2000-05-20 15:36:21 (SKOVGAF at GFORD1)
 restart
►Restart requested by SKOVGAF at GFORD1
►Ready; T=1.01/1.11 15:36:26
►Loading authorization file PIPESERV AUTHUSER A1 dated 2/18/97 16:03:20
►PIPESERV restarted after request by SKOVGAF at GFORD1
►
►                      PIPESERV Version 1.2.4 running.
►    (c) Copyright International Business Machines Corporation 1996, 1997.
►                            All Rights Reserved.
►
►Ready(); 2000-05-20 15:36:28 (SKOVGAF at GFORD1)
►Loading authorization file PIPESERV AUTHUSER A1 dated 2/18/97 16:03:20
 reminder query
►Requestno. Userid   Nodeid   Time         Date  Rep A T String (A=Actn T=Type)
►---------- -------- -------- --- -------- ----- --- - - ----------------------
►         1 SKOVGAF  GFORD1   Rel 00:01:00 Any   Onc R T I want a reminder afte
►Ready(0); 2000-05-20 15:36:42 (SKOVGAF at GFORD1)
 pquery level
►PIPESERV Version 1.2.4 sublevel 0, released on 19970310 10:20
►Ready(0); 2000-05-20 15:36:55 (SKOVGAF at GFORD1)
 rlddata
►Loading authorization file PIPESERV AUTHUSER A1 dated 2/18/97 16:03:20
►Reloading TIMER & REMINDER commands from PIPESERV INITCMD A1 dated 11/22/96 11:02:58
►Ready(0); 2000-05-20 15:37:07 (SKOVGAF at GFORD1)
►Timer request number 1:
►You have requested the following reminder text to be sent to your userid:
►I want a reminder after 60 seconds
 stop
►Ready; T=1.47/1.60 15:37:30
```

Figure 4. Sample PIPESERV session

The Ready messages are generated by PIPESERV. The return codes from commands entered on the console are always shown in parantheses after Ready. When PIPESERV is started, no return code is displayed, as no commands have been entered on the console.

Following the return code is the current date and time, the userid running PIPESERV, and the RSCS nodeid.

In the logfile, `19970402 PIPSRVLG Z6`, we can see what happened:

```
15:35:23 Pipeserv SKOVGAF  GFORD1   PIPESERV Version 1.2.4 sublevel 0 running, released on 19970310 10:20
15:35:40 Console  SKOVGAF  GFORD1   listfile * blah
15:35:40 Pipeserv SKOVGAF  GFORD1   Executing command for SKOVGAF at GFORD1: listfile * blah
15:35:50 Console  SKOVGAF  GFORD1   listfile * pipsrvlg *
15:35:50 Pipeserv SKOVGAF  GFORD1   Executing command for SKOVGAF at GFORD1: listfile * pipsrvlg *
15:36:11 Console  SKOVGAF  GFORD1   rem +::60 string I want a reminder after 60 seconds
15:36:11 Pipeserv SKOVGAF  GFORD1   Executing command for SKOVGAF at GFORD1: rem +::60 string I want a reminder after 60 seconds
15:36:12 Pipeserv SKOVGAF  GFORD1   Timer request number 1 stored
15:36:20 Console  SKOVGAF  GFORD1   rexx say 2**8
15:36:20 Pipeserv SKOVGAF  GFORD1   Executing command for SKOVGAF at GFORD1: rexx say 2**8
15:36:26 Console  SKOVGAF  GFORD1   restart
15:36:26 Pipeserv SKOVGAF  GFORD1   Executing command for SKOVGAF at GFORD1: restart
15:36:27 Pipeserv SKOVGAF  GFORD1   PIPESERV Version 1.2.4 sublevel 0 running, released on 19970310 10:20
15:36:27 Pipeserv SKOVGAF  GFORD1   PIPESERV restarted after request by SKOVGAF at GFORD1
15:36:27 Ireader  SKOVGAF  GFORD1   2444 SKOVGAF  GFORD1                     18 T00 CON SKOVGAF  19970402 12:34:36       74 n 0
15:36:42 Console  SKOVGAF  GFORD1   reminder query
15:36:42 Pipeserv SKOVGAF  GFORD1   Executing command for SKOVGAF at GFORD1: reminder query
15:36:55 Console  SKOVGAF  GFORD1   pquery level
15:36:55 Pipeserv SKOVGAF  GFORD1   Executing command for SKOVGAF at GFORD1: pquery level
15:37:06 Console  SKOVGAF  GFORD1   rlddata
15:37:06 Pipeserv SKOVGAF  GFORD1   Executing command for SKOVGAF at GFORD1: rlddata
15:37:12 TimerExp SKOVGAF  GFORD1            101 7291151997040256171                    SKOVGAF GFORD1  R--00060    0*0864000000086
15:37:12 TimerEnd SKOVGAF  GFORD1            1
15:37:12 Pipeserv SKOVGAF  GFORD1   Timer request number 3 has been deleted.
15:37:30 Console  SKOVGAF  GFORD1   stop
15:37:30 Pipeserv SKOVGAF  GFORD1   Executing command for SKOVGAF at GFORD1: stop
```

Figure 5. Log file from a sample PIPESERV session

PIPESERV syntax

Use the following syntax when calling the PIPESERV stage from a pipeline.

```
►►──PIPESERV──┤ configfid ├──┬────────────────────────────────┬──────►◄
                             └─(──┬────────────────┬──┬───┬──┘
                                  └─┤ options ├(1)─┘  └─)─┘
```

options:

```
├──┬─DISPLAY───┬──┬─INITRDR───┬──┬─CONSOLE─┬──┬─RDYMSG───┬──────┤
   └─NODISPLAY─┘  └─NOINITRDR─┘  └─IMMCMD──┘  └─NORDYMSG─┘
```

configfid:

```
├──┬─────────────────────────────────────────────────┬──────────┤
   └─┬─PIPESERV─┬──┬─CONFIG *─────────────────────┬──┘
     └─filename─┘  │                              │
                   └─┬─CONFIG───┬──┬─*────────┬───┘
                     └─filetype─┘  │          │
                                   └─filemode─┘
```

Note:
1 Options may be specified in any order.

DISPLAY	Display messages, warnings, CP informational messages, SCIF from the CP *MSG service (IUCV classes 1, 2, 7 and 8 respectively) and reader file information on the console. Note that only if you have connected to the WNG output stream, warning display is controlled by PIPESERV. Otherwise, warnings will always be displayed, clearing the display. This is the default.
NODISPLAY	Suppress display of messages, warnings, CP informational messages, SCIF and reader file information. See also the DISPLAY option. NODISPLAY does not suppress logging.
INITRDR	At PIPESERV invocation, query existing reader files and write information about them on the *irdr* output stream, if connected. The information will be written to the log file as well. This is the default.
NOINITRDR	Suppress query of existing reader files at PIPESERV invocation, regardless if the *irdr* output stream is connected. No information about existing reader files is written to the log file.
CONSOLE	All console input is controlled by PIPESERV. Immediate CMS commands must be prefixed with the CP terminal linend character to be recognised. See “PIPESERV commands” on page 15 for a discussion of IMMCMD versus CONSOLE. CONSOLE is the default.
IMMCMD	Console input is managed by the use of immediate commands. See “PIPESERV commands” on page 15 for a discussion of IMMCMD versus CONSOLE.
RDYMSG	Tell PIPESERV to display a ready message after completion of each command. This is the default.

NORDYMSG	Tell PIPESERV not to display a ready message after completion of each command. **Note:** If you run PIPESERV in NORDYMSG mode and enter fullscreen mode by executing for example a FILELIST command, then you need to press the ATTN key or send a msg to the userid in order to revert to line mode.
filename	Specify CMS filename of the PIPESERV configuration file. Default is `"PIPESERV"`. If you do not specify a configuration file, and the default file is not found, PIPESERV will assign default values for the variables in the configuration file.
filetype	Specify CMS filetype of the PIPESERV configuration file. Default is `"CONFIG"`. If you do not specify a configuration file, and the default file is not found, PIPESERV will assign default values for the variables in the configuration file.
filemode	Specify CMS filemode of the PIPESERV configuration file. The default is to search all filemodes. If you do not specify a configuration file, and the default file is not found, PIPESERV will assign default values for the variables in the configuration file.

PIPESERV commands

Once PIPESERV is running, you may use the PIPESERV commands described in this chapter interactively.

PIPESERV commands can be entered in six different ways:

1. In the `PIPESERV INITCMD` file. These commands are the first to be executed at PIPESERV initialisation. Refer to “PIPESERV INITCMD” on page 9 for a description of this file.
2. In the CMS stack before PIPESERV invocation. These commands are executed immediately after the commands in the `PIPESERV INITCMD` file.
3. On the console of the userid running PIPESERV.
4. Via the *cmd* input stream. See “Command input stream” on page 30 and “cmd input stream” on page 45.
5. Via the *cmdt* input stream. See “Alternate command input stream” on page 30 and “cmdt input stream” on page 45.
6. Via a MSG or SMSG from a userid on the same host, or via an RSCS message from a userid on a remote RSCS node.
7. Via the synchronous IUCV command interface from a userid on the same host.

   To execute the command `query disk` in PIPESERV and receive the command output in the rexx stem `reply.`:

   ```
   'PIPE literal q disk|iucvclient serverid name cmd|stem reply.'
   ```

   Where `serverid` is the userid of the server running PIPESERV and `cmd` is the service requested from the server Only the service `cmd` is supported by PIPESERV. Do not confuse the service with the *cmd* PIPESERV command. Refer to “Synchronous IUCV command interface” on page 33 for a full description.

   If executed from a rexx stage executing in the same userid as PIPESERV, the `PIPE` command must be replaced with `callpipe` or `addpipe`. If not, the pipeline will stall. No host commands may be active when this interface is used to communicate with PIPESERV in the same userid, as they would suspend PIPESERV execution, thereby making it stall.

Execution of commands from other userids must be authorised in the `PIPESERV AUTHUSER` file. Logical commands separated by the PIPESERV logical lineend character are verified individually. Commands to be executed as a result of an expiring timer request are verified before execution. See “PIPESERV AUTHUSER” on page 8. PIPESERV always accepts commands from the userid where it is running.

Managing console input

PIPESERV can handle console input in two different ways, depending on the option you specify on the `PIPESERV` command.

CONSOLE

If you do not specify the IMMCMD option on the PIPESERV command, PIPESERV defaults to CONSOLE.

This has the advantage that any command entered on the console is handled directly by PIPESERV. If the command is not recognised as a PIPESERV command, it is executed as a host command, where the normal CMS command resolution takes place.

The disadvantage is that immediate CMS commands have to be prefixed with the CP terminal linend character to be recognised by CMS.

IMMCMD

If you specify the IMMCMD option on the `PIPESERV` command, PIPESERV is only reacting immediately on console input that is prefixed with the word *cmd*. Other console input, not recognised by CMS as immediate CMS commands, is placed in the terminal input buffer.

The advantage is that you can use immediate CMS commands without prefixing them with the CP terminal linend character.

The disadvantage is that host commands entered on the console must be prefixed with *cmd* to be executed immediately. If you forget to prefix a command with *cmd*, it will be placed in the terminal input buffer instead of being executed. However, whenever PIPESERV is activated, for example by a PIPESERV command, the terminal input buffer will be emptied and the records executed as commands. So if you forgot to prefix your host commands with *cmd*, the trick is to enter a *cmd* command without arguments, thereby forcing PIPESERV to empty the terminal input buffer.

Command syntax

host command

```
►►──host command──────────────────────────────────────────►◄
```

Host commands are executed using the standard CMS command resolution. If the IMMCMD option is specified on the PIPESERV command, host commands entered on the console must be prefixed with *cmd* for immediate execution.

The host command may identify the userid that requested the execution by inspecting the variables `ORIGUSER` and `ORIGNODE` in GLOBALV group PIPESERV. These variables contain the userid and RSCS nodeid respectively, and for the duration of the command execution.

Note: You may use immediate CMS commands as described above. For application debugging purposes, HI (Halt Interpretation) may be useful, because PIPESERV will continue execution, while other REXX programs will be halted. HX (Halt eXecution) will halt PIPESERV as well as other programs. Refer to the CMS documentation for more information about immediate commands.

CMD

