CMS/TSO Pipelines

PIPE Command Programming Interface

1.1.12

jh95-0068-03

CMS/TSO Pipelines

PIPE Command Programming Interface

1.1.12

jh95-0068-03

! **Fourth Edition, May 2010.**

! This edition applies to *CMS/TSO Pipelines* level 1.1.12/04.

Third Edition, March 1998. This edition applies to *CMS/TSO Pipelines* level 1.1.10/24.

Minor editorial changes and clarifications, February 1997, for *CMS/TSO Pipelines* level 1.1.10/13.

First edition, November 1995. This book applies to level 1.1.9 of the *CMS/TSO Pipelines* Service Offering and to subsequent levels. Both VM/ESA Version 2 Release 1 Modification 0 and BatchPipes/MVS Release 2 contain this level, but neither product supports this specification.

Contents

Table of Contents

Part 1. Issuing Pipelines Programmatically

This part of the book describes ways to define and issue pipelines that are defined by control blocks and parameter lists rather than strings, as in the PIPE command.

Chapter 1. Issuing Pipeline Requests with Parameter Tokens

This chapter describes the ways you can call *CMS/TSO Pipelines* from an application program. We describe first how you get the request to *CMS/TSO Pipelines*; the mechanics of this depend on the operating system you are using. Then we describe the common format of the parameters that you pass to *CMS/TSO Pipelines*.

Getting to the Pipeline

The way to issue the PIPE command depends on the operating system you are using; and possibly also on the particular programming language. The following sections describe the contents of the parameter list that you must supply.

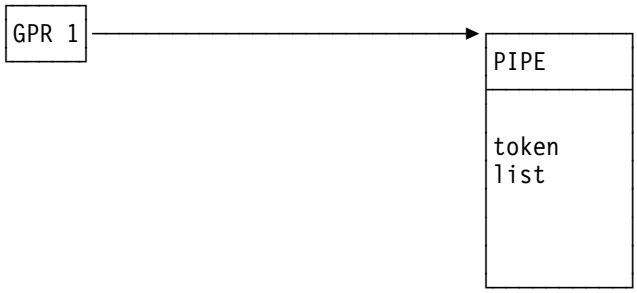
The objective is to pass a list of parameter tokens to the PIPE command.

CMS

On CMS, you must tunnel the parameter tokens through CMS command processing. This is done by prefixing the list of parameter tokens with a doubleword that contains the PIPE command; thus, the list of parameter tokens becomes part of the CMS tokenised parameter list.

The parameter list is passed in general register 1. It must be list of tokens, each eight bytes long and aligned on a fullword boundary (these are CMS tokens). The first token contains the PIPE command. The remainder of the parameter list contains the token list described in "Parameter Token List" on page 4.

Figure 1. CMS Parameter List



You can choose between CMSCALL or a direct branch to issue the pipeline.

Using CMSCALL to Invoke *CMS Pipelines*

Use `CMSCALL CALLTYP=PROGRAM,COPY=NO` to invoke *CMS Pipelines* through SVC-assisted linkage. If you wish to use general register zero as a base for addresses of the format described in Chapter 3, "Specifying Addresses and Lengths in Control Blocks" on page 14, you must instead use `CMSCALL CALLTYP=EPLIST,COPY=NO`.

Using Direct Branch to Invoke *CMS Pipelines*

Except when you wish to run the pipeline disabled, there are no compelling reasons to branch directly to *CMS Pipelines*.

But if you do, these are the steps to perform:

- Find the SCBLOCK describing the PIPE nucleus extension. (The NUCXMAP command can do this; or you can issue the NUCEXT macro to obtain the address.) If there is no PIPE nucleus extension, you must issue a dummy PIPE command to make the bootstrap install the main pipeline module (issue "pipe hole"). This process cannot be prevented from enabling; it is best that you issue the dummy PIPE command before you invoke the program that must run disabled. Do not try to emulate the process that installs the *CMS Pipelines* code and activates it as this process is likely to change over time.
- Load the address of the parameter list into general register 1; load the SCBLOCK address into general register 2.
- Ensure that general register 13 points to a CMS user save area and that the flags and call type are set to indicate that the call is a program call, an SCBLOCK is present, and that there is no extended parameter list. That is, the user save area must be twenty-four fullwords followed by the eight bytes of flags and call type.
- Load the address of the entry point from the SCBLOCK and issue the BALR instruction to branch to it. The addressing mode must be 31-bit.

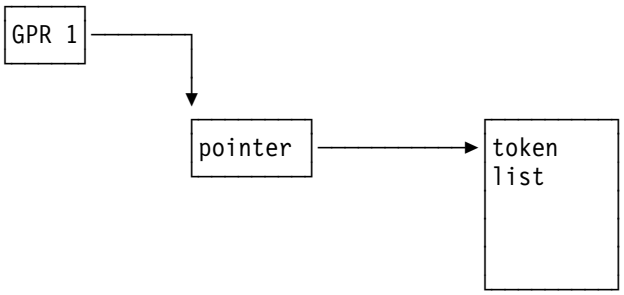
MVS

On z/OS, you have the choice of LINK; or LOAD, branch, and DELETE. The entry point name is PIPE (which is an alias for FPLPIPE).

You must use LOAD when you intend to run a copipe; otherwise the PIPE module could be unloaded from storage and you would be branching wildly when you resume the pipeline.

Pass in general register 1 a parameter list that has one entry. The parameter must point to the token list described in "Parameter Token List" on page 4.

Figure 2. MVS Parameter List



The pointer must have the leftmost bit turned on to indicate a parameter list that has one entry.

Thus, the parameter is a pointer to the parameter structure in C parlance. In PL/I you would pass a pointer to the first element of the token list.

Parameter Token List

A list of parameter tokens is used rather than the traditional list of addresses of individual parameters. It was designed this way to allow the same parameter format to be used with both CMS and z/OS.

The token list is normally generated using the PIPTPARM macro.

The generated token list contains three parts:

1. A doubleword that contains a "magic" token:

   ```
   DC   X'FFFF',C' pipe'   Two blanks
   ```

   This marks the parameter list as one that cannot possibly be issued as a command or addressed from REXX.

2. Tokens describing the actual requests. These tokens contain a fullword token type followed by a variable number of argument words. There is always an odd number of argument words to ensure that each token is aligned on a doubleword boundary (assuming the entire structure is aligned on a doubleword boundary).

3. A doubleword "fence", which contains all one bits.

Figure 8 on page 42 shows an example of a parameter token list.

Parameter Token Types

The supported token types are:

`encd` Provide the address of an encoded pipeline specification to be run. Refer to Chapter 2, "Encoded Pipeline Specifications" on page 7 for a description of this.

`fitg` Provide the address of the copipe communications area. This area is defined with a PIPFTPRM macro. This token sets *fitting mode*. That is, it indicates that the pipeline is to be run as a copipe. Normal mode (running the pipeline specification to completion) is used if this token is omitted.

`flag` Specify switches.

The flag bits are, from right to left (bigendian bit numbering):

`nospie`	31	Do not issue SPIE macro instructions. (This is the least significant bit.)
`nostae`	30	Do not issue STAE macro instructions.
`nostax`	29	Do not issue STAX macro instructions.

The leftmost bit (bit 0) is reserved for the sign; it will never be assigned.

`msgl` Provide a default message level. This is equivalent to *runpipe* MSGLEVEL. On CMS, all of the rightmost sixteen bits can be set, in contrast to the pipeline option which masks some of them off. On z/OS, the bits for X'3100' are masked off, because they would cause an ABEND if they were enabled.

`pipe` Provide the address and length of a character string that contains the pipeline specification to be run.

If the operand sublist does not contain a length operand, the length attribute of the address operand is used as the length of the pipeline specification.

`rc` Provide the address of a fullword to receive the return code from the request. Whether or not this token is specified, the return code will also be set in general register 15. Because the `rc` token cannot be used to report problems in the parameter list itself, the application should initialise the area pointed to with a value that will indicate failure to the application. Once the token list has been validated, the return code will be stored at the address contained in the `rc` token whenever the pipeline returns.

`sysv` Provide the address of a vector of system services entry points. The vector is described in Chapter 14, "System Services Vector" on page 54. This token is optional; a default set of system services interface routines is supplied by *CMS/TSO Pipelines*.

`uwrd` Provide the address to be stored as the user word for the pipeline set that will be created for the request. See Chapter 10, "*CMS/TSO Pipelines* User Words" on page 34.

You can specify tokens in any order. When a particular type of token is specified more than once, the last instance takes precedence. A `pipe` token overrides a `encd` token and *vice versa*. Note that the token identifier is a lower case character string. The formats of the tokens are summarised in Figure 3.

Addresses and lengths are specified in a generalised form described in Chapter 3, "Specifying Addresses and Lengths in Control Blocks" on page 14. If you do not build the tokens with the macros supplied, you should refer to that chapter; it may save you some work.