```
►►──CMD──┬──────────────────┬──────────────────────────────────────►◄
         ├─host command─────┤
         └─PIPESERV command─┘
```

In some cases, commands must be prefixed with *cmd* for PIPESERV to identify the input as a command.

Table 1 summarizes the use of the *cmd* prefix. Brackets ("[]") around *cmd* indicates that it is optional. Any host command or PIPESERV command may be prefixed with *cmd* at any time, regardless how it is entered to PIPESERV.

Table 1. CMD use, overview

Input method	CONSOLE option	IMMCMD option

Console	`[CMD] host-command`
	`[CMD] PIPESERV-command`
		`CMD host-command` ♠
		`CMD PIPESERV-command` ♠

Stack	`[CMD] host-command`
	`[CMD] PIPESERV-command`	

PIPESERV INITCMD	`[CMD] host-command`
	`[CMD] PIPESERV-command`	

cmd **input stream**	`[CMD] host-command`
	`[CMD] PIPESERV-command`	

cmdt **input stream**	`[CMD] host-command`
	`[CMD] PIPESERV-command`	

MSG	`CMD host-command`
	`CMD PIPESERV-command`	

SMSG	`[CMD] host-command`
	`[CMD] PIPESERV-command`	

Sync. IUCV interface	`[CMD] host-command`
	`[CMD] PIPESERV-command`	

♠: However, see “IMMCMD” on page 16.

ADDRESS (subcommand environment)

```
►►──ADDRess──subcommand environment──subcommand command────────────►◄
```

The *address* command lets you execute a command in a different subcommand environment than the default of CMS.

"subcommand environment" must be available at the time of execution.

"subcommand command" must be a valid command in the specified environment.

For example, to have an active XEDIT environment while running PIPESERV, go into XEDIT and start PIPESERV from there. Underneath the PIPESERV/CMS console, you will have an XEDIT environment. From the command line, you can issue commands to XEDIT, like for example

```
address xedit refresh
address xedit status
```

Refer to “Using PIPESERV with an active, parallel XEDIT environment” on page 41 for a guide to using XEDIT together with PIPESERV.

DEBUG

```
►►──DEBUG──rexx-expression─────────────────────────────────────►◄
```

Use the *debug* command to execute REXX expressions interactively within the PIPESERV environment.

You may use the REXX command separator ; to execute more than one REXX expression in the same *rexx* command.

You have access to PIPESERV's internal rexx variables, as well as its input and output streams. Modifying any of these may cause unpredictable results.

The environment is pipeline.

Refer to the description of the command “REXX” on page 20 for examples.

Note: If executing a pipeline command like CALLPIPE or ADDPIPE from another userid, via the *cmdt* input stream, or via the synchronous IUCV interface, console output destined for the server may be mixed up with console output from the pipeline command. This is due to the design of CMS Pipelines and PIPESERV. For ADDPIPE commands, it is unpredictable if any console output is written to the server console or to the command origin, because of its asynchronous nature.

Note: Do not alter the environment permanently, for example by using the ADDRESS command without a host command.

PQUERY

```
            ┌─All─────┐
►►──PQuery──┼─────────┼────────────────────────────────────────►◄
            ├─Version─┤
            ├─Level───┤
            ├─Config──┤
            ├─Options─┤
            └─AUths───┘
```

The *pquery* command displays active information about PIPESERV.

If the VERSION keyword is specified, only the PIPESERV version and maintenance level (sublevel) are displayed. LEVEL is a synonym for VERSION.

```
pquery version
►PIPESERV Version 1.2.4 sublevel 0, released on 19970310 10:20
```

Figure 6. Sample output from a pquery version command.

If the CONFIG keyword is specified, the name of the current PIPESERV configuration file - if any - is displayed, followed by the current configuration values.

```
 pquery config
►PIPESERV configuration values from file PIPESERV CONFIG *:
►  MSGCMD = MSG
►  MSGNOH userid(s):
►    TODEVENT
►    RSCS
►    DIRMAINT
►    PVM
►    RACFVM
►    RACMAINT
►    HMFSERVE
►  KEEPLOG = 7
►  RESTART = PIPE PIPESERV
►  LINEND = ;
►  LOGWRITE = SAFE
►  LOGFM = A
```

Figure 7. Sample output from a pquery config command.

Refer to the sample configuration file PIPESERV SAMPCONF or “PIPESERV CONFIG” on page 7 for a description of the individual parameters.

If the OPTIONS keyword is specified, only the options specified on the PIPESERV command are displayed.

```
 pquery options
►Active PIPESERV options:
►  DISPLAY
►  INITRDR
►  RDYMSG
►  CONSOLE
```

Figure 8. Sample output from a pquery options command.

If the AUTHS keyword is specified, only the current PIPESERV authorisations are displayed.

```
 pquery auths
►Active PIPESERV authorisations:
►  *        *        REM
►  MAINT    GFORD1   *
►  SYSCNTRL GFORD1   *
```

Figure 9. Sample output from a pquery auths command.

If the ALL keyword or no keywords are specified, all of the above information is displayed.

REMINDER

The *reminder* command is a synonym for the *timer* command, except that the REMINDER option is enforced. This permits general users to take advantage of the timing facilities without allowing them to schedule command execution.

RESTART

```
►►──RESTART──────────────────────────────────────────────────►◄
```

The *restart* command terminates the PIPESERV stage and all its connections. Before termination, it places the command specified in the RESTART record in the PIPESERV configuration file in the CMS stack. See “Tailoring” on page 7. Timer requests will survive from one invocation of PIPESERV to the next, unless the VOLATILE option of the *timer* or *reminder* command was specified by the user or was implied by entering the commands via the *cmd* input stream or the CMS stack without specifying the TEMPORARY or PERMANENT option.

REXX

```
►►──REXX──rexx-expression────────────────────────────────────►◄
```

Use the *rexx* command to execute REXX expressions interactively. For example, you can use PIPESERV as a calculator with memory.

```
 rexx a = 2
 rexx b = 4
 rexx say a*b
►8
```

Sometimes it is useful to try how REXX commands work in real life instead of searching the manuals. Perhaps you cannot remember the exact output from the DATE function. You would enter

```
rexx say date()
►5 Nov 1996
```

You may use the REXX command separator ; to execute more than one REXX expression in the same *rexx* command. To execute a loop, you could enter

```
rexx do i = 1 to 3;say i;end
►1
►2
►3
```

PIPESERV will normally be able to recover from REXX syntax errors.

REXX variables are kept in a protected environment, where you cannot access PIPESERV's internal REXX variables. Your variables are not stored when PIPESERV is stopped or restarted.

The environment is pipeline. That is, you can issue commands like

```
rexx 'callpipe literal a | console'
rexx 'addpipe literal q disk | iucvclient PIPEUSER name cmd | console'
rexx address command 'Q DISK'
```

With the ADDPIPE command, you may start new pipelines to run in parallel with the PIPESERV stage.

Note: If executing a pipeline command like CALLPIPE or ADDPIPE from another userid, via the *cmdt* input stream, or via the synchronous IUCV interface, console output destined for the server may be mixed up with console output from the pipeline command. This is due to the design of CMS Pipelines and PIPESERV. For ADDPIPE commands, it is unpredictable if any console output is written to the server console or to the command origin, because of its asynchronous nature.

Note: Do not alter the environment permanently, for example by using the ADDRESS command without a host command.

RLDDATA

```
►►──RLDData────────────────────────────────────────────────────►◄
```

The *rlddata* command reloads the `PIPESERV AUTHUSER` and the `PIPESERV INITCMD` files. This allows you to update these files without interrupting the PIPESERV service. All pending permanent timer requests, except those including an IPL, START, or RESTART keyword, are deleted. Only *timer* and *reminder* commands from the `PIPESERV INITCMD` file are executed.

STOP

```
►►──STOP───────────────────────────────────────────────────────►◄
```

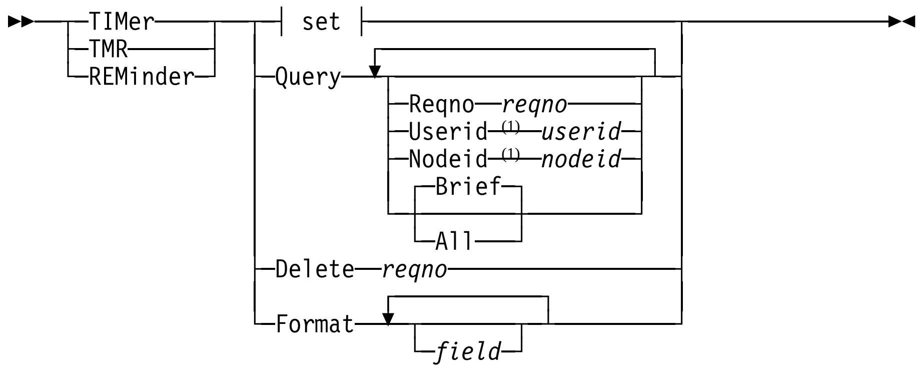
The *stop* command terminates the PIPESERV stage and all its connections. Timer requests will survive from one invocation of PIPESERV to the next, unless the VOLATILE option of the *timer* or *reminder* command was specified by the user or was implied by entering the commands via the *cmd* input stream or the CMS stack without specifying the TEMPORARY or PERMANENT option.

In case you cannot stop PIPESERV normally with a *stop* command from the console or from an authorised userid, PIPESERV has a built-in "emergency brake": Press the CP break key (normally PA1) and issue the command

```
smsg * stop
```

without any trailing blanks. This will bypass the normal command processing routine. However, as this is not an orderly way to stop PIPESERV, it may cause pipelines to stall, and it may generate some pages of error messages. The CP settings, altered by PIPESERV at initialisation, may not be reset.

TIMER



Note:
1 This keyword is only valid for the *timer* command.

The *timer* and *reminder* commands request one or more actions to take place at the specified time, either once or repeatedly. *reminder* and *tmr* are synonyms for *timer*, except that the REMINDER option is enforced for *reminder*. This permits general users to take advantage of the timing facilities without allowing them to schedule command execution.

The *timer* and its synonyms also let you query and delete timer requests.

Creating a timer request

The `time` and `date` specifications are positional. If specified, the STRING keyword must be the last, as all text following this keyword will be interpreted as variable text. All other keywords may be specified in any order. If a keyword is specified more than once, or if more than one keyword from a choice group are specified, the last occurrance will take priority.

Time keywords: The `timespec` group is mandatory and must be specified with no intervening blanks. *hours*, *minutes*, and *seconds* must be whole, non-negative numbers if specified. If one or more variables are not specified at all, they will default to 0. Any combination of hours, minutes and seconds is valid. Minutes and seconds do not have to follow normal conventions of being 60 or less. The total value must greater than or equal to 0, and less than or equal to 24 hours (= 1440 minutes = 86400 seconds) in general. A relative time specification, prefixed with a plus sign (+), must be greater than 0. The specification for the PAST keyword must be less than or equal to 1 hour (= 60 minutes = 3600 seconds).

The following are all examples of valid time specifications:

2	02:00:00 = 2 hours.
:45	00:45:00 = 45 minutes.
::10000	00:00:10000 = 10000 seconds = 02:46:40 = 2 hours, 46 minutes and 40 seconds.
:	0.
4::3	04:00:03 = 4 hours and 3 seconds.
3:134	03:134 = 3 hours and 134 minutes = 5 hours and 14 minutes.
:1:80	00:01:80 = 1 minute and 80 seconds = 2 minutes and 20 seconds.

set:

```
                              ┌───────────────────────────────────────────┐
├──┤ time ├─┬────────────┬──▼─┬──────────────────────────────────────────┬─┴───────────────►
            └─┤ date ├───┘    ├─REPeat───────────────────────────────────┤
                              ├─NOdeid──nodeid───────────────────────────┤
                              ├─RAnge──┤ timespec ├──┤ timespec ├────────┤
                              │ ┌─CMd─(1, 2)─────┐                       │
                              ├─┼────────────────┼───────────────────────┤
                              │ ├─REMinder─(2)───┤                       │
                              │ └─STREam─(1)─────┘                       │
                              │ ┌─TEMporary─(3, 4)─┐                     │
                              └─┼──────────────────┼─────────────────────┘
                                ├─Volatile─(4)─────┤
                                └─PErmanent─(1, 3)─┘

►──┬──────────────────────┬─────────────────────────────────────────────────────────────────┤
   └─STRIng──string───────┘
```

time:

```
├──┬─┤ timespec ├────────────────────────────────────────────────────────┬─────────────────┤
   ├─ + ─┤ timespec ├────────────────────────────────────────────────────┤
   ├─┬─Ipl───────────────────────────────────────┬─┬──────────────────┬──┤
   │ ├─REStart───────────────────────────────────┤ └─ + ─┤ timespec ├──┘  │
   │ └─STArt─┬────────┬─┬────────────┬──────────┘                         │
   │         └─NIpl───┘ └─NRestart───┘                                    │
   └─PAst──┤ timespec ├──────────────────────────────────────────────────┘
```

timespec:

```
├──┬─hours──┬────────────────────────────────────────────┬──┬──────────────────────────┤
   │        └─:─┬─────────────┬─┬──────────────────────┬─┘  │
   │            └─minutes─────┘ └─:─┬──────────────┬───┘    │
   │                                └─seconds──────┘        │
   └─:─┬─────────────┬─┬──────────────────────┬─────────────┘
       └─minutes─────┘ └─:─┬──────────────┬───┘
                           └─seconds──────┘
```

date:

```
   ┌─Anyday──────┐
├──┼─────────────┼─────────────────────────────────────────────────────────────────────────┤
   ├─Monday──────┤
   ├─TUesday─────┤
   ├─WEDnesday───┤
   ├─THursday────┤
   ├─Friday──────┤
   ├─SAturday────┤
   ├─SUnday──────┤
   ├─WDay────────┤
   ├─WEEkend─────┤
   ├─dd/mm───────┤
   └─dd/*────────┘
```

Notes:

1 This keyword is only valid for the *timer* command.

2 For the *reminder* command, the REMINDER keyword is default.

3 For commands specified in the PIPESERV INITCMD file, the PERMANENT keyword is default.

4 For commands entered via the *cmd* input stream, or via the CMS stack, the VOLATILE keyword is default.

A `timespec` alone denotes an **absolute time** of day in the 24 hour clock in the local timezone.

A `+timespec` specification denotes a **relative time**, or a **time offset** from the time of the command entry. There must be no blanks between the plus sign and the time specification.

The IPL keyword specifies the request to expire at the first following **system ipl**. If the IPL keyword is specified in a command loaded from either the `PIPESERV INITCMD` file or the CMS stack, the request will expire at PIPESERV initialisation if this happens at a system ipl. If a time offset is specified, the offset is calculated with the time stamp resulting from a `cp q cplevel` command used as base.

The RESTART keyword specifies the request to expire at the first following PIPESERV restart, unless the next start of PIPESERV is not a restart. In that case, the request will be cancelled.

The START keyword specifies the request to expire at the first following PIPESERV start, including start at system ipl and restart. If the NIPL keyword is also specified, start at system ipl will be ignored. If the NRESTART keyword is also specified, PIPESERV restart will be ignored. If the first start of PIPESERV does not match the type of start specified in the *timer* command, the request will be cancelled.

If a time specification is given after an IPL, RESTART, or START keyword, the expiry of the request will be offset with the specified time.

The PAST keyword specifies that the request expires the specified time after the first full hour. The time specification must be less than or equivalent to 1 hour.

If a range is specified, it is an error to specify a request to expire at an absolute time of day if the time is outside the range specified.

It is an error to specify the VOLATILE keyword on the same command as an IPL, RESTART, or START keyword, unless the command is entered via the `PIPESERV INITCMD` file or the CMS stack. Such a command would only exist in PIPESERV's virtual storage, and it would therefore have been cleared at the next start of PIPESERV.

It is an error to specify IPL, START, or RESTART in connection with any other date keywords than ANYDAY

Date keywords: The `date` group is optional. If specified, it must follow immediately after the time specification.

The default keyword is ANYDAY, denoting that the request may expire on any day of the week.

The keywords MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, and SUNDAY indicates that the request can only expire on the specified day of the week.

The keyword WDAY indicates that the request can only expire on a weekday, that is, Monday-Friday.

The keyword WEEKEND indicates that the request can only expire on a day in a weekend, that is, Saturday-Sunday.

A date specified in the format `dd/mm` indicates that the request can expire only on the specified date, where `dd` denotes the day of the month (1-31), and `mm` denotes the calendar month (1-12). dd and mm may be specified with one or two digits, with or without leading zeroes.

A date specified in the format dd/* indicates that the request can expire only on the specified day of the month, but in any month. dd denotes the day of the month (1-31). dd may be specified with one or two digits, with or without leading zeroes.

It is an error to specify any other date keywords than ANYDAY in connection with the IPL, START, or RESTART keywords.

Repeat keyword: Unless you specify the REPEAT keyword, the request will expire only once, and then be deleted from PIPESERV's storage and control files.

If you specify the REPEAT keyword, the request will remain active until manually deleted with a `timer delete` command. Permanent requests must be manually deleted from the `PIPESERV INITCMD` file, and a *rlddata* command must then be issued to refresh PIPESERV's storage and control files.

Nodeid keyword: The NODEID is designed for use in the `PIPESERV INITCMD` file, which can then be shared between multiple VM systems. Thus, you can avoid having a different file for each system.

If the nodeid specified after the NODEID keyword matches the RSCS nodeid where PIPESERV is running, the keyword is simply ignored. Otherwise, the *timer* command is ignored, and an informational message is typed on the console.

Range keyword: If a RANGE keyword is specified, the expiry of the request will be ignored if the actual time of day in the local timezone is outside the range specified.

It is an error to specify a request to expire at an absolute time of day, which is outside the range specified.

A specified range applies strictly to the time of the actual expiry of the request, and not to the requested expiry. For example, a request for 3:00:00, delayed until 3:10:00 because PIPESERV was busy or out of service, would not be executed if a range of 2:00:00 to 3:00:00 was specified.

Action keywords: The expiry of a request may result in different actions being taken. The default action depends if the request was specified using the *timer* or the *reminder* command.

For the *reminder* command, only the REMINDER keyword is valid, and it is the default.

For the *timer* command, the CMD keyword is the default.

The REMINDER keyword requests the character string following the STRING keyword to be sent to the userid that issued the *timer* or *reminder* command at the expiry. It will be sent as a CP MESSAGE or MSGNOH, depending on the configuration file setting. If requested by the userid running PIPESERV, the string will be typed on the console.

The CMD keyword requests the character string following the STRING keyword to be executed as a host command or a PIPESERV command at expiry. The console output from the command will be sent to the userid that issued the *timer* or *reminder*. It will be sent as CP MESSAGES or MSGNOH, depending on the configuration file setting. If requested by the userid running PIPESERV, the command output will be typed on the console. A command authorisation check will be done before executing the command. If the userid is not authorised for the command in the `PIPESERV AUTHUSER` file, the command will be rejected.

You may add pipelines permanently or temporarily to the running pipeline set by using the *rexx* PIPESERV command.

For example, to run a temporary pipeline, you could specify a string of

```
rexx 'callpipe literal Allo monde | console'
```

This would type out the string `Allo monde` on the console. PIPESERV does not regain control until the `callpipe` command has completed.

To permanently add a pipeline to run in parallel with PIPESERV and other active pipelines, you could specify a string of

```
rexx 'addpipe starsys *account | >> account file a'
```

This would permanently add a pipeline to read account records from CP and write them to a file. PIPESERV will regain control immediately, and the new pipeline will execute in parallel with PIPESERV.

The STREAM keyword requests the character string following the STRING keyword to be written as the final part of a record on the *tmr* output stream at expiry. If the *tmr* is not connected, the command is rejected.

String keyword: The character string following the STRING keyword is used for either a command, a reminder text, or a record on the *tmr* output stream, as described under the CMD, REMINDER, and STREAM keywords. It is optional to specify a string. If a string is specified, the string begins after one blank after the end of the STRING keyword. Trailing blanks are conserved. There is no limit to the length of the string.

Request types: Three different types of requests are possible:

Volatile A volatile request is identified by the VOLATILE keyword. It exists only in virtual storage. If PIPESERV is stopped or restarted before the requested action takes place, the request will disappear, and the action will not take place. VOLATILE is the default for commands entered via the *cmd* input stream and the CMS stack.

Temporary A temporary request will survive from one invocation of PIPESERV to the next, until the requested action has taken place. TEMPORARY is the default for commands entered on the CMS command line, via MSG or SMSG, or via the synchronous IUCV command interface.

It is an error to specify the VOLATILE keyword on the same command as an IPL, RESTART, or START keyword, unless the command is entered via the `PIPESERV INITCMD` file or the CMS stack. Such a command would only exist in PIPESERV's virtual storage, and it would therefore have been cleared at the next start of PIPESERV.

Permanent A permanent request is handled like a temporary request by PIPESERV, except that whenever PIPESERV is started or restarted, or an *rlddata* command is issued, all permanent requests are deleted from PIPESERV's virtual storage and then reloaded from the `PIPESERV INITCMD` file. A permanent request can only be deleted by removing the corresponding command from the `PIPESERV INITCMD` file and reloading this file. A `timer delete` command will be rejected. PERMANENT is the default for commands entered via the `PIPESERV INITCMD` file.

If you have updated the `PIPESERV INITCMD` file with a new timer request and you want to wait reloading the file, you can you use the PERMANENT keyword on a *timer* command to create the request in PIPESERV, until the file is reloaded.

Note: If there is not enough space on the minidisk or directory accessed as A for PIPESERV to update its control files, a *timer* or *reminder* command will be rejected unless volatile.

Querying a timer request

Use the QUERY keyword to query the details about pending timer requests. `timer query` displays either a brief summary, using one line per request, or the complete information, using the necessary number of lines. The keyword to display the brief information is BRIEF, which is the default. Specify the ALL keyword to display all details.

tmr and *reminder* are synonyms for *timer*, but with the important difference that the *reminder* command may only display information about requests entered by the userid issuing the `reminder query` command, while *timer* and *tmr* may display information about any request. Thus, by authorising users to either the *timer* command or the *reminder* command, you may separate privileged system users from non-privileged end users.

```
 tmr query brief
►Requestno. Userid   Nodeid   Time         Date  Rep A T String (A=Actn T=Type)
►---------- -------- -------- --- -------- ----- --- - - ----------------------
►0000000002 9SKOVGF  GFORD1   Abs 15:00:00 Any   Rep R T 15 o'clock reminder
►0000000003 9SKOVGF  GFORD1   Abs 10:00:00 Wed   Rep R T Today is Wednesday (10
►0000000014 9SKOVGF  GFORD1   Abs 15:03:00 06/*  Rep R T This is a 15:03 remind
►0000000016 9SKOVGF  GFORD1   Pas 00:30:00 Wed   Rep R T Past 30 on Wednesday
►0000000021 9SKOVGF  GFORD1   Pas 00:15:00 M-F   Rep R T Past 15 on a weekday
►0000000022 9SKOVGF  GFORD1   Abs 10:00:00 01/09 Onc R T Requested for 1/9 at 1
►0000000023 9SKOVGF  GFORD1   Rel 00:33:00 M-F   Rep R T 33 mins interval on a
```

Figure 10. Sample output from a timer query brief command.

"Requestno." is a unique request identifier assigned by PIPESERV.

"Userid" and "Nodeid" refer to the requesting userid and the RSCS nodeid of the user respectively.

The information under the "Time" heading shows the kind of time and the time itself:

Abs	Absolute time
Rel	Relative time
Pas	Past time
Sta	A start type, comprising ipl, restart and start.

The information under the "Date" heading shows the date parameter. The weekdays are abbreviated to three characters. Wday is displayed as "M-F" (Monday-Friday), and Weekend is displayed as "S-S" (Saturday-Sunday).

The information under the "Rep" heading shows if the request is to expire once ("Onc") or repeatedly ("Rep").

The information under the "A" heading shows the action to be taken at expiry:

C	CMD
R	Reminder
S	Stream

The information under the "T" heading shows the type of request:

T	Temporary
V	Volatile
P	Permanent

Figure 11 illustrates the ALL keyword.

```
 tmr query reqno 23 all
►
►Request number: 0000000023 Userid: 9SKOVGF  Nodeid: GFORD1   Type  : Temporary
► Entry        : 19940831 12:57:26          Time  : 00:33:00 Action: Reminder
► Last executed: 19940926 11:26:11          Time  : Relative
► Range        : 00:00:00 24:00:00
► Date         : On weekday repeat
► String: 33 mins interval on a weekday
```

Figure 11. Sample output from a timer query all command.

"Entry" shows when the request was entered to PIPESERV.

"Last executed" shows when the request was last executed.

"Range" shows the specified or default time range.

"String" shows the full string.

The REQNO keyword limits the query command to the specified request number. If you are not authorised to query the specified request number, the command is rejected. If a request does not exist for the specified number, an error message is typed.

The USERID and NODEID keywords limit the query command to search for requests entered by the specified userid on the specified RSCS nodeid.

Deleting a timer request

Use the DELETE keyword to delete pending timer requests before they expire.

tmr and *reminder* are synonyms for *timer*, but with the important difference that the *reminder* command may only delete requests entered by the userid issuing the `reminder delete` command, while *timer* and *tmr* may delete any request. Thus, by authorising users to either the *timer* command or the *reminder* command, you may separate privileged system users from non-privileged end users.

You must specify the number of the request to delete after the DELETE keyword. If you are not authorised to delete the specified request number, the command is rejected. If a request does not exist for the specified number, an error message is typed.

The keyword CANCEL is a synonym for DELETE.

Querying the format of a timer stream output record

A *timer* command with the FORMAT keyword specified will display a record for each *field* specified, containing the field name, start position of the field in the *tmr* output stream record, and field length, separated by blanks. The length is not specified for the field named STRI, since it is variable and unlimited.

The purpose of this function is to allow your application to dynamically identify the fields instead of hardcoding the offsets. If the record layout is updated in future versions, your application need no changes to adapt.

Please refer to “tmr output stream” on page 47 for a complete list of valid fields.

Figure 12 illustrates the FORMAT keyword.

```
 timer format origuser orignode stri
►ORIGUSER 77 8
►ORIGNODE 85 8
►STRI 125
►Ready(0); 2000-05-20 16:48:02 (SKOVGAF at GFORD1)
 tmr format
►HEADER 1 24
►REQNO 25 10
►FMTCODE 35 2
►ENTRBASE 37 7
►ENTRDATE 44 8
►ENTRTIME 52 5
►LASTBASE 57 7
►LASTDATE 64 8
►LASTTIME 72 5
►ORIGUSER 77 8
►ORIGNODE 85 8
►TIMETYPE 93 1
►TIMEIPL 94 1
►TIMEREST 95 1
►TIMETIME 96 5
►DATEDATE 101 4
►DATETYPE 105 1
►DATEWDAY 106 1
►DATEREPT 107 1
►DELAY 108 5
►RANGE1 113 5
►RANGE2 118 5
►ACTION 123 1
►TYPE 124 1
►STRI 125
```

Figure 12. Sample output from timer format commands.

If you do not specify any fields, all fields are displayed, starting with a field named HEADER. This covers the common PIPESERV output record prefix, as described in “tmr output stream” on page 47.

Programming interface

After having familiarised yourself with PIPESERV, you may want to take further advantage of the facilities offered by PIPESERV. These include a number of streams that your pipeline may connect to.

When coding applications, do not forget that the host command may identify the userid that requested the execution by inspecting the variables `ORIGUSER` and `ORIGNODE` in GLOBALV group PIPESERV. These variables contain the userid and RSCS nodeid respectively, and for the duration of the command execution.

CMS stack and PIPESERV INITCMD file

When PIPESERV is started, it will look for a PIPESERV INITCMD file. If that file exists on any accessed filemode, the commands therein are executed. Then all records - if any - from the CMS stack are read and executed as commands. When the stack is empty, commands entered on the CMS command line are accepted.

This feature is useful if you want certain commands executed immediately after PIPESERV start, with PIPESERV fully running.

The *timer* command would be a good candidate for stacking. This is demonstrated in the example “Collecting EREP, Account and Symptom records” on page 37.

Input and Output Streams

Figure 13 on page 31 illustrates the possible connections to the PIPESERV stage. Note that the streams are not numbered as for example first output stream, secondary output stream and so on, but instead marked with a streamid. It will be illustrated in examples how you connect to them. Note that pipeline stream identifiers are case sensitive. All the identifiers used to connect to PIPESERV are in lowercase.

The record formats are explained in “Input & output stream record formats” on page 45.

Log input stream

If you connect to the *log* input stream, your program may generate log records in addition to the PIPESERV generated log records.

Command input stream

If you connect to the *cmd* input stream, your program may generate commands for execution, just as if they were entered from the terminal.

Alternate command input stream

If you connect to the *cmdt* input stream, your program may generate commands for execution, just as if they were entered from the terminal.

If the *cmdt* output stream is connected, then the command output is written to that stream only; not to the console. In that case, one record only will be produced for each input record, containing the original command, the console output, and the PIPESERV ready message, separated by x'15' characters.

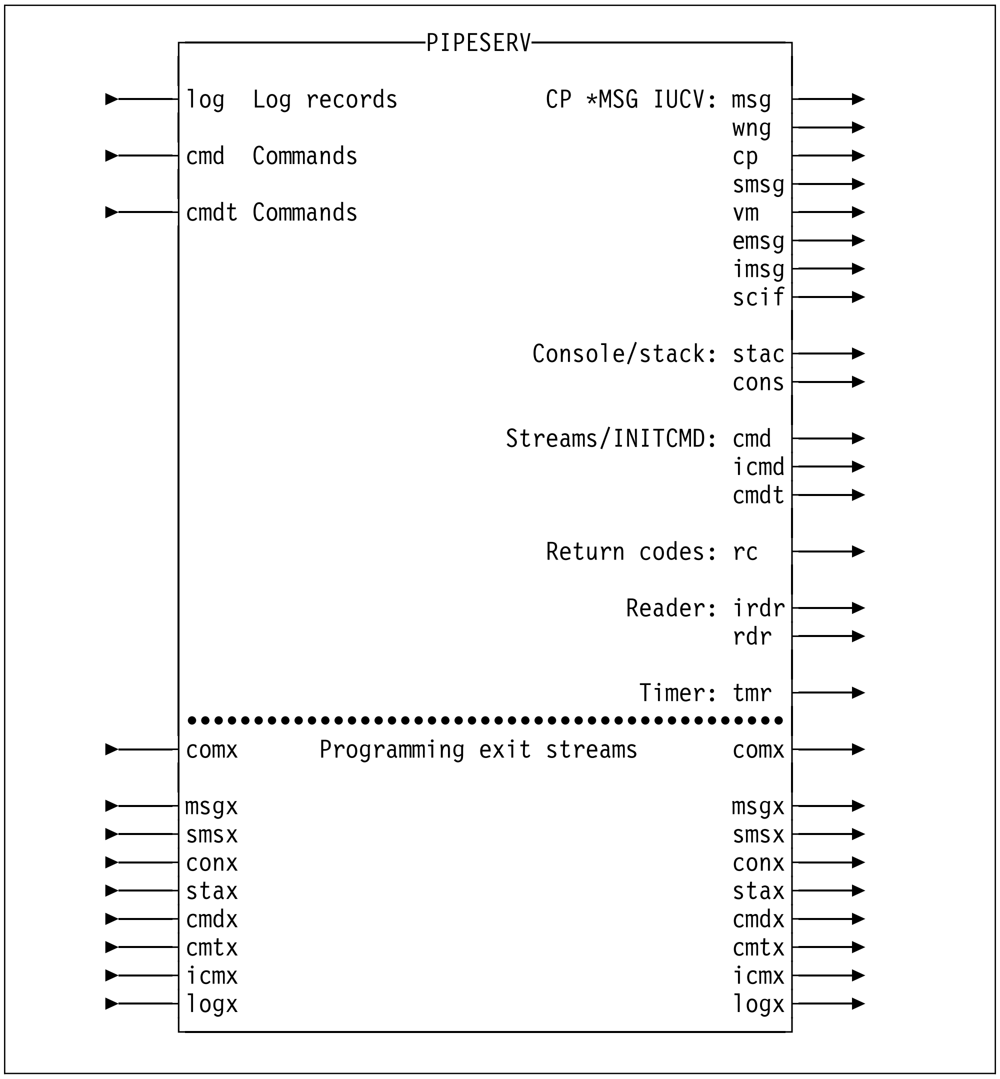


Figure 13. The possible connections to the PIPESERV stage

If the *cmdt* output stream is not connected, then the command output is written to the console in the usual way.

If the *rc* and the *cmdt* output streams are both connected, then output records on the *cmdt* and *rc* output streams, originating from a record on the *cmdt* input stream, are synchronised, so that the record is written to the *rc* stream first.

CP *MSG IUCV output streams

If you connect to one or more of the CP *MSG IUCV output streams, your program has access to the records from the CP *MSG service as they arrive. Each stream corresponds to an IUCV class. Refer to the description of the *starmsg* stage in the pipeline reference manual for a description of each class.

Stack output stream

If you connect to the *stac* output stream, your program has access to all records that were in the CMS stack when PIPESERV was started.

Console output stream

If you connect to the *cons* output stream, your program has access to all commands entered on the console.

Reader file output streams

If you connect to the reader file output streams, your program will get information about reader files. The *irdr* stream gives information about the reader files available when PIPESERV is started. The *rdr* stream gives information about reader files as they arrive.

Timer output stream

When a timer request that specified the STREAM keyword expires, a record is written to the *tmr* output stream.

Command output stream

The *cmd* output stream gives you access to the records written to the *cmd* input stream.

Alternate command output stream

The *cmdt* output stream gives you access to the records written to the *cmdt* input stream, console output from the commands, and the PIPESERV ready message. See “Alternate command input stream” on page 30 for a detailed explanation of the interaction with the *cmdt* input stream and the *rc* output stream.

rc output stream

If connected, one record containing the return code is written for each command executed. See “Alternate command input stream” on page 30 for a detailed explanation of the interaction with the *cmdt* input stream and output streams.

icmd output stream

The *icmd* output stream gives you access to the records from the PIPESERV INITCMD file when PIPESERV is started.

Pre-processing exits

To control command execution in PIPESERV, it lets you modify and/or filter commands from most sources before execution. The log records may be filtered or modified before being written to the log file.

If doing this, you must conform to the internal record format, which is described in “Pre-processing exit stream record formats” on page 51.

Also, please note that logging takes place before the exit point, except for the logging exit itself.

Commands resulting from the expiry of a timer that specified the CMD keyword only passes through the logging exit, and so do commands from the synchronous IUCV command interface. Such commands cannot be changed before execution.

The following list describes the purpose of each exit:

comx	All commands except those explicitely excepted above pass through this exit.
msgx	All commands from CP messages (MSG or MSGNOH) pass through this exit, including messages arriving via RSCS.
smsx	All commands from special messages (SMSG) pass through this exit.
conx	All commands entered on the server console pass through this exit.
stax	All commands entered via the CMS stack pass through this exit.
cmdx	All commands entered via the *cmd* input stream pass through this exit.
cmtx	All commands entered via the *cmdt* input stream pass through this exit.
icmx	All commands entered via the PIPESERV INITCMD file pass through this exit.
logx	Logging records ready to be written to the PIPESERV log file may be freely filtered or modified through this exit.

Refer to “Implementing your own application commands” on page 35 for an example of how to use exits.

Synchronous IUCV command interface

The following describes the syntax for the synchronous command interface.

```
►►──iucvclient──serverid──name──cmd──────────────────────────────►◄
```

iucvclient	The name of the pipeline stage that activates a synchronous IUCV communication.
serverid	The userid of the server running PIPESERV.
name cmd	Identifies the IUCV service requested from the server. Only the service `cmd` is supported by PIPESERV. Do not confuse the service with the *cmd* PIPESERV command.

The host or PIPESERV command that you want the server to execute must be input on the primary stream to the *iucvclient* stage.

The command output is written to the primary output stream.

Only one command may be sent into the IUCV interface at a time. This is a restriction in the PIPESERV design. The restriction also applies to multiple commands separated by the PIPESERV linend character. Any commands following the first will be ignored.

Included with PIPESERV is a sample program called `PIPESERV EXEC` that demonstrates how you can use the IUCV interface with a timeout and recover the return or error code.

Refer to “Use of the synchronous IUCV interface in the server” on page 42 for further guidance.

Programming Guide

This chapter gives some practical examples and advice of how to utilise the PIPESERV features.

Using HI and HX

When developing applications under PIPESERV, you will sooner or later want to stop a REXX program because it is looping or doing something else you don't want. To do that, enter the CMS immediate command HI (Halt Interpretation), prefixed with the CP terminal lineend character, for example

```
#HI
```

If you are running PIPESERV in IMMCMD mode, leave out the CP terminal lineend character. See “PIPESERV syntax” on page 13 and “Managing console input” on page 15 for further information about IMMCMD mode.

PIPESERV will continue executing, but note that if you use the HX command (Halt eXecution), PIPESERV will be halted as well.

If your program keeps placing you in VM READ, the old trick is to place the cursor at the first position in the command input field, then press the "left arrow" key once, then press ENTER, and then type your immediate command and press ENTER again. This will bypass the VM READ condition, where the input is trapped by the program.

Stopping your application when PIPESERV stops

This could be more tricky than you would think at first glance. But remember that your application is an independent program that may or may not know about the existence of PIPESERV.

Nevertheless, when PIPESERV stops, you may well want your application to detect this and stop as well.

The traditional way of closing down pipeline stages is to reflect EOF on a stream. If your application connects to one or more of PIPESERV's input or output streams, you can detect when they go to EOF and then close your application.

If you don't connect to any PIPESERV streams, you can set up an indefinite timer and connect it to an input stream. When PIPESERV stop, it will issue a *pipestop* command, which will in turn halt all stages waiting for external interrupts. To do this, add the following to the start of your application:

```
'addpipe',
  '| literal +999999999',
  '| delay',
  '| append literal EOF',
  '| *.in.0:'
```

This example connects to your primary input stream, but you can use any existing input stream instead.

When *delay* terminates, EOF will be reflected to *append literal*, which will in turn produce one record and present it to your input stream.

Implementing your own application commands

When implementing an application with PIPESERV as the server, it is likely that you want your application to process commands in some way. Since PIPESERV is already connected to the CP *MSG IUCV service and other possible sources for commands, your application would not be able to connect directly to these sources. PIPESERV offers a variety of ways for your application to get access to incoming commands.

1. You can connect your application to one of the PIPESERV output streams described in “Input & output stream record formats” on page 45. Thus, you can choose the sources from which you are willing to accept commands: CP messages, CP warnings, CP special messages, the CMS stack, the console, the *cmd* input stream, and reader files.

   But note that what is written to the output streams is also processed by PIPESERV's command processor. So you need a way of separating the commands.

   a. One way of doing this is to define your application commands to begin with the asterisk character (*). Such a command will be ignored by CMS and PIPESERV, but you can process it in your application. If you want this format to be transparent to the clients, you can provide an interface program to add the asterisk to the command in the client environment.

   b. Another way is to create dummy EXEC's having the names of your application commands. For example, if you want your application to process a command named `xyzstart`, create an empty XYZSTART EXEC to avoid error messages from PIPESERV's command processor. Then create a table of valid application commands and use a *lookup* to filter out non-application commands.

2. Alternatively, you can connect your application to one of the provided pre-processing exits described in “Pre-processing exits” on page 32. You may find that the COMX exit is the most useful, since all commands pass through it before being presented to the PIPESERV command processor. Using this exit, you can use an application command table and *lookup* to filter out your application commands and forward the remaining commands to PIPESERV's command processor. Figure 14 shows you how to do that. A similar method can be used for the other exits.

```
/* Sample application */
address command
'PIPE (end \)',
  '| f: faninany',                   /* Feedback to exit           */
  '| piplbl.comx: pipeserv',         /* Run the server itself      */
  '| l: lookup substr w1 of 25-* w1 detail', /* Valid appl. command?  */
  '| yourappl',                      /* Feed to application        */
  '\ disk VALID COMMANDS *',         /* Read list of appl. commands */
  '| l:',                            /* Feed to lookup             */
  '| f:'                             /* Feedback to exit           */
```

Figure 14. Use the comx exit to filter application commands */

Serial connections to PIPESERV

The simplest way to utilise the PIPESERV programming features is to connect to one or more streams and use the available information.

When doing so, you must ensure that your application consumes output records from PIPESERV in time for other parts of your application to continue running. While waiting for your application to consume an output record, part or all of PIPESERV may be unable to continue.

Often, good coding practice for pipeline stages will be a good choice. According to this practice, your stage runs in a `peekto - output - readto` loop. This prevents your stage from delaying the records. This will in most cases be suitable for applications of a synchronous nature.

In some cases, a `readto - output` loop will be the right choice. For example, if your pipeline stage starts a fullscreen application that waits for operator input, you probably do not want PIPESERV operation to be suspended until the operator presses an attention key.

Thus, when choosing your reading method, the question you should ask yourself is: "Do I want PIPESERV to continue running and accepting input from the console and other sources while my application is processing output?"

If you fail to consume PIPESERV output records, you may cause your application and/or PIPESERV to stall. Particular care is required when connecting to the *cmdt* and *rc* output streams at the same time. The *rc* output stream produces a record with the return code from all commands executed in PIPESERV. The *cmdt* output stream produces an output record for each command from the *cmdt* input stream. PIPESERV first writes the record to the *rc* output stream, then waits for it to be consumed, and then writes the output record to the *cmdt* stream. It is important that your application consumes the records in the same order to avoid stalling.

Monitoring reader files

Figure 15 demonstrates how PIPESERV can be used to monitor reader files.

```
/* SHELL EXEC: Monitor reader files */
address command
'PIPE (end \)',
  '| piplbl.rdr: pipeserv',           /* Run the server itself      */
  '| fsr',                            /* Handle reader file records  */
  '\ piplbl.irdr:',                   /* Connect to initial rdr info */
  '| irdrmsg:tolabel *' ||,           /* Re-direct last record      */
  '| fsr',                            /* Handle the records         */
  '\ irdrmsg:',                       /* Get the last record        */
  '| spec /SHELL: End of initial reader file processing/ 1',
  '| console'                         /* Tell user when ready       */
```

Figure 15. SHELL EXEC: Monitor reader files

`fsr` is a user-written stage to decide how to handle different kind of reader files, for example purging based on origin and file contents.

The first `fsr` stage is connected to PIPESERV's *rdr* output stream, which generates a record for each reader file arriving while PIPESERV is running.

The second `fsr` stage is connected to PIPESERV's *irdr* output stream, which generates a record for each reader file existing when PIPESERV is started. The last record on this output stream contains `*`. This makes the pipeline able to determine when all initial reader files have been handled and to display a message.

Figure 16 illustrates the resulting pipeline topology.

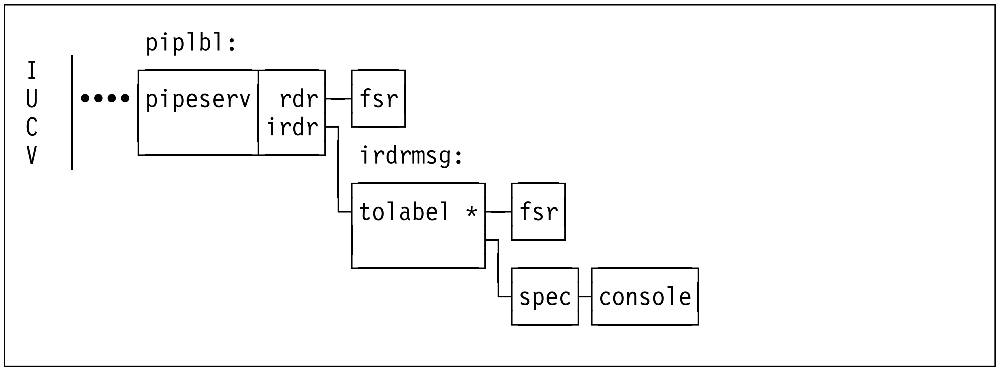


Figure 16. The pipeline topology resulting from SHELL EXEC

Note that `piplbl` is a pipeline label. `rdr` and `irdr` are stream identifiers. The stream identifier is necessary to tell PIPESERV which streams you are connecting to. You cannot rely on a PIPESERV stream having a specific stream number, even though you might find out by experimenting. The stream numbers for PIPESERV are unspecified, and they may change without notice. Furthermore, the stream numbers depend on how many streams you connect to.

Running PIPESERV in parallel with other pipeline stages

In addition to connecting pipeline stages to PIPESERV, you may run stages in parallel with PIPESERV. That is, you do not have to connect your stages to PIPESERV at all.

Collecting EREP, Account and Symptom records

Figure 17 on page 38 illustrates how to collect CP account, symptom and EREP records in a virtual machine, using PIPESERV as the driver.

```
/* SYSRECS REXX: Collect account, symptom & EREP records */
'commit 0'                          /* Make sure data can flow     */
queue 'tmr 24 repeat cmd string acnt all' /* Close acnt every midnight*/
'addpipe pipeserv'                  /* Run pipeline server         */
'addpipe',                          /* Collect Account records     */
  '| starsys *account',
  '| >> account file a'
'addpipe',                          /* Collect EREP records        */
  '| starsys *logrec',
  '| >> erep file a'
'addpipe',                          /* Collect SYMPTOM records     */
  '| starsys *symptom',
  '| >> symptom file a'
```

Figure 17. SYSRECS REXX: Collect account, symptom & EREP records

To invoke SYSRECS REXX, you could simply use the command `pipe sysrecs`. Every day at midnight, *delay* will release a record, which is changed into an SMSG command. That command ensures that accounting is closed. Other commands could be useful. For example a command to process the accumulated records.

Figure 18 illustrates the resulting pipeline topology.

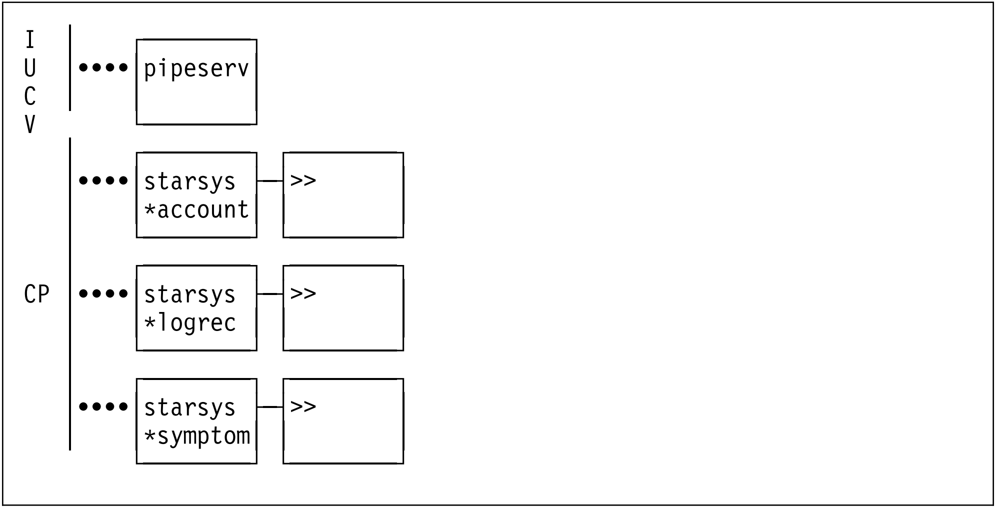


Figure 18. The pipeline topology resulting from SYSRECS REXX

For those not familiar with ADDPIPE, Figure 19 on page 39 shows how to do the job with only one PIPE command.

```
/* SYSRECS EXEC: Alternative to SYSRECS REXX */
address command
queue 'tmr 24 repeat cmd string acnt all' /* Close acnt every midnight*/
'PIPE (end \)',
  '| pipeserv',                        /* Run pipeline server        */
  '\ starsys *account',
  '| >> account file a',
  '\ starsys *logrec',
  '| >> erep file a',
  '\ starsys *symptom',
  '| >> symptom file a'
```

Figure 19. SYSRECS EXEC: Alternative to SYSRECS REXX

Now, suppose that you wanted to take advantage of the *cmd* input stream instead of queueing a command. You would then connect the output from *spec* to PIPESERV, as illustrated in Figure 20:

```
/* SYSRECS EXEC: Coding with connection to PIPESERV */
address command
'PIPE (end \)',
  '| starsys *account',
  '| >> account file a',
  '\ starsys *logrec',
  '| >> erep file a',
  '\ starsys *symptom',
  '| >> symptom file a',
  '\ literal tmr 24 repeat cmd string acnt all',
  '| piplbl.cmd: pipeserv'
```

Figure 20. SYSRECS EXEC: Coding with connection to PIPESERV

The resulting topology is illustrated in Figure 21.

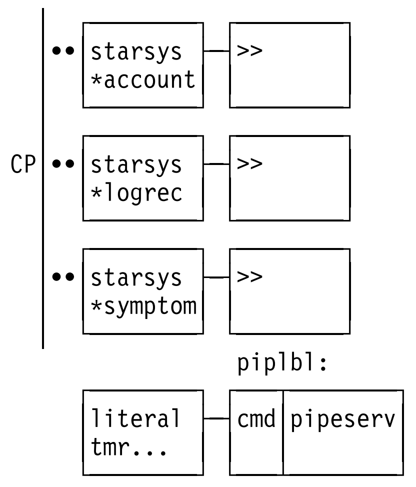


Figure 21. Topology when connecting to PIPESERV's command input

Real-life example of collecting account records

The critical reader will by now have noticed that the previous example of collecting account records suffer a major inconvenience: The records are not separated into different files for each date. This will prove impractical if you want to process your account data.

The following example has been taken from real-life collection of account data. This solution is plug-compatible with the `RETRIEVE` module.

To use this solution, replace the `RETRIEVE` command with a `PIPE SYSRECS` command. Remember to remove any instructions to force the accounting userid at midnight, as this may result in a loss of account records that are waiting in the I/O buffer to be written to the account file. To avoid loss of records at shutdown, authorise a userid to issue the CP `ACNT ALL` command, followed by a *stop* command.

Figure 23 illustrates the pipeline topology. The pipeline inside `SYSRECS` is rebuilt every midnight to reflect the new day's account file. The SUSPEND stage is executed to allow PIPESERV to complete its own midnight processing before SYSRECS continues. PIPESERV has to close the current log file and create a pipeline stage to write the new day's log file. If the SUSPEND stage is omitted, log records may end up in the wrong log file. The PIPESERV stage is executing uninterrupted past the midnight boundaries. A *stop* command will result in PIPESERV executing a `PIPMOD STOP` command, which in turn will terminate the *starsys* stage. The complete pipeline set will thereby be terminated. The `midnight?` flag will contain 0, if the pipeline set is terminated by PIPESERV, and this will tell `SYSRECS` not to rebuild another account record processing pipeline.

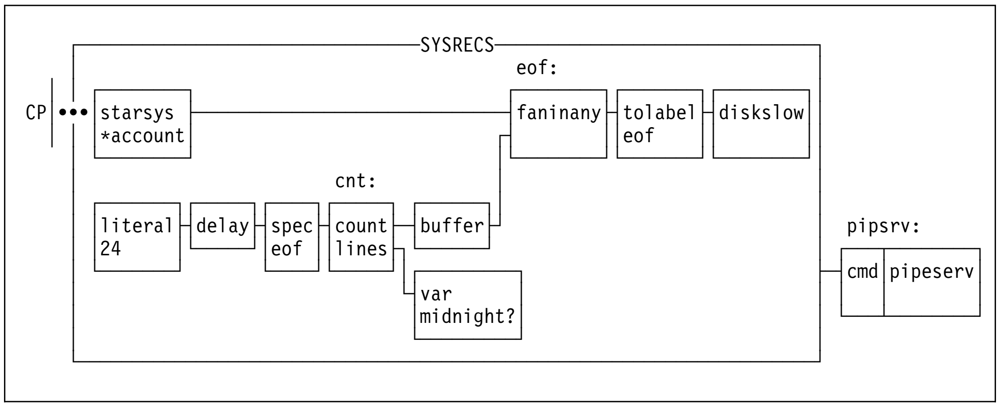


Figure 23. Topology of the real-life accounting solution

Writing your own stages for parallel processing

Of course, you may write your own stages in REXX to run in parallel with PIPESERV. Note especially the SUSPEND pipeline command. If your REXX program is performing heavy processing between calls to pipeline like READTO and OUTPUT, PIPESERV will not get a chance to process input. But if your program issues the SUSPEND pipeline command at regular intervals, the pipeline dispatcher will give PIPESERV and other stages a chance to execute.

```
/*-----------------------  SYSRECS  REXX      --------------------------

  (c) Copyright International Business Machines Corporation 1994, 1999.
                          All Rights Reserved.

Function:       Read system records from CP and write them to a file.
                Change the output file at midnight.
How executed:   As a pipeline filter
Syntax:         PIPE SYSRECS
Disk output:    ACCOUNT  $Ammddyy A
Output streams: cmd (created dynamically and connected to PIPESERV)

------------------------------------------------------------------------
Written by : Finn Skovgaard                               Date: 19940509
Release:     1.0            Fix Level: 0
----------------------------------------------------------------------*/
close = 10                             /* Close for each 10  records  */
eof = copies('ff'x,8)                  /* EOF indicator               */
midnight? = 1                          /* Initialise midnight flag    */
'addstream output cmd'                 /* Create a command stream     */
'addpipe *.out.cmd:|pipsrv.cmd:pipsrv' /* Start the PIPESERV driver   */
do forever                             /* One iteration per day       */
  'streamstate output cmd'             /* PIPESERV still connected?   */
  if rc > 8 | rc < 0 | ¬midnight?      /* If not midnight             */
    then leave                         /* Then get out of here        */
  date = date('u')                     /* mm/dd/yy                    */
  'callpipe',
    '| var date',
    '| change "/""',                   /* Remove / from date          */
    '| var date'                       /* mmddyy                      */
  fid = 'ACCOUNT $A'date' A6'          /* Current account file        */
  'callpipe (end \)',
    '| starsys *account',              /* Connect to CP account       */
    '| eof:faninany',                  /* Merge EOF string            */
    '| tolabel 'eof ||,                /* EOF at midnight             */
    '| close:fanout',                  /* Copy records                */
    '| diskslow 'fid' f 80',           /* Write the account records   */
    '\ literal 24',                /* Wake up at midnight         */
    '| delay',
    '| spec /'eof'/ 1',                /* EOF indicator               */
    '| cnt:count lines',
    '| buffer',                        /* Tolabel doesn't consume rec */
    '| eof:',
    '\ cnt:',
    '| var midnight?',
    '\ close:',                    /* Get account records         */
    '| spec 1 1',                      /* Keep just 1 byte            */
    '| join 'close-1,                  /* Make 1 rec to control CLOSE */
    '| spec /FINIS * * A/ 1',          /* Close file                  */
    '| command'
  if midnight?
    then do
      'suspend'                        /*Let PIPESERV complete process*/
      'select output cmd'              /* Prepare for a command       */
      'output cmd disklea'             /* Clean up old files          */
      call diag 8,'SPOOL CONS CLOSE'
    end
end
```

Figure 22. Real-life example of collecting accounting records

Using PIPESERV with an active, parallel XEDIT environment

It is possible to have a hidden, active XEDIT environment in parallel with PIPESERV. To obtain that, first start XEDIT, and then start PIPESERV from the XEDIT command line. You have now at least the following possibilities to access the XEDIT environment:

1. Use the *address xedit* command. This will allow you to interactively direct commands to the XEDIT environment.
2. Use the pipeline *xedit* stage. This will allow you to access the file data in the XEDIT environment directly.
3. Use the pipeline *subcom xedit* stage. This will allow you to direct commands to the XEDIT environment.
4. Use the *rexx address xedit* command. This will allow you to direct commands to the XEDIT environment and get variables stored in PIPESERV's interactive REXX processor. The XEDIT EXTRACT command would be a good candidate for this purpose.
5. Address the XEDIT environment from a REXX program executing as a command under or in parallel with PIPESERV. This could be an XEDIT macro.

Use of the synchronous IUCV interface in the server

For applications running in parallel with PIPESERV, it may sometimes be useful to be able to exchange information with PIPESERV. One way to do this would be to connect to various of PIPESERV's input and output streams or to send CP messages (MSG) or special messages (SMSG) to the server id. But this would involve asynchronous coding, which can be complicated and even unreliable if not the greatest care is taken.

The easiest method is to use the synchronous IUCV interface. The following sample EXEC illustrates how to set up the communication.

```
'commit 0'
'callpipe literal +3 | delay | hole'
'callpipe literal pquery | iucvclient PIPEUSER name cmd | stem tmrq.'
do i = 1 to tmrq.0
  ...
```

Figure 24. Parallel application using the synchronous IUCV interface

`commit` is necessary to ensure that PIPESERV has been started.

The first `callpipe` command allows time for PIPESERV to initialise.

The second `callpipe` command will send the `pquery` command to PIPESERV and store the response in the rexx array `tmrq`. To activate the application, issue the command