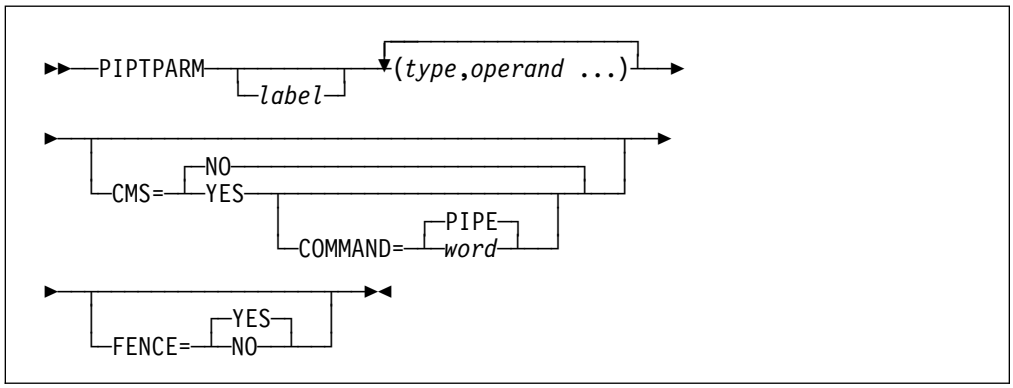
Figure 3. Parameter Token Summary

Type	Word 1	Word 2	Word 3
`encd`	Address of encoded pipeline specification.		
`fitg`	Address of communications area.		
`flag`	32 flag bits.		
`msgl`	Message level, 32 bits.		
`pipe`	Address of pipeline specification string.	Length of pipeline specification string.	0
`rc`	Address of word to receive return code.		
`sysv`	Address of the system services vector.		
`uwrd`	An anchor or the address of a global control block whence a device driver can retrieve interface information.		

Macro

Use the macro PIPTPARM to generate the list of parameter tokens.

PIPTPARM—Generate Parameter Token List



label The label is attached to the beginning of the parameter list. On z/OS, this label should be used as the argument in the CALL macro instruction.

type The token type. It can be written in upper case or in lower case (but not in mixed case). The macro will generate a lower case token in both cases. The type can optionally be enclosed in quotes.

operand The form that is appropriate to the request. Refer to Chapter 3, "Specifying Addresses and Lengths in Control Blocks" on page 14 for ways to specify operand values. If fewer operands are supplied than are required, the remaining slots are padded with zeros. Excessive operands are quietly ignored. If the `pipe` token is specified with only one additional suboperand, the operand specifies the address of the pipeline specification, and it is assumed that the length of the string can be determined from the length attribute of the specified symbol.

CMS Specify YES to generate the eight character CMS command as a prefix to the parameter token list. The COMMAND= operand specifies the command; the default is PIPE. If it is present, the *label* will be attached to this string rather than to the list of tokens. Omit the CMS parameter or specify NO to generate a token list that can be used on z/OS.

COMMAND Specify the command to issue. This operand has effect only when CMS=YES is specified.

FENCE Specify NO to suppress the fence that terminates the parameter token list. You would use this operand only if you intend to assemble further tokens "by hand". The token list must be concluded by a doubleword fence of all one bits.

Any addresses may be specified using the address notation that is defined in Chapter 3, "Specifying Addresses and Lengths in Control Blocks" on page 14.

Chapter 2. Encoded Pipeline Specifications

In contrast to the character string you normally issue as the argument to the PIPE command, an encoded pipeline specification is a control block structure that does not rely on special characters to delimit stages and pipelines.

The encoded pipeline specification is specified by one PIPSCBLK macro and several PIPSCSTG macros.

The PIPSCBLK macro specifies the overall structure of the pipeline specification; it includes the message level, options, and name of the pipeline specification. If a string representation of the encoded pipeline specification exists, it can also be specified; this will be made available in an event record.

PIPSCSTG macros are used to specify stages of the pipeline, as well as connectors and labels. The macro PIPSCSTG is also used to specify the beginning of a pipeline. When a stage is specified, its entry point can be specified as an address or as a name. If you specify an address, you can use stages that are local to your program.

A pipeline specification containing two pipelines could be written like this:

```
 PIPSCBLK TYPE=RUNPIPE
 PIPSCSTG TYPE=BEGIN
 PIPSCSTG TYPE=STAGE,LABEL=ABC,VERB=OSCAR
 PIPSCSTG TYPE=STAGE,VERB=CONSOLE
 PIPSCSTG TYPE=BEGIN
 PIPSCSTG TYPE=LABEL,LABEL=ABC
 PIPSCSTG TYPE=STAGE,VERB=CONSOLE
 PIPSCSTG TYPE=DONE
*
OSCAR    DC    C'oscar'
CONSOLE  DC    C'console'
```

The encoded pipeline specification can consist of more than one list of PIPSCSTG macros in which case the STAGES= keyword of the PIPSCBLK macro lists the first PIPSCSTG macro in each list. Each list is terminated by TYPE=DONE. The list can be broken at any point.

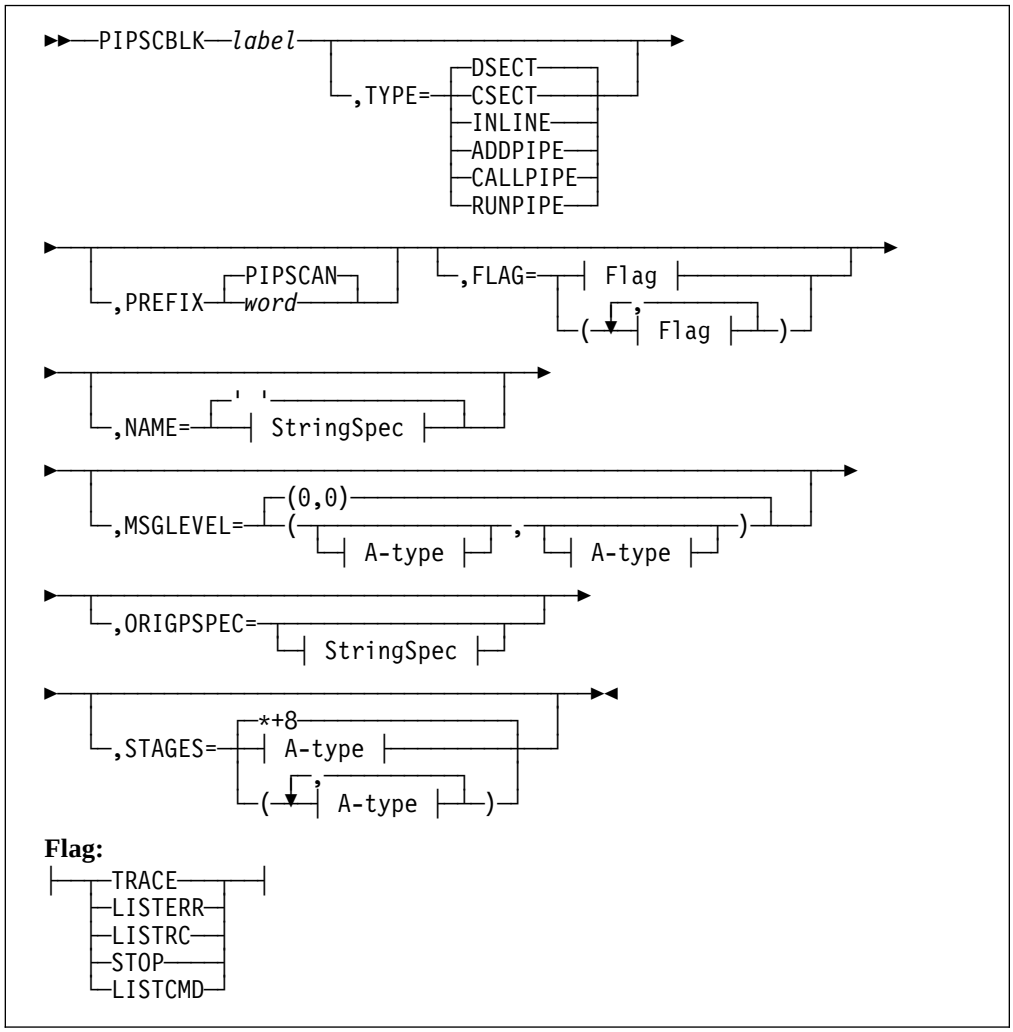
To run an encoded pipeline with a PIPSCBLK of type RUNPIPE, you should build a parameter token list (using the PIPTPARM macro). The parameter token list must include an “encd” token that specifies the name in the label parameter of the PIPSCBLK macro.

Encoded pipelines with a type of ADDPIPE or CALLPIPE are invoked through the PIPMISC macro with the ENCODED operand. The address of the PIPSCBLK macro expansion must be in register 1 when the PIPMISC macro is issued.

Macros

Note: The PIPSCBLK and PIPSCSTG macros are also used to define an encoded pipeline specification to be added to the currently running pipeline set. For completeness, all aspects of the macros are documented here even though some operands are not appropriate to the initial pipeline specification of a pipeline set.