```
pipe (end \) pipeserv \ test
```

Note that there are no stream connections between the two stages.

Restrictions

- Do not start other applications using the CP *MSG service while PIPESERV is active. PROP, WAKEUP, VMSERVE, APBOX, TODEVENT, and HMF are all examples of such applications.
- Do not define more than one invocation of PIPESERV, as they would compete about the same resources.
- If you update files on the minidisk where PIPESERV maintains the log file, the changes are not immediately reflected in the CMS file index on the disk, unless you specify `logwrite safe` in the configuration file. This behaviour is determined by the CMS minidisk architecture.

  In this case, if the userid is logged off while running PIPESERV, files and/or file updates may be lost. To prevent this, PIPESERV will detect if the `logoff` command or a valid abbreviation of it is entered. PIPESERV will then stack the `logoff` command and issue a *stop* command to itself. This feature does not depend on the source (for example console or *cmd* input stream) of the `logoff` command. However, if the `logoff` command is prefixed with `cp` or executed from a program, PIPESERV will not detect it.

  Alternatively, a `FINIS * * A` command may be issued by the user or a program. This will ensure rewriting of the CMS file index on the disk up to that point.
- The RAC command interface to RACF uses SMSG. PIPESERV will change a RAC command into a `RACF ( BATCH` command. But PIPESERV cannot trap a RAC command entered from a fullscreen application like XEDIT. If you accidentally enter a RAC command that cannot be trapped by PIPESERV, your userid will hang for at least 60 seconds due to the minimum timeout setting in the RAC interface.
- You must prefix immediate CMS commands with the CP terminal linend character if you do not specify the IMMCMD option on the PIPESERV command.
- While a host command is active, PIPESERV cannot handle timer activities, interrupts, or anything else. These will be held until PIPESERV is active again. While fullscreen applications like XEDIT may be executed from PIPESERV, you should be aware that PIPESERV is inactive until you exit the fullscreen application and see the Ready message.
- If you define a secondary userid for the userid running PIPESERV, some of the IUCV records from the CP *MSG service will be sent to the secondary userid. Thus, they will not be available to PIPESERV or any pipelines connected to PIPESERV's output streams for the affected IUCV classes, which are at least 1 (messages created by a CP MSG or MSGNOH command) and 7 (CP informational messages). This behaviour is determined by the design of CP.
- Execution of host commands that manipulates the timer via REXX is likely to cause problems for PIPESERV and should be avoided.
- Do not manipulate the MSG, IMSG, SMSG, WNG, EMSG, CPCONIO, and VMCONIO CP settings while PIPESERV is active.
- If you execute *rexx* commands that contains pipeline commands like CALLPIPE and ADDPIPE from another userid or via CP MSG, CP SMSG, the *cmdt* input stream, or the synchronous IUCV interface, console output from other executing stages may end up together with the command output instead of on the console. This only applies if

the pipeline commands are executed directly in the rexx environment. If called via a host interface, there will be no interference.

- The PC file transfer program IND$FILE causes PIPESERV to hang and should be avoided.

  If executing a pipeline command like CALLPIPE or ADDPIPE from another userid, via the *cmdt* input stream, or via the synchronous IUCV interface, console output destined for the server may be mixed up with console output from the pipeline command. This is due to the design of CMS Pipelines and PIPESERV. For ADDPIPE commands, it is unpredictable if any console output is written to the server console or to the command origin, because of its asynchronous nature.

- If during PIPESERV startup the minidisk or directory accessed as A has not enough space for PIPESERV to update its control files, PIPESERV will terminate immediately.

- If during operation the minidisk or directory accessed as A has not enough space for PIPESERV to update its control files, *timer* and *reminder* commands will be rejected unless volatile.

- If logging to the filemode specified in the configuration file cannot continue due to lack of space, logging will continue to the console.

- You cannot use the *logx* pre-processing exit if you specify `LOGWRITE OFF` in the PIPESERV configuration file.

Input & output stream record formats

cmd input stream

Free format. The records will be executed as commands and copied to the *cmd* output stream, unless otherwise defined by a pre-processing exit.

cmdt input stream

Free format. The records will be executed as commands, unless otherwise defined by a pre-processing exit.

log input stream

The input records may contain free-format text that you want to place in PIPESERV's log file. PIPESERV will automatically prefix your records with a timestamp, `"User"`, your userid and RSCS nodeid.

CP *MSG IUCV output streams

Columns	Description
1-8	IUCV class in the format `0000000n`, where `n` is an IUCV class (1-8). In the case of a MSG from a userid, delivered by the local RSCS machine, the format will be `RU000001` (Remote User). In the case of a MSG from an RSCS server on a different node, delivered by the local RSCS machine, the format will be `RR000001` (Remote Rscs).
9-16	Origin userid.
17-24	Origin nodeid.
25-*	Text. Note that in case of a remote MSG delivered by the RSCS server, the prefix of `From nodeid(userid):` or `From nodeid:` has been removed by PIPESERV. PIPESERV issues some SMSG'es to itself to manage commands from other userids. Therefore, you may find some records beginning with `*CMD*` in the text part for SMSG (IUCV class 4) records. They are issued whenever a command from another userid has finished execution.

stac output stream

Columns	Description
1-7	Blanks.
8	`"Q"` (short for Queue).

9-16	Origin userid.
17-24	Origin nodeid.
25-*	The stacked record.

cons output stream

Columns	**Description**
1-7	Blanks.
8	`"C"` (short for Console).
9-16	Origin userid.
17-24	Origin nodeid.
25-*	The console input record.

rdr and irdr file output streams

Note: Reader files in SYSHOLD status are ignored, since they cannot be fully processed by PIPESERV.

Columns	**Description**
1-4	Spool file id.
6-13	Origin userid.
15-22	Origin nodeid.
24-31	Filename.
33-40	Filetype.
42-43	Return code from the CMS RDR command.
45	Reader class.
46	`"1"`: File has been transferred. `"0"`: File has not been transferred.
47	`"1"`: File is from another node. `"0"`: File is from the same host.
49-51	Unit record device type.
53-60	Distribution code.
62-69	File creation date in the format `yyyymmdd`.
71-78	File creation time in the format `hh:mm:dd`.
80-87	Number of records in the spool file.
89	Hold status: `"n"`: None. `"U"`: User hold.
91	Diagnose F8 secure spool file origin available: `"1"`: Yes. `"0"`: No.

93-100	If column 91 = "1" and network file, then origin userid.
102-109	If column 91 = "1" and network file, then origin RSCS nodeid.
111-120	The number of kilebytes (1024 bytes) occupied by the spool file. The number is derived from the number of 4k blocks occupied. Thus, the precision is limited accordingly.

The last record from the *irdr* output stream contains only an asterisk (*) in column 1.

tmr output stream

Please also refer to the description of the *timer format* command: “TIMER” on page 22, which describes how you can access PIPESERV's internal list of the following field and column values.

Table 2 (Page 1 of 3). Record layout of the tmr output stream

Field name	**Columns**	**Description**
HEADER	1-7	Blanks.
	8	`"T"` (short for Timer).
	9-16	Userid that requested the timer.
	17-24	RSCS nodeid of the userid.
REQNO	25-34	Request number: A unique, decimal number assigned by PIPESERV. Right justified, blank padded.
FMTCODE	35-36	Format code: A decimal number indicating the level of timer record layout. Right justified, padded with zeroes.
ENTRBASE	37-43	Entry date in base format: The date where the timer command was entered. A decimal number indicating the number of days since and including 01.01.0001, and including the entry day. Gregorian calender correction is disregarded. Right justified, blank padded.
ENTRDATE	44-51	Entry date in yyyymmdd format (year month day): The date where the timer command was entered.
ENTRTIME	52-56	Entry time in seconds: The time of day where the timer command was entered. Right justified. Padded with zeroes.
LASTBASE	57-63	Date of last expiry in base format: The date where the timer request last expired. A decimal number indicating the number of days since and including 01.01.0001, and including the entry day. Gregorian calender correction is disregarded. Right justified, blank padded.
LASTDATE	64-71	Date of last expiry in yyyymmdd format (year month day): The date where the timer request last expired.

Table 2 (Page 2 of 3). Record layout of the tmr output stream

Field name	Columns	Description
LASTTIME	72-76	Time of last expiry in seconds: The time of day where the timer request last expired. Right justified. Padded with zeroes.
ORIGUSER	77-84	Origin userid: The userid that entered the timer command. Left justified.
ORIGNODE	85-92	Origin nodeid: The RSCS nodeid of the origin userid. Left justified.
TIMETYPE	93	Type of timer: A: Absolute, R: Relative, S: Start offset, P: Past whole hour.
TIMEIPL	94	Ipl type? "1": Yes, "0": No, blank: Bypass ipl test, "-": Not ipl timer type.
TIMEREST	95	Restart type? "1": Yes, "0": No, blank: Bypass restart test, "-": Not ipl timer type.
TIMETIME	96-100	Time: Time to execute request, according to timer type. Seconds, right justified, padded with zeroes.
DATEDATE	101-104	Date: Date to execute request: "mmdd": On the specified month and day, " dd": On the specified day in any month, " " (blank): On any day.
DATETYPE	105	Type of day to execute request: O: On the date, L: Last day of the month, 1, 2, 3, 4, and 5: Respectively on the first, second, third, fourth, and fifth occurence of the weekday specified in a month.
DATEWDAY	106	Weekday: 1: Monday, 2: Tuesday, 3: Wednesday, 4: Thursday, 5: Friday, 6: Saturday, 7: Sunday, W: Weekday (Mo-Fr), E: Weekend, "*": Any day.
DATEREPT	107	Repeat flag: "1" if request is to expired repeatedly, "0" if not.
DELAY	108-112	Delay in seconds: The number of seconds after a request should have expired where it will expire on PIPESERV initialisation. Right justified, padded with zeroes.
RANGE1	113-117	Start of range in seconds: The first time of the day where the request can expire. Right justified. Padded with zeroes.
RANGE2	118-122	End of range in seconds: The last time of the day where the request can expire. Right justified. Padded with zeroes.
ACTION	123	Action requested: C: Execute as command, R: Reminder, S: Write record to output stream.
TYPE	124	Type of timer: P: Permanent, T: Temporary, V: Volatile.

Table 2 (Page 3 of 3). Record layout of the tmr output stream

Field name	Columns	Description
STRI	125-*	String: The string specified.

cmd output stream

Columns	Description
1-7	Blanks.
8	`"S"` (short for Stream).
9-16	Origin userid.
17-24	Origin nodeid.
25-*	The command from the *cmd* input stream.

cmdt output stream

The *cmdt* output stream will write one record for each command presented on the *cmdt* input stream. One input record containing more than one command, separated by the PIPESERV lineend character, is treated as individual commands. Each record is divided into a number of logical records, separated by the character x'15'.

The format of the first logical record is

Columns	Description
1-7	Blanks.
8	`"s"` (short for Stream).
9-16	Origin userid.
17-24	Origin RSCS nodeid.
25-*	The command from the *cmdt* input stream.

The following logical records contain the console output from the execution of the command from the input stream, including the PIPESERV ready message.