PIPSCBLK—Generate Encoded Pipeline Specification



label Specify the label for the encoded pipeline specification. Use this label in the "encd" parameter token. Note that the label is not specified in column one of the card, as is customary.

TYPE — Specify the type of macro expansion desired.

- DSECT — Generate a dummy section defining the control block that represents an encoded pipeline specification. This is the default.
- INLINE — Like DSECT, but suppress generating a DSECT statement when the expansion is included in some larger control block.
- CSECT — Generate the control block inline in a work area. DCs are used; the fields are labelled.
- ADDPIPE — Generate a data area representing an encoded pipeline specification that is to be added to the currently running set. Except for the initial label, the contents of the data area have no labels. The encoded pipeline specification must be issued with the macro PIPMISC ENCODED. This operand is not valid with the PIPE command.
- CALLPIPE — Generate a data area representing an encoded pipeline specification that is to be run before the stage is resumed. Except for the initial label, the contents of the data area have no labels. The encoded pipeline specification must be issued with the macro PIPMISC ENCODED. This operand is not valid with the PIPE command.
- RUNPIPE — Generate a data area representing an encoded pipeline specification that is to be run as the initial pipeline of a new pipeline set. Except for the initial label, the contents of the data area have no labels. This operand is not valid with the PIPMISC macro.

PREFIX — Specify the prefix to be used for labels that are generated when the type is DSECT, CSECT, or INLINE.

The remaining operands are used only when the type is ADDPIPE, CALLPIPE, or RUNPIPE.

FLAG — Specify global options. Note that multiple options are specified in a comma separated sublist. The actual flags defined are prefixed the string specified by the &PREFIX keyword (PIPSCAN by default). You can specify TRACE, LISTERR, LISTRC, STOP, and LISTCMD.

NAME — Specify the name string. The string can be enclosed in quotes; it can be represented by the label on a character string constant; or it can be specified as two suboperands containing the address and length of the string, respectively.

MSGLEVEL — Specify the bits to add and the bits to remove from the message level. Only the bits for X'000017FF' can be turned on or off.

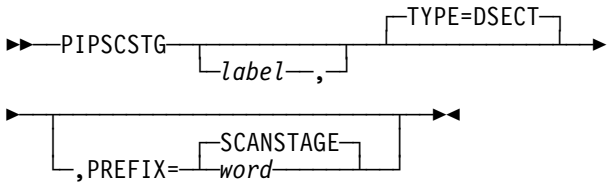
ORIGPSPEC — Specify the original pipeline specification, if one exists. The string can be enclosed in quotes; it can be represented by the label on a character string constant; or it can be specified as two suboperands containing the address and length of the string, respectively. This string is not used by *CMS/TSO Pipelines*, but its contents are made available in an event record, which can be used by the pipeline performance monitor known as RITA and similar applications. Thus, you are encouraged to supply this string if you have it; but do not spend time computing it.

STAGES — Specify the address of one or more sets of PIPSCSTG macros. The default assumes that the pipeline specification consists of one list of PIPSCSTG macros that immediately follows the PIPSCBLK.

PIPSCSTG—Generate Part of an Encoded Pipeline Specification

Depending on the type specified, this macro can generate a DSECT or the specification of part of an encoded pipeline specification.

Define DSECT

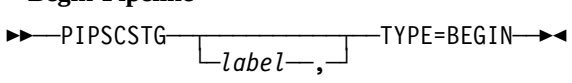


label Specify the label to be associated with this item.

TYPE Specify DSECT (which is the default) to expand the DSECT describing the four variants of this control block.

PREFIX Specify the prefix to be used for the labels in the DSECT.

Begin Pipeline

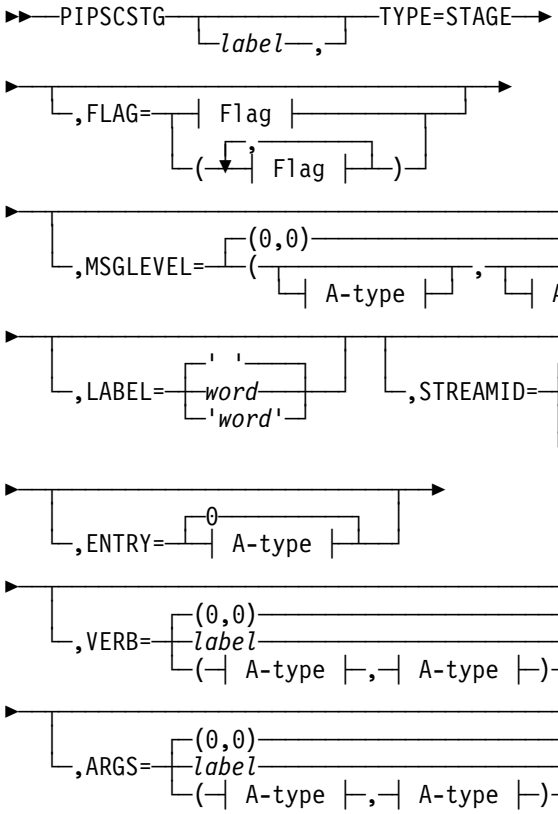


label Specify the label to be associated with this item.

TYPE Specify BEGIN to specify the beginning of a pipeline.

Note that the first PIPSCSTG macro of an encoded pipeline specification must be the begin type; there are always as many begin items as there are pipelines in the pipeline specification. (This is unlike the string representation of a pipeline specification, where the initial end character is optional.)

Define Stage



label Specify the label to be associated with this item.

TYPE Specify STAGE to specify a stage of a pipeline.

FLAG Specify local options. Note that multiple options are specified in a comma separated sublist. The actual flags defined are prefixed PIPSCAN to make them compatible with the default DSECT.

MSGLEVEL Specify the bits to add and the bits to remove from the message level. Only the bits for X'000017FF' can be turned on or off.

LABEL Specify a ***pipeLabel*** if the stage is to have more streams than the primary one being defined by this macro. This label is a label in the sense of *CMS/TSO Pipelines* stages in that it specifies what a stage's higher numbered streams are connected to; but it is also a label in the Assembler sense.

All ***pipeLabel***s in an Assembly must be unique, because they are resolved at Assembly time rather than at run time. The label must be a valid Assembler symbol (unlike a label in a pipeline specification string).

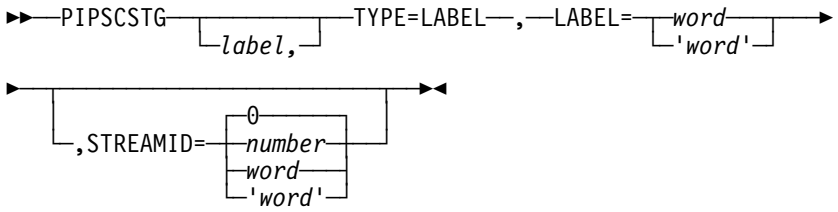
STREAMID Specify a stream identifier, if one is required or desired.

ENTRY Specify this operand if you can resolve the program to run at assembly time. Omit this operand if the pipeline specification parser must resolve the program to run in the normal way. Specify the label on a PIPDESC macro instruction that contains the program descriptor for the stage to run, if the stage uses a program descriptor. If the stage does not use a program descriptor (rather, it uses the original published interface), specify the label of the first instruction in the program.

VERB Specify the name of the program to run. This can be specified as a character string, the label on a character string, or a sublist containing the address and the length of the character string.

ARGS Specify the argument string for the stage. This can be specified as a character string, the label on a character string, or a sublist containing the address and the length of the character string. This is often specified by reference to the program's register set.

Label Reference



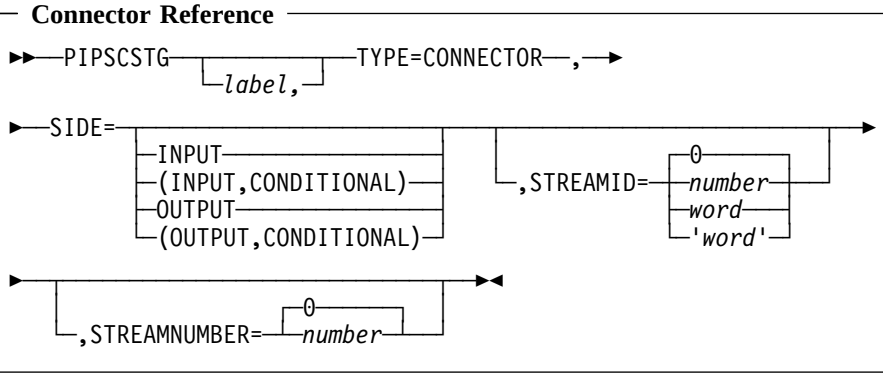
label Specify the label to be associated with this item.

TYPE Specify LABEL to specify a label reference.

LABEL Specify the ***pipeLabel*** of the stage that is to have another set of input and output streams.

STREAMID Specify a stream identifier, if one is required or desired.

Connector Reference



label Specify the label to be associated with this item.

TYPE Specify CONNECTOR to specify a connector.

SIDE Specify INPUT or OUTPUT. Specify CONDITIONAL if you wish a stream that is not defined to be treated as end-of-file.

STREAMID Specify a stream identifier if you wish to refer to the actual stream's identifier.

STREAM-NUMBER Specify a number if you wish to refer to the number of the stream you are connecting.

End of Segment

This form generates a terminator to show the end of the segment being defined. It can be after any other type of stage block.

Chapter 3. Specifying Addresses and Lengths in Control Blocks

An address, a length, or an integer in general can be specified in one of these ways:

- As an offset from the fullword being assembled. This is indicated by "-*" at the end of the parameter. The offset is limited to 8M in either direction (that is, it is a signed 24-bit number).

  ```
  PIPELINE-*
  ```

  This allows for relative address constants, which do not need relocation. Thus, the parameter list can be moved at runtime without needing to be relocated; it can be stored in a file and read in dynamically.

- As the number of a register in parentheses. The contents of the register at entry to *CMS/TSO Pipelines* are substituted.

  ```
  (4)
  (R5)
  ```

- As an S-type constant: a displacement plus the contents of a general register on entry to *CMS/TSO Pipelines*. The constant can be explicit base/displacement or it can refer to a label in your program. In particular, the label can refer to the contents of a DSECT.

  ```
  S(8(R2))
  S(PARMSTRING)
  ```

  When an S-type address is specified that is resolved by the Assembler, the data must be addressable from the register contents at the time *CMS/TSO Pipelines* is called. The Assembler must have the same USING statements in effect when the macro is assembled as will be in effect at the time *CMS/TSO Pipelines* is invoked to process the request.

- As the address of a fullword that contains a pointer to the argument; that is, an indirect reference. This is specified by prefixing a percent sign (%) to the address. The fullword must be addressable from the register contents at the time *CMS/TSO Pipelines* is called. The Assembler must have the same USING statements in effect when the macro is assembled as will be in effect at the time *CMS/TSO Pipelines* is invoked to process the request.

  ```
  %PIPEPOINTER
  ```

- As a label of a storage address or a self-defining symbol. This is assembled as an address constant. You can specify any expression that is valid in an A-type DC instruction.

  ```
  PIPELINE
  L'PIPELINE
  ```

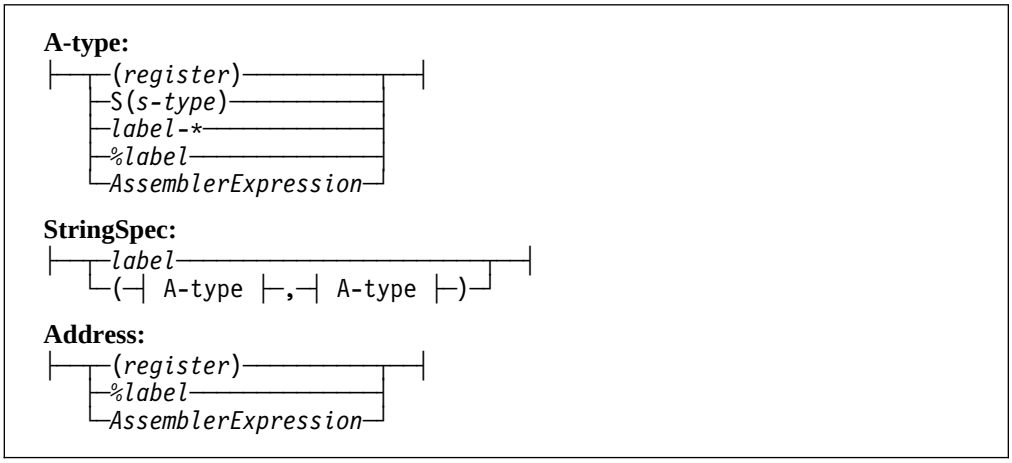
  Negative numbers must be greater than -16M, lest they be interpreted as one of the forms above; that is, if the sign bit is one, all eight leftmost bits must be one.

Using S-type References on CMS

When the PIPE command is issued by the CMSCALL macro with the CALLTYP=PROGRAM keyword operand, only register 1 is passed through CMS from your application to *CMS/TSO Pipelines*. If you specify CALLTYP=EPLIST, the address of the extended parameter list in register 0 is also passed to *CMS/TSO Pipelines*. In both cases it is *de rigueur* that you specify COPY=NO to ensure that CMS does not provide a copy of your parameter lists to *CMS/TSO Pipelines*. You need not construct an extended parameter list where

register 0 points; the parameter list is ignored by *CMS/TSO Pipelines* when it discovers the parameter token list.

Syntax Summary



Part 2. Filter Packages

This part of the book describes the interface between the glue code between *CMS/TSO Pipelines* and the contents of a filter package.

A sample filter package is built as part of the exposition.

Chapter 4. Overview of Filter Packages

A filter package is a collection of functions that act as an extension of the main pipeline module. It can contain a combination of any of these:

- An entry point table that resolves filters within the package.
- A message table that is searched when a message is issued from a stage in the filter package. The main message repository is used when the message is not resolved in the filter package.
- A table of built-in functions for *spec*.

Figure 4. Contents of a Filter Package

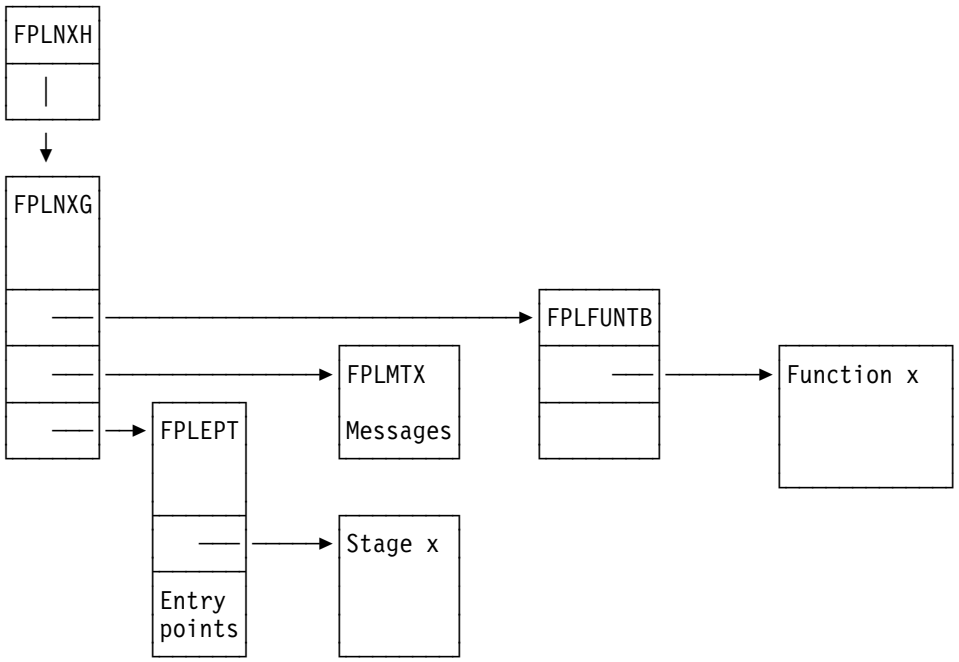


Figure 4 shows a modern type-2 filter package for CMS. Omit FPLNXH for TSO.

Installation and Retraction

The notion of a filter package grew over time. Thus, there is a number of interfaces, initially with no help from *CMS/TSO Pipelines*, but eventually embodied in various glue modules that are used to link the user's programs to *CMS/TSO Pipelines*.

Filter packages may be installed actively by the filter package invoking the appropriate interface or passively through the *filterpack* stage. The first type is often referred to as a type-1 filter package whereas the latter is a type-2 filter package. *CMS Pipelines* supports both types of filter packages, but *TSO Pipelines* supports type-2 only.

Interface Levels

The initial implementation consisted of a way for a program to install and retract its entry point table by invoking the PIPMOD command using a parameter list indicating the action to perform (install or retract) and the object type (entry point table or message table). Loading the program and issuing the PIPMOD commands were the user's responsibility.

PIPNXF, DMSPFP, and FPLNXF

The service module PIPNXF was soon written to handle the details of the PIPMOD interface. It was later augmented by code to support the filter package being in a CMS nucleus extension that would install itself when invoked as a CMS command; it retracts from *CMS Pipelines* when the nucleus extension is dropped.

The corresponding CMS module is DMSPFP, which is on the system disk. The runtime library supplies FPLNXF.

This level supports an entry point table and a message table, but no function table.

FPLNXG

FLNXG is the glue used with type-2 filter packages. It contains the assembled macro FPLFLTPK. This object module must be linked into type-2 filter packages. It contains address constants to resolve the contents of the filter package, but no code other that setting return code 16 when invoked as a command.