If the *rc* output stream is also connected, the records on the *rc* and *cmdt* output streams are synchronised for the commands originating from the *cmdt* input stream. The output record is first written to the *rc* output stream, then to the *cmdt* output stream.

rc output stream

Columns	Description
1-8	The source of the command executed, right justified:

C	Command entered on the server console.

I	Command from the PIPESERV INITCMD file at PIPESERV initialisation.
i	Command from the PIPESERV INITCMD file at an *rlddata* command.
Q	Command from the CMS stack (queue).
S	Command from the *cmd* input stream.
s	Command from the *cmdt* input stream.
t	Command from an expired *timer* command.
IUCVCMD	Command from the synchronous IUCV interface.
00000001	Command from a CP message (MSG or MSGNOH).
00000004	Command from a CP special message (SMSG).

9-16	Origin userid.
17-24	Origin RSCS nodeid.
25-*	

Word 1	The return code from the command execution or an asterisk (*) if an unauthorised command was attempted.
Word 2-*	The command being executed.

icmd output stream

Columns	**Description**
1-7	Blanks.
8	`"I"` (short for Initcmd).
9-16	Origin userid.
17-24	Origin RSCS nodeid.
25-*	The command from the PIPESERV INITCMD file.

Pre-processing exit stream record formats

comx exit

Columns	Description	
1-8	The source of the command executed, right justified:	
	C	Command entered on the server console.
	I	Command from the PIPESERV INITCMD file at PIPESERV initialisation.
	Q	Command from the CMS stack (queue).
	S	Command from the *cmd* input stream.
	s	Command from the *cmdt* input stream.
	00000001	Command from a CP message (MSG or MSGNOH).
	00000004	Command from a CP special message (SMSG).
9-16	Origin userid.	
17-24	Origin RSCS nodeid.	
25-*		
	Word 1	The return code form the command execution.
	Word 2-*	The command being executed.

cmdx exit

The records have the same format as the *cmd* output stream.

cmtx exit

Columns	Description
1-7	Blanks.
8	`"s"` (short for Stream).
9-16	Origin userid.
17-24	Origin RSCS nodeid.
25-*	The command from the *cmdt* input stream.

icmx exit

The records have the same format as the *icmd* output stream.

conx exit

The records have the same format as the *cons* output stream.

stax exit

The records have the same format as the *stac* output stream.

msgx exit

The records have the same format as the *msg* output stream.

smsx exit

The records have the same format as the *smsg* output stream.

logx exit

The records have the same format as the log file records described in “Log file format” on page 53.

Log file format

Columns	Description
1-8	Date in the format `yyyymmdd`.
10-17	Time in the format `hh:mm:ss`.
19-26	Log record type (see below).
28-35	Userid.
37-44	RSCS nodeid.
46-*	Log text.

Log record types

Type	Description
Pipeserv	PIPESERV driver messages.
Initcmd	Records from the PIPESERV INITCMD file.
InitcmdR	Records from the PIPESERV INITCMD file that was reloaded.
Stack	Records that were in the CMS stack when PIPESERV was started.
Console	Console input.
StrmCmd	Records from the *cmd* input stream.
StrmCmdt	Records from the *cmdt* input stream.
TimerExp	When timer expires, action requested by a timer command.
TimerEnd	The last time a particular timer expires and so will no longer be repeated.
Reader	Reader file information.
Ireader	Initial reader file information.
User	User-written log records from the *log* input stream.
***M1:Msg**	CP *MSG service IUCV class 1 (message).
***M2:Wng**	CP *MSG service IUCV class 2 (warning). This is only logged if you connect to the *wng* output stream.
***M3:Cp**	CP *MSG service IUCV class 3 (cpconio).
***M4:Smsg**	CP *MSG service IUCV class 4 (smsg).
***M5:Vm**	CP *MSG service IUCV class 5 (vmconio).
***M6:Emsg**	CP *MSG service IUCV class 6 (emsg).
***M7:Imsg**	CP *MSG service IUCV class 7 (imsg).
***M8:Scif**	CP *MSG service IUCV class 8 (scif).
IucvReq	IUCV connection request received from a client.
IucvCmd	IUCV command received from a client.

PIPESERV return codes

RC	Description
0	Normal return from PIPESERV.
4	The command input stream to the internal PIPESERV command processor has disappeared. Possibly by incorrect use of the *debug* or *rexx* command. Start PIPESERV again.
8	The required level of CMS Pipelines has not been loaded. Refer to the requirements section to see where to find CMS Pipelines.
12	PIPESERV cannot execute in CMS subset. Enter RETURN to exit from CMS subset and try again.
16	PIPESERV could not commit to level 0, where data can flow. Examine your application programs for possible initialisation errors to determine if they are not starting up.
20	You are trying to start PIPESERV on an unsupported level of VM. Upgrade your VM system or move your PIPESERV application to a system with a supported level of VM.
24	You specified an invalid option on the PIPESERV command. Correct the syntax and try again.
28	You are trying to start PIPESERV on an unsupported level of CMS. Upgrade CMS or move your PIPESERV application to a system with a supported level of CMS.
32	A compatibility problem has occurred when reading the PIPESERV control files from a previous version of PIPESERV. If you do not need saved *timer* or *reminder* commands, simply erase the control files using the following commands: `pipe command ERASE PIPESERV savetmr` and `pipe command ERASE PIPESERV saveext`. Note that all saved *timer* or *reminder* commands will be lost. If you do need the commands, try to examine the files to determine which commands were saved. After erasing or renaming the files, issue the commands to PIPESERV, which will then create new files.
36	You have specified an invalid configuration file on the PIPESERV command. Examine the syntax and try again.
40	The configuration file you specified on the PIPESERV command was not found. Verify that all required minidisks and SFS directories are accessed.
44	When reading the configuration file, PIPESERV encountered an invalid statement. Verify the syntax in the configuration file and try again.
48	You have connected to only one of a pre-processing exit's input or output streams. Both input and output streams must be connected. Correct your pipeline topology and try again.
52	PIPESERV could not update its control files on filemode A. Verify that the filemode is accessed R/W, and that space is available.
54	PIPESERV could not update the LASTING GLOBALV file on filemode A. Verify that the filemode is accessed R/W, and that space is available.

56	You have connected to the *logx* pre-processing exit, which is provided to let your pipeline modify or filter the log records before they are written to the PIPESERV log file. However, the PIPESERV configuration file you are using specifies `LOGWRITE OFF`. Thus, the part of PIPESERV that updates the log file is inactive, and you cannot connect to it via the exit. You must either change the
58	You have connected to the *irdr* output stream, which is provided to give your pipeline access to an informational record for each reader that exists when PIPESERV is started. However, PIPESERV was invoked with the NOINITRDR option, which means that no records will be generated on the *irdr* output stream. You must either remove the NOINITRDR option or the connection to the *irdr* output stream.
60	You have connected to the *cmd* output stream, which is provided to give your pipeline access to records entered to PIPESERV on the *cmd* input stream. However, you have not connected to the *cmd* input stream, so no records will be generated on the *cmd* output stream. You must either remove the output connection or add the input connection.
62	You have connected to the *cmdt* output stream, which is provided to give your pipeline access to commands and command output for commands entered to PIPESERV on the *cmdt* input stream. However, you have not connected to the *cmdt* input stream, so no records will be generated on the *cmdt* output stream. You must either remove the output connection or add the input connection.
1xxxx	Error updating PIPESERV's control files on filemode A. RC-10000 is the return code from EXECIO. Ensure that a R/W filemode A is accessed, and that space is available. Then try again.
20000	An undefined REXX variable was encountered in PIPESERV. This could be an internal PIPESERV error or be caused by incorrect use of the *debug* command.
2xxxx	An REXX syntax error was encountered in PIPESERV. This could be an internal PIPESERV error or be caused by incorrect use of the *debug* command. RC-20000 is the REXX syntax error return code.
-4095	The pipeline has stalled. Investigate your application pipelines for possible stalling problems.

Program logic

This part of the document is for those who are curious to know how PIPESERV is working, and to support the author's volatile memory.

Diagram notation

Bullets (•) denote optional connections, depending on external stream connections or PIPESERV options.

Bullets followed or preceded by a word indicate stream identifiers, for example `•tmr`.

External pipeline stages and external interfaces are written in capitals.

Unless otherwise indicated, the direction of flow is right or down.

Arrowheads (▲ ▼ ◄ ►) in the diagram indicate the direction of flow.

Double arrowheads indicate external interfaces.

Words followed by a colon (:) are pipeline labels.

Similar stages, serially connected, performing similar functions (for example 3 *spec* stages) may be abbreviated to one box in the diagram, indicating the number and kind of stages.

Numbering of the input streams for faninany and output streams for fanout cannot be determined from the diagrams.

Question marks (?) refer to separate diagrams.

PIPESERV REXX

Figure 25 on page 57 shows the PIPESERV REXX flow. PIPESERV REXX is the main driver.

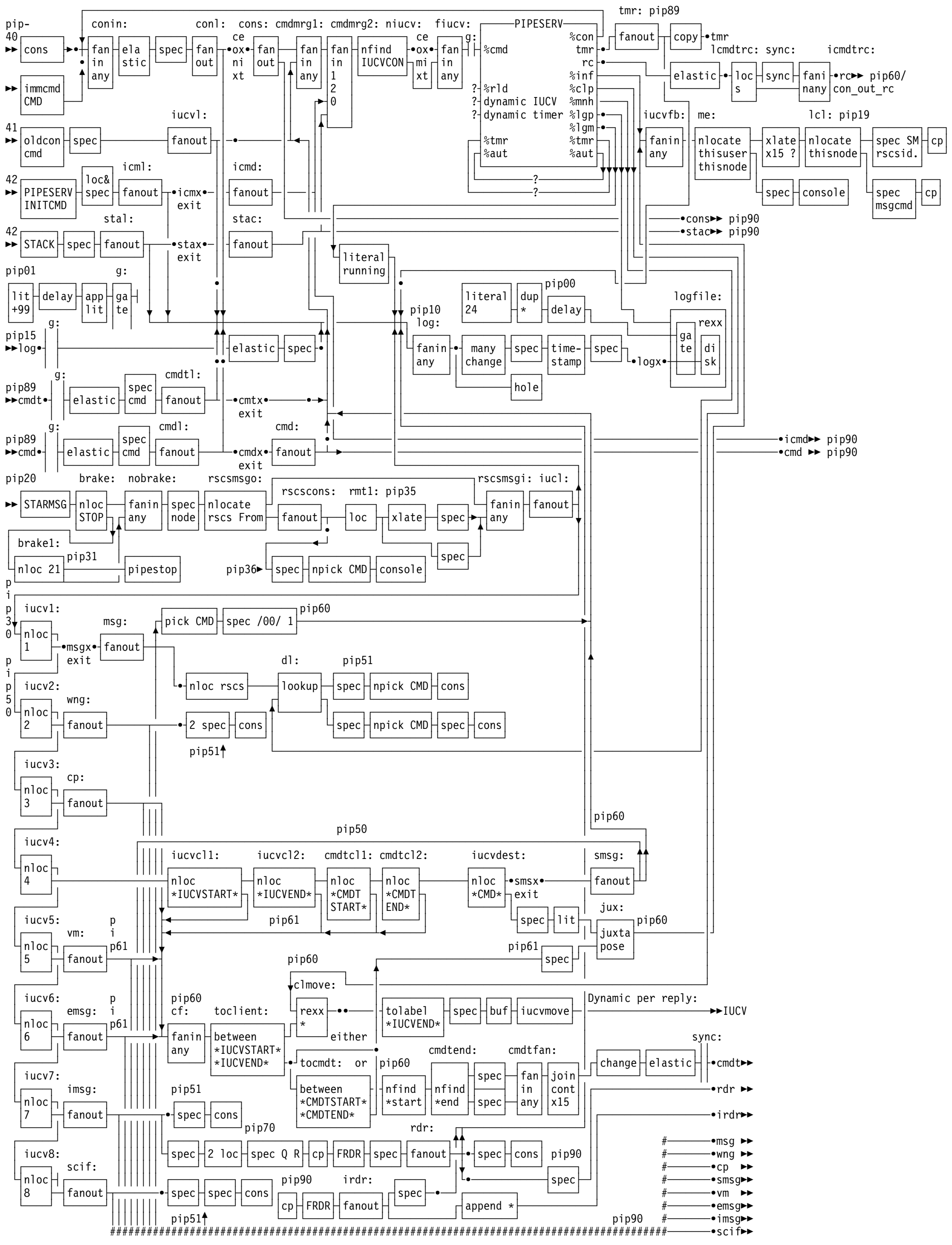


Figure 25. PIPESERV REXX flow

Figure 26 on page 58 illustrates the PIPESERV dynamic timer input connections. For every outstanding timer or reminder request, a temporary pipeline exists and is connected to a PIPESERV input stream. When the timer expires, the pipeline disappears, and the unconnected input stream is available for recycling by another timer request. If there are no unconnected input streams, a new will be created by PIPESERV.

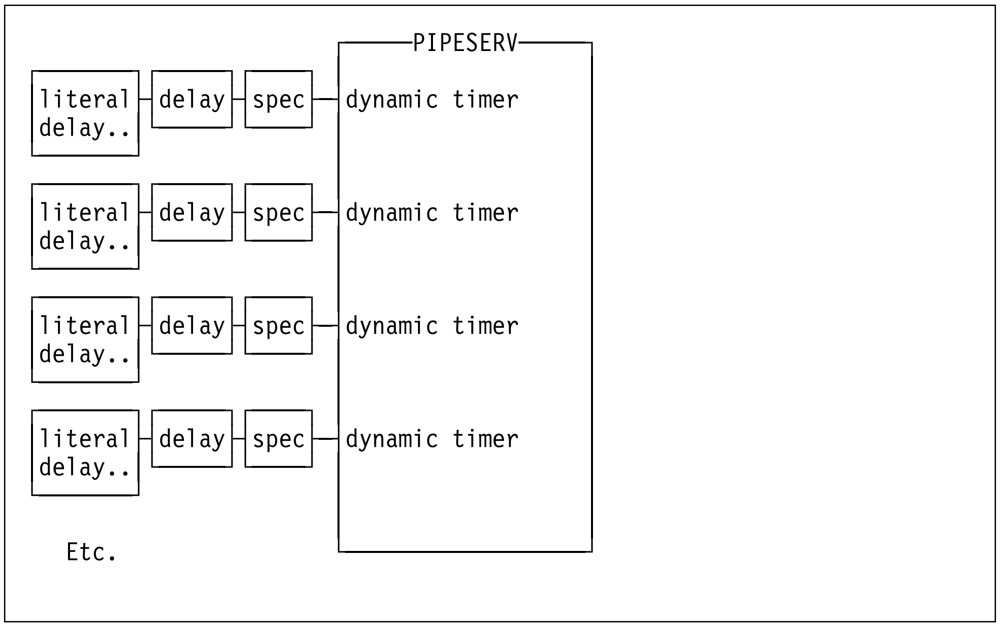


Figure 26. PIPESERV dynamic timer input connections

Figure 27 on page 59 illustrates the principle of the PIPESERV authorisation filters. When PIPESERV is initialised or restarted, or when the *rlddata* command is issued, the PIPESERV AUTHUSER file is read and transformed into a cascade of pipeline filters. Every command will be written to the AUTH output stream, except if originating from the PIPESERV user itself. The filters will try to match the command to the patterns specified in the authorisation file. A matching record will be written to the AUTH input stream. A record slipping through all the filters without match will be changed into a "0" and written to the AUTH input stream. This ensures that exactly one record is written to the input stream for every record written to the output stream. If there is no match, the command in question is rejected.

The filters make use of the *nfind* stage's facility of a blank character matching any character, and an underscore matching a blank.

In this example, the following commands would be authorised:

1. *rlddata* commands from JOHNDOE at VMNODE.
2. Any command starting with "Q" from JOHNDOE at VMNODE. For example QUERY DISK.
3. *rexx* commands from any user at VMNODE.
4. Any command from THISDOE at VMNODE.
5. *rexx* commands that have SAY as the first argument, from any user at any node starting with VM, for example VMGB5.

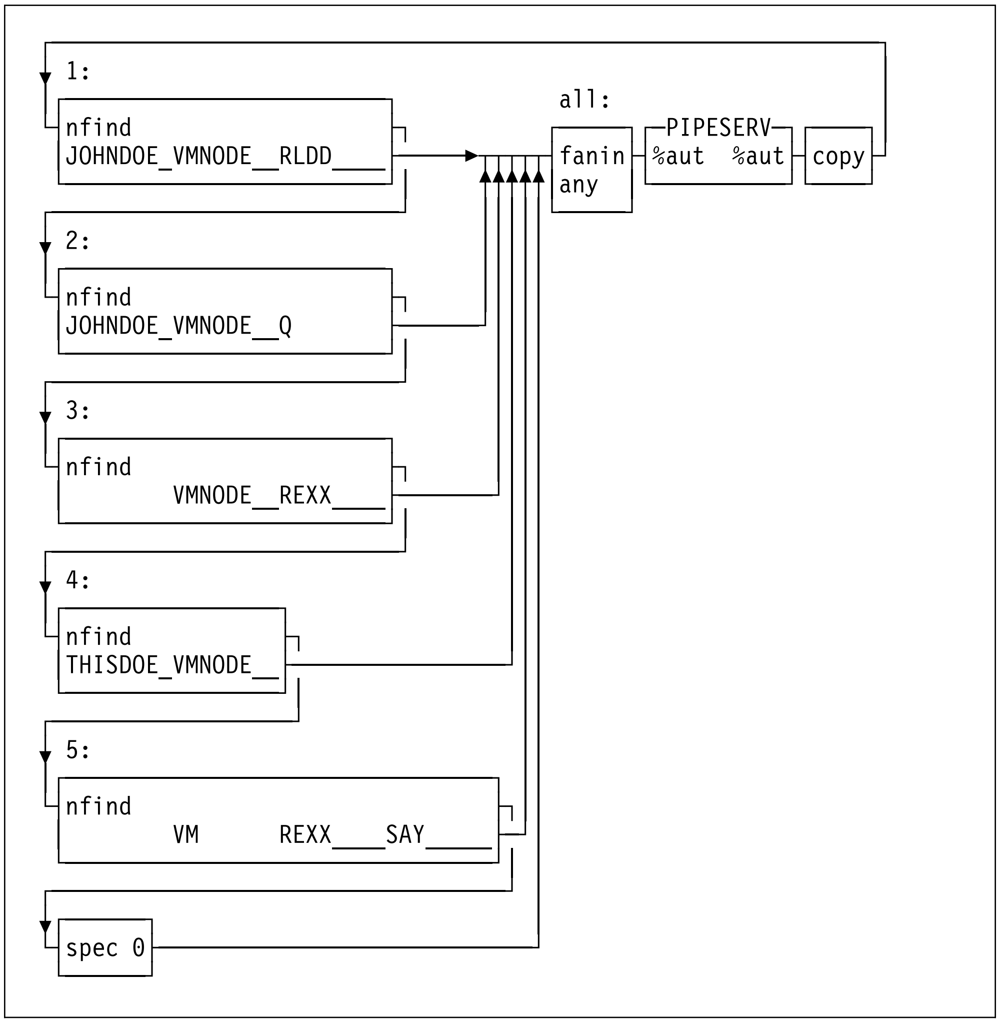


Figure 27. PIPESERV AUTH input and output connections

The authorisation records that created this example are illustrated in Figure 28.

```
JOHNDOE  VMNODE   RLDD
JOHNDOE  VMNODE   Q*
*        VMNODE   REXX
THISDOE  VMNODE   *
*        VM*      REXX     SAY
```

Figure 28. Authorisation records example

Index