FPLNXH

FPLNXH supplies the active code of FPLNXF to install using the parameter list in FPLNXG. Thus, an active type-2 filter package can be constructed by linking both FPLNXH and FPLNXG into a CMS module.

Summary of Interface Modules

Figure 5. Glue Module Summary

	PIPNXF	**DMSPFP**	**FPLNXF**	**FPLNXG**	**FPLNXH**
Entry point tables (only first found used)	PIPEPT	PIPEPT	PIPEPT FPLEPT	FPLEPT	
Message text table (only first found used)	PIPMTX	PIPMTX	PIPMTX FPLMTX	FPLMTX	
Function table				FPLFUNTB	
Install/retract	Yes	Yes	Yes	No	Yes

Chapter 5. Entry Point Table

An entry point table can be generated by the utility FPLEPTBL as described in appendix A of *CMS/TSO Pipelines* Author's Edition or by assembling an entry point table with your code.

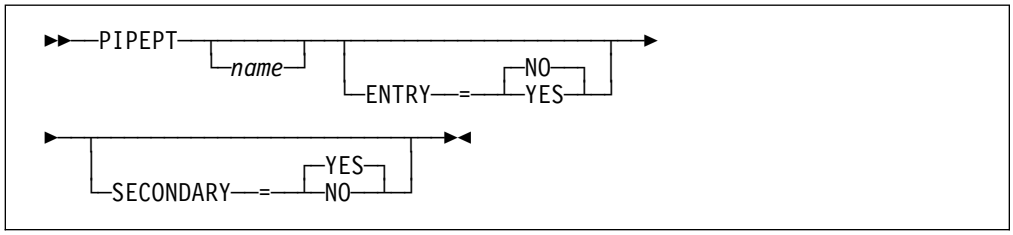
Macros to Assemble an Entry Point Table

Use the macros PIPEPT, PIPEPTEN, and PIPEPTED to assemble an entry point table.

For completeness, all operands of the macros are described, even ones not appropriate to an entry point table in a filter package.

PIPEPT—Open an Entry Point Table

PIPEPT opens the table and generates code to identify it and define its name.



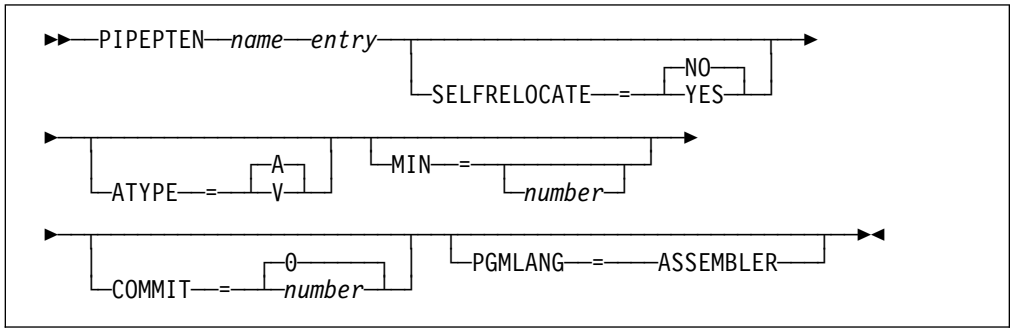
name Specify the label to generate for the symbol table. The default is PIPEPT. You must specify FPLEPT for use with FPLNXG or FPLNXH.

ENTRY Specify YES or NO, as appropriate. ENTRY is ignored when SECONDARY is specified, as the symbol must be an entry to the object module for the symbol to be resolved.

SECONDARY Specify whether the entry point table is referred to from another entry point table or not. In particular, specify NO for a filter package.

PIPEPTEN—Define an Entry Point

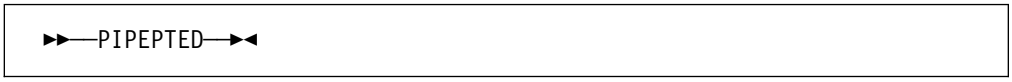
Each PIPEPTEN macro defines one filter. The macros must be ordered ascending in the EBCDIC collating sequence.



name	The name of the stage as it is specified in a pipeline specification.
entry	The label on the PIPDESC macro that assembles the program descriptor; or the label on the program list of an in-core REXX program.
SELFRELOCATE	Specify yes to save a relocation directory entry.
ATYPE	Specify V for a pipeline descriptor that is in another source file.
MIN	Specify the length of the minimum acceptable abbreviation. No abbreviation is available when this operand is omitted.
COMMIT	Specify the starting commit level. This operand is not appropriate when the name references a program descriptor; it is used to specify the stating commit level for a REXX program.
PGMLANG	Deprecated; do not specify.

PIPEPTED—End of Entry Point Table

Finish the entry point table with a PIPEPTED macro. No operands are accepted.



Example

Using an entry point table file:

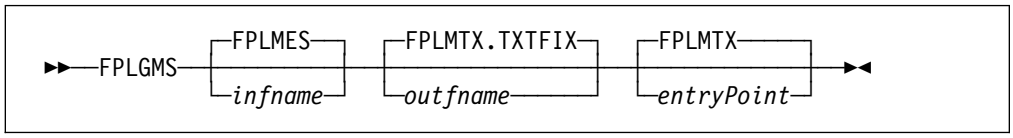
```
 pipe < fpltfp eptable | console
►msg900 fpltfpr1
►msg901 fpltfpr2
►Ready;
 fpleptbl fpltfp fpltfpep.text ( fplept
►Ready;
```

Chapter 6. Message Text Table

The input to generating a message table is a standard CMS message repository, even when generating messages for TSO. FPLGMS is for *CMS/TSO Pipelines* what GENMSG is to CMS, a way to generate a message text object module.

If you wonder why *CMS/TSO Pipelines* does not use the CMS infrastructure, the answer is twofold: First, *CMS Pipelines* predates the CMS message repositories; and second, TSO does not support the CMS message infrastructure.

FPLGMS—Generate Message Text Object Module



The three operands are:

1. The input file name. The file type is REPOS.
2. A single word that specifies the output file name and optional file type. The file type is abutted to the file name with a period. The default output file type is TXTFIX.
3. The name of the entry point for the message table. Refer to Figure 5 on page 19 for the appropriate entry point name, which depends on the glue module you will be using.

Example

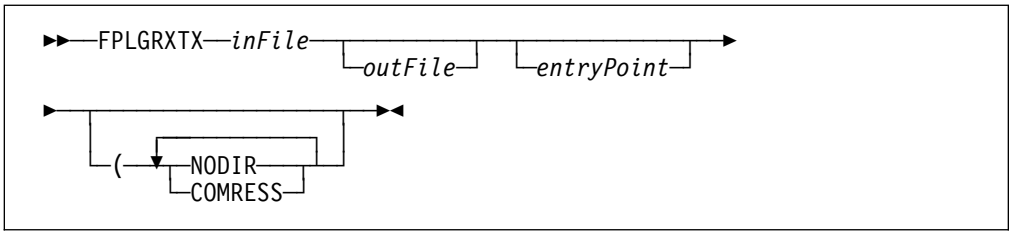
The input repository file is shown below. It contains messages 900 and 901. Both are format 01 and one line.

```
 pipe < fpltfp repos | console
►& 3
►09000101E This is the first test message.  (&1)
►09010101W This is the second test message.  (&1)
►Ready;
 fplgms fpltfp fpltfpmt.text fplmtx
►Ready;
```

Chapter 7. Generating an Object Module Containing REXX Stages

FPLGRXTX generates an object module containing in-core REXX stages.

FPLGRXTX—Generate REXX Filters Object Module



inFile Specify the input file name and type separated by a period. Several file types modify the processing of FPLGRXTX:

- REXXES — The input file contains a list of file names; the implied file type is REXX. This is the default file type.
- PACKAGE — The input file contains a list of files. The tokens &1 and &2 are removed from the first seven columns and the first three words are then the file name, the file type, and the file mode.
- EPTABLE — The first word of each input line is the file name of the filter; the file type is REXX.

For all file types, a leading asterisk indicates a comment, which is discarded.

outFile Specify the output file name, type, and mode separated by periods. The default file type is TXTFIX.

entryPoint Specify the the control section name to be generated.

NODIR Specify NODIR when you generate the entry point table manually, in particular if the filter package contains both REXX and Assembler filters. The file REXXES EPTABLE is created by default.

COMPRESS Compress each REXX program to a single line with redundant blanks removed.

Example

This example shows the list of files, the contents of the files, and the generation of the object module.

```
 pipe < fpltfp rexxes | console
►fpltfpr1
►fpltfpr2
►Ready;
 pipe < fpltfpr1 rexx | console
►/* Sample                                                          */
►'issuemsg 900 fpltfpr1 /parm1/'
►Ready;
 pipe < fpltfpr2 rexx | console
►/* Sample                                                          */
►'issuemsg 901 fpltfpr1 /parm2/'
►Ready;
 fplgrxtx fpltfp fpltfprx.text ( nodir compress
►Ready;
```

Trying to run the filters outside the filter package gets an error message because the 900 series of messages is reserved for user written filter packages:

```
 pipe fpltfpr1
►No message text for message 900.
►... Issued from stage 1 of pipeline 1.
►... Running "fpltfpr1".
►Ready;
```

Chapter 8. User Written Functions for *spec*

User-written functions for *spec* are supported in type-2 filter packages as of 1.1.12/04. A filter package can contain both filters and functions, but this chapter deals with functions only.

The function table is at the entry point FPLFUNTB (nonnegotiable). You can generate a self-installing type-2 filter package by including both FPLNXG and FPLNXH in the module.

A function can take op to 255 arguments; it produces a single result.

Refer to Chapter 16, "Sample *spec* Function Package" on page 67 for a complete sample function package.

Notes:

1. In this chapter, "floating point" refers to a counter that contains a decimal floating point number, as defined later. That is, not a hardware-defined data item.

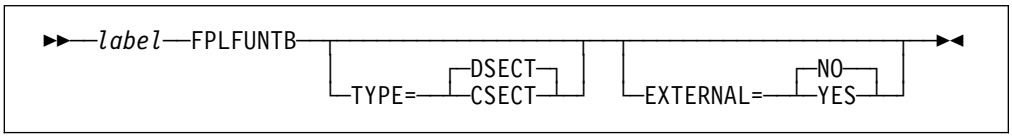
   Such a floating point number consists of a 31-digit decimal fraction, which is stored in a 16-byte packed decimal number, and a binary integer scale, which represent the power of ten with which the fraction is multiplied to obtain the number. A truly large range.
2. You must use the PIPCALL macro to call routine; external address constants will not work.
3. The pipeline services transfer vector must be available when a *CMS/TSO Pipelines* macro is issued or one of its subroutines is called.

Defining Functions to *CMS/TSO Pipelines*

The functions are defined by a function table, which is built using the macros FPLFUNTB, FPLFUNTE, and FPLFUNTN. You must define the DSECT versions of the first two macros before you can generate the CSECT version.

FPLFUNTB—Function Table Header

The function table header has this syntax:



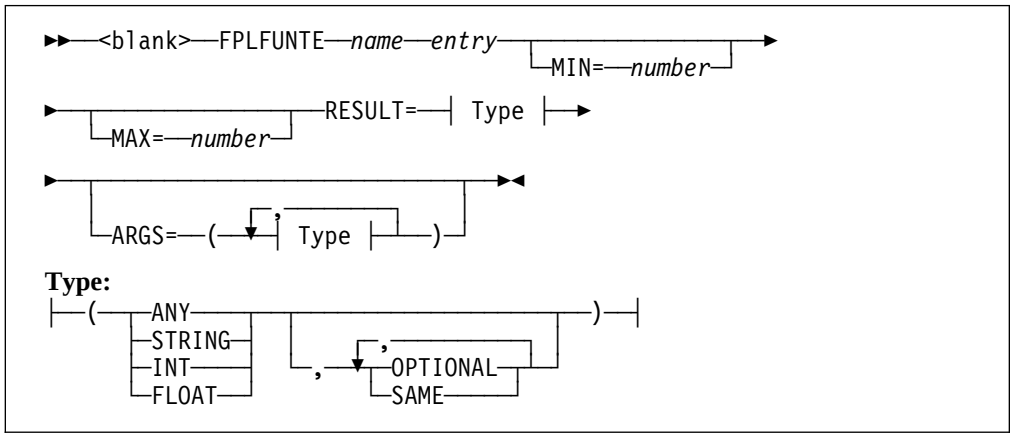
label Label on the table when TYPE=CSECT is specified. Must be blank otherwise.

TYPE Specify DSECT or CSECT as appropriate.

EXTERNAL Specify YES to generate V-type pointers to function implementation; specify NO for A-type.

FPLFUNTE—Function Table Entry

This macro defines one function, its result, and its arguments.



name The name of the function, as written in an expression.

entry The label of the entry point to the code that implements the function.

`MIN` Specify the minimum number of arguments. This is usually inferred from the first argument that is specified OPT.

`MAX` Specify the maximum number of arguments. This can be used in conjunction with SAME. The default is 255 when SAME is specified.

The type of the result and of each argument is specified by the two keyword operands. The types and optional flags are:

`ANY` Any type. The function determines and handles the actual argument.

`STRING` The argument is converted to a string, as required. This is unsuccessful only when there is no storage to hold the converted number.

`INT` The argument is converted to an integer. *spec* will issue a message and terminate if the argument cannot be converted to a signed 32 bit binary number.

`FLOAT` The argument is converted to a decimal floating point number having 31 digits precision and a fullword binary scale. *spec* will issue a message and terminate if the argument cannot be converted to float.

`OPTIONAL` The argument is optional. You may also specify `OPT`.

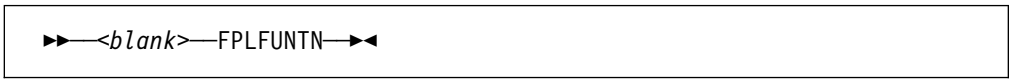
`SAME` The remainder of the arguments are the same as the one being defined. Specify `MAX` if the maximum supported number of operands is less than 255.

Notes:

1. The label field must be blank.
2. The minimum and maximum argument counts are normally inferred by FPLFUNTE from the ARGS operand.
3. The ARGS operand may be omitted only when the function takes no arguments; otherwise one suboperand is required.
4. FPLFUNTE macros must be supplied in the collating sequence by name.
5. Assembly will fail for function names that are longer than 100 characters.

FPLFUNTN—End of Function Table

This macro must be coded with a blank label field and no operands.



Example of REXX Function Definitions

This example shows how a few of the REXX-like functions are defined for *spec*. FUNX is a wrapper macro (not published) that issues FPLFUNTE with the first operand as both name and entry point.

```
FUNX    OVERLAY,                                                +
              RESULT=STRING,                                    +
              ARGS=(STRING,STRING,(INT,OPT),(INT,OPT),          +
              (STRING,OPT))                           FUNPACK
FUNX    POS,RESULT=INT,                                         +
              ARGS=(STRING,STRING,(INT,OPT))          FUNPACK
FUNX    REVERSE,RESULT=STRING,ARGS=STRING               FUNPACK
FUNX    RIGHT,                                                  +
              RESULT=STRING,                                    +
              ARGS=(STRING,INT,(STRING,OPT))          FUNPACK
FUNX    SIGN,RESULT=INT,ARGS=FLOAT                      FUNPACK
```

Function Entry Conditions

A function is entered with these register contents. The function must return with unchanged register contents, except for the return code in register 15.

! 0 Address of a counter where a numeric result is returned. You indicate that a value is present by setting the sign nibble to the appropriate value.

! 1 Address of a buffer (PIPBFR) where a string result is returned. The buffer is empty. A string result is indicated by not modifying the sign of the counter passed in register 0.

! 2 Address of the first argument of the argument list. Each argument is 20 bytes. Do not modify the arguments as they may contain information that *spec* needs after the function has returned.

! 3 Count of arguments.

! 9 Pipeline Services Transfer Vector (PSTV). Do not touch this register.

! 11 *spec* work area. Must be preserved on calls to FPLSPSLW.

! 13 Save area. The forward pointer at offset 8 contains the address of a 400-byte area that the function may use as save area or work area, or both. That is, HLLSTACK in PL/j parlance. Note that his 400-byte area is not contiguous with the save area in register 13.

! 14 Return address.

! 15 Entry point address. Return code on exit. A nonzero return code indicates a severe error; *spec* identifies the failing item in message 1490 and terminates.

! Format of the Result and an Argument

! The result (other than a string) and each argument is a 20 byte *spec* counter. While still ! termed a counter for historical reasons, this is a bit misleading as the counter can contain ! nonnumerical data.

! The counter is defined by the mapping macro FPLSPCTR. This section describes the labels ! defined when `PREFIX=` is omitted from the macro.

! The Sign Nibble

! The final nibble (which is the sign when a floating point number) defines the type of the ! counter's contents. This is of interest to you only if you have specified ANY for the type ! of the argument; otherwise *spec* will have coerced the argument to the type you require.

! For completeness, the following table describes all values used by *CMS/TSO Pipelines*.

&P._SIGNNULL	0	Null (unassigned) counter. Converts to 0 for numeric or a null string.
&P._SIGNOMITTED	1	Omitted parameter.
&P._SIGNCTRX	2	Lvalue. Pointer to actual counter in &P._SCALE. A user function should not see this type of argument.
	3	Unused
&P._SIGNLIT	4	The first 19 bytes of the counter contains a literal string that is prefixed a byte containing its length.
&P._SIGNSTRCNST	5	Address and length (four bytes each) of constant literal string in &P._STRING_ADDRESS.
&P._SIGNSTRSPACE	6	String in string space. _STRING_ADDRESS contains the offset and length of the string in the string space. Call FPLSPSLW to obtain the actual string.
	7	Unused
&P._SIGNBIN	8	Two's complement binary integer in &P._BINARY.
&P._SIGNUNSIGNEDBIN	9	Unsigned binary integer in &P._BINARY.
&P._SIGNPOSXACT	10	Exact positive floating point.
&P._SIGNNEGXACT	11	Exact negative floating point.
&P._SIGNPOS	12	Inexact positive floating point.
&P._SIGNNEG	13	Inexact negative floating point.

You do not need to decode all of the sixteen possible values. The sign nibble may be summarised by calling FPLSPVES. The address of the counter must be in general register 7; the encoded value is returned in general register 1. If the encoded value indicates a string, PIPSPSLW will return the string for any of the three string types.

For the floating types, the implied decimal point is at the right of the digits string. Thus, a value is definitely an integer when the scale is not negative. The number is also an integer when the scale is negative, but greater than the negative of the number of trailing zeros in the digits field. You may call FPLSPVTI to normalise the counter so that the absolute value of the scale is as small as possible. After that, a negative scale indicates a true fractional number.

The encoded values of the sign are also defined by the counter mapping macro FPLSPCTR. The prefix is not applied to these names.

OPENC_NULL	0	Null or omitted.
OPENC_BIN	1	Fixed point binary; possibly unsigned.
OPENC_FP	2	Floating point any sign, any exactness.
OPENC_STR	3	String of any variation.
OPENC_PCTR	4	Lvalue; should not occur.
OPENC_BAD	5	Not a valid encoding. This is an error in *CMS/TSO Pipelines*.

One way to decode an argument is shown in the next example.

```
LR   R7,R2      Get counter
PIPCALL SPVES   Encode it
CASBEG TYPE=VECTORED,MIN=0,MAX=OPENC_BAD,REG=R1
CASITEM OPENC_NULL
  SR R0,R0
CASITEM OPENC_BIN
  L  R0,_BINARY
CASITEM OPENC_FP
  L  R0,_SCALE
  CP _DIGITS,=P'0'  Zero?
CASITEM OPENC_STR
  LR R1,R2  Get counter
  PIPCALL SPSLW
  <R4,R5> now contains the string.
CASITEM
  <Trouble>
CASEND
```

Data Fields

The data fields of the counter are:

&P._SCALE	0	4	Exponent when the counter contains a floating point number; a pointer to another counter when the counter represents an lvalue.
&P._DIGITS	4	16	Packed decimal fraction (31 digits) when the counter contains a floating point number.
&P._BINARY	4	4	Integer.
&P._STRING_ADDRESS	4	4	Address of string.
&P._STRING_OFFSET	4	4	Offset to string.
&P._STRING_LENGTH	8	4	Length of string.
&P._LITERAL_LENGTH	0	1	Length of literal string.
&P._LITERAL_STRING	1	18	Literal string.

Again, recall that FPLSPSLW will take care of the last five fields for you.

CMS/TSO Pipelines Subroutines to Support Functions

A number of macros are available to support the programmer implementing a function. In fact, the full PL/j language is at your disposal as are all the scanning macros.

Supporting Entry Points You Can PIPCALL

The prefix FPL is omitted in this list as you do not specify it on the PIPCALL macro.

SPSLW Given a counter pointed to by register 1, return the string stored in it (irrespective of which of the three types) in registers 4 (address) and 5 (length). Note that the value of the string may not be stable beyond the return from the function.

SPVES Given a counter pointed to by register 7 return in register 1 the encoded type (the summary type).

SPVTI Given the address of a counter containing a floating point number in general register 7, try to reduce the absolute value of the scale. For a positive scale the digits are shifted right until the leftmost digit is nonzero; for a negative scales the digits are shifted right until the leftmost one is nonzero.

Macros

You may need to use several of the *CMS/TSO Pipelines* macros. Here are a few.

PIPBFRAP Use PIPBFRAP to append a string to a buffer, in particular the one provided for the result. Load the buffer address provided into register 1. The string to append is in registers 4 and 5 or you can specify a label as the second operand. If register 4 contains a negative value, the negative of the pad character is in the rightmost byte of the register. Assuming you have squirrelled away the buffer address in register 6:

```
PIPBFRAP (R6),'Not on your nelly!'
```

Or you can specify everything in the required registers:

```
LR   R1,R6
LA   R4,=C'Not on your nelly!'
LA   R5,18   (Perhaps)
PIPBFRAP ,
```

A nonzero return code from PIPBFRAP indicates that *CMS/TSO Pipelines* has run out of storage. You must issue message 122 in this case unless you specify EXIT= on the macro.

PIPBFRLD PIPBFRLD is a variant of PIPBFRAP that clears the buffer before loading the specified string.

PIPVERV2 Verify that your function runs on a recent version of *CMS/TSO Pipelines*. After this macro, general register 15 contains a positive value if the entry point is available.

Chapter 9. Generating the Filter Package Module

To generate the filter package after the constituent object modules are built.

```
 global txtlib fpllib
►Ready;
 load fplnxh (clear
►Ready;
 include fplnxg fpltfpep fpltfpmt fpltfprx fplfun (reset fplnxhsi
►Ready;
 genmod fpltfp ( from fplnxh
►Ready;
```

Note that FPLNXH is loaded first, that the module entry point is reset to it, and that the module is generated from that symbol.

Testing the filter package:

```
 fpltfp
►Ready;
 pipe msg900
►fpltfp900E This is the first test message. (parm1).
►... Issued from stage 1 of pipeline 1.
►... Running "msg900".
►Ready;
 pipe fltpack modlist fpltfp | console
►Module FPLTFP   loaded dynamically
►The first message is 900; last 901
►It contains 2 entry points
►Stage MSG900   at 019073F8 flags 00000000.
►Stage MSG901   at 01907478 flags 00000000.
►It contains 1 functions
►Ready;
 nucxdrop fpltfp
►Ready;
```

Part 3. Miscellaneous Interfaces

This part of the book describes other interfaces that do not fit within the previous parts of the book.

Chapter 10. *CMS/TSO Pipelines* User Words

User words are pieces of storage, within the *CMS/TSO Pipelines* control block structure, that are reserved for the use of the programmer writing pipeline device driver stages or interfaces to stages that require a runtime environment to be established for each stage.

User words are established in a hierarchy, which is shown in Figure 6.

Figure 6. User Word Hierarchy

Lvl	Name	Description
1	PIPGLOBUSER	A single word that is global to the entire pipeline structure within a virtual machine (strictly, for a particular nucleus extension installed for *CMS Pipelines*). On CMS, this user word is kept in storage that persists until the pipeline nucleus extension is dropped or until ABEND cleanup. On z/OS, this user word is kept in storage that persists until the end of the job step, or the *TSO Pipelines* module is reset by the command FPLRESET.
2	PIPTHRDUWRD	A single word that is specific to the process/thread combination. This user word is kept in storage that persists after a PIPE command has completed; it will exist at least as long as there is an active pipeline on the process/thread in question. An idle pipeline thread block (one representing a process/thread that has no active pipelines) may be reused for a different process/thread. Once it has been allocated, the thread block is retained until the pipeline nucleus extension is unloaded or until ABEND recovery.
3	PIPHUWRD	A single word that is specific to a particular pipeline set. Each PIPE command and each record passed to *runpipe* creates a new pipeline set. When the parameter token “uwrd” is specified, this user word is initialised to the value specified. That is, the contents of the second word of the “uwrd” token are copied into this word.
4	PIPVUWRD	A single word that is specific to a particular pipeline specification.
5	PIPBUSER	Three words that are specific to each stage.

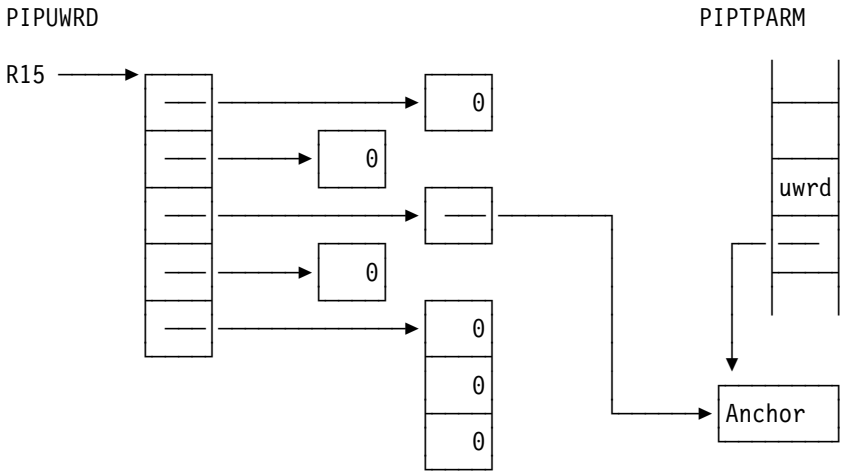
Notes:

1. On z/OS prior to *TSO Pipelines* 1.1.11, persistent storage was not used for the global control blocks; thus the first two user words were initialised on each PIPE command. Multitasking is not supported within a PIPE command. Recursions of the PIPE command are not recognised; the pipelines will run in parallel, as determined by the z/OS dispatcher.
2. On CMS, concurrent PIPE commands on different threads or processes are supported.

CMS/TSO Pipelines allocates the storage that contains the user words and initialises it to binary zeros. It exposes the address of the user words in two ways:

- The *CMS/TSO Pipelines* exit facility exposes the user words for global, set, and stage, but it does not expose the the user words for thread and specification. The stage user words are exposed as part of a larger data area (called the PIPEBLOK), which is made available to the exit. Refer to *User Exit* in *CMS Pipelines Installation and Maintenance Reference*, SL26-0019.
- The PIPUWRD macro returns in register 15 the address of a list of five pointers to the user words in the hierarchy described above. The list is stable until the next call for pipeline services in the pipeline specification. (The return code is zero if there is no active stage.)

Figure 7. PIPUWRD Return Code and the uwrd Parameter Token



Part 4. Copipes

This part of the book contains the answer to one of the questions often asked by programmers who use *CMS/TSO Pipelines*:

> I have this module written in [...] and I would like to apply the [...] filter to process one of its input files in a pipeline. How do I do that without writing an intermediary file or converting the program to a filter (which would be impossible, because of [...])? I would also like to do some of my own processing before passing the record to the [...] filter.

Copipes allow a program that is not a stage in a pipeline to start a pipeline that runs in conjunction with this controlling program. The controlling program can inject and extract data from the pipeline through *fitting* stages. A fitting stage is the controlling program's agent in the pipeline.

Each copipe is independent of all other pipeline activity; any number of copipes can be active concurrently. Control is returned to the controlling program when no more data can move in the pipeline. The controlling program must then direct *fitting* stage(s) to produce or consume records, as appropriate, to make data flow.

The POPEN interface is an example that uses a copipe to implement UNIX `popen()` with *CMS/TSO Pipelines*.

Chapter 11. Overview of Copipes

Two significant new concepts are introduced into *CMS/TSO Pipelines* to resolve the requirement to make pipelines available to existing programs: copipes and fittings.

Copipes make it possible for an application to be written partly as a traditional program and partly as a pipeline. When copipes are used, the application program and the pipeline run as coroutines. This means that the program and the pipeline each maintain their separate states; they take turns at running; and they transfer control between themselves by a *resume* operation.

Coroutines are not a new concept for users of *CMS/TSO Pipelines*. Stages in a pipeline run as coroutines; they take turns at processing data, but they do not call each other directly. Instead, pipeline stages yield control to the pipeline dispatcher, which creates the appearance that each stage is entirely in control of the sequence of events.

In the same way, pipeline sets run as coroutines when the *runpipe* built-in program is used to run a pipeline and process the messages and event records that are issued.

With copipes, this concept is taken further to allow an application that is not in any way controlled by *CMS/TSO Pipelines* to access data in the pipeline and to provide data for the pipeline. A *fitting* stage is the space warp through which records move between the pipeline and the application program. A *fitting* stage is the application's agent in the pipeline. A pipeline can contain any number of *fitting* stages.

To allow for additional information that must be specified to run copipes, *CMS/TSO Pipelines* has been changed to accept a new form of parameter list containing parameter tokens. This parameter list cannot be mistaken for a command entered from the terminal. The structure of the parameter list allows for easy future enhancement. Refer to Chapter 1, "Issuing Pipeline Requests with Parameter Tokens" on page 2.

When the application starts the copipe, it can use either the current string representation of the pipeline specification or a new format called an *encoded pipeline specification*. The encoded pipeline specification is structured and does not rely on special characters to delimit stages or pipelines. This is particularly useful when the application wishes to hand over data that are in a format that the application does not understand or does not wish to parse. Refer to Chapter 2, "Encoded Pipeline Specifications" on page 7.

Encoded pipeline specifications and copipes can be invoked only through parameter tokens, but they are not mutually requisite; nor are they mutually exclusive. A pipeline that contains a `fitg` parameter token is run as a copipe with the application that issues the request. A pipeline request that does not include the `fitg` parameter token is run to completion before control is returned to the application, just as it would be on a PIPE command. In either case, the pipeline to be run can be specified either by a character string (`pipe` token) or as an encoded pipeline specification (`encd` token).

General Notes

1. Addresses and lengths are specified in a generalised way that can save you building parameter lists dynamically when you need to substitute bits of runtime information into a statically assembled structure. Refer to Chapter 3, "Specifying Addresses and Lengths in Control Blocks" on page 14.

2. The word “label” is ambiguous when *CMS/TSO Pipelines* terminology is merged with the terminology of procedural programming. In both cases, however, a label identifies an object. In the syntax diagrams in this book, the syntax variable *label* is used for labels that identify an Assembler instruction or a control block, whereas *pipeLabel* is used for labels that define a multistream pipeline topology.

Chapter 12. Copipes and Pipeline Fittings

When the PIPE parameter token list includes a `fitg` token, the pipeline runs as a coroutine with the invoking program.

A pipeline set that runs as a copipe contains one or more *fitting* stages, each of which defines a point at which the application can send data into the pipeline or receive data from the pipeline depending upon the position of the *fitting* stage in the pipeline.

For each *fitting* stage, the application defines a request parameter list (RPL), which it uses to indicate the next action the *fitting* stage should take.

For each pipeline set run as a copipe, the application defines a communications area to contain the information needed to allow the application and the copipe to resume one another. An application can define multiple communications areas and invoke multiple concurrent copipes, but it remains to be seen whether this capability can be put to any practical use.

The pipeline is invoked using a parameter token list that contains a `fitg` token that specifies the address of the communications area and a `pipe` or `encd` token that specifies the pipeline to run.

When `nostart` is specified in the flag byte of the communications area, the pipeline returns on the initial command or call after the scanner has processed the pipeline specification, but before any stages have been started. When `nostart` is omitted, the pipeline is started and runs until it can move no further data.

Any errors detected by the pipeline specification parser or syntax exits will cause a nonzero return code on the initial call. This means that the pipeline has been abandoned and thus, no address is provided for the resume function. If an `rc` parameter token was provided, the fullword addressed by this token will be set to the return code, in addition to it being provided in general register 15.

If the initial return code is zero, *CMS/TSO Pipelines* stores information in the communications area before it returns to the application (resumes the application). The communications area is declared and generated with the PIPFTPRM macro.

The communications area contains:

- The level of the communications area. This is assembled as the constant 1.
- A flag byte for the application to control the copipe:[1]

[1] In the values for the flag bits, the variable symbol "&P." represents the string specified by the `PREFIX=` keyword on the PIPFTPRM macro.

`xxxx xxx1` (&P.NOSTART) Return to the application after the pipeline has been scanned, but before it is started. When this flag is zero, the pipeline is started (assuming the scanner finds no errors).

`xxxx xx1x` (&P.TERMINATE) Terminate the pipeline. The list of request parameters is ignored. The same effect can be achieved by specifying zeros instead of the pointer to the list of RPLs.

`xxxx x1xx` (&P.WAITRPL) Wait for the specified RPL to become ready. When this bit is zero, the pipeline will resume the application whenever it has no work to do.

- A status byte, which reports the progress of the pipeline:

  0 (&P.READY) One or more requests changed state. The copipe has not terminated.

  1 (&P.DONE) The copipe has terminated. Register 15 contains the return code from the pipeline. The return code is also stored at the address specified in an `rc` parameter token.

  2 (&P.NO_DATA) No request changed state. The application should provide data, consume data, or wait on the wake-up ECB. The pipeline is stalled if the wait count is zero and the application cannot provide or consume data. Refer to "Using Stages that Wait for External Events" on page 44.
- A list of RPLs to be processed.
- The address of the RPL that must be ready before the application wishes to be resumed. The 04 flag bit (&P.WAITRPL) must be on to enable this field. If this is enabled, *CMS/TSO Pipelines* will wait on external events until the RPL changes state rather than returning control to the application.
- An anchor whence *CMS/TSO Pipelines* retrieves its control blocks when it is resumed.
- A pointer to the "resume" function, which the application calls to resume pipeline operations.
- A pointer to a fullword that contains the count of the stages currently waiting on an external event.
- A pointer to an ECB that will be posted when an external event occurs.

When *CMS/TSO Pipelines* returns control on the initial call with a return code of 0, the pipeline set just created is kept in suspended animation until the application resumes it by issuing the PIPRESUM macro. As the only parameter on PIPRESUM, the application provides the address of the communications area, which contains the anchor that was returned initially and a list of *fitting requests* (RPLs) describing buffers that contain input records and places where the pipeline can store the address and length of output records. See Figure 9 on page 43 for the parameter list layout. The pipeline dispatcher then runs until it can find no more work to do. If the pipeline terminates, the status code is set to that effect. The pipeline then returns to the application, allowing it to resume.

The application then takes another turn, processing the data it has received from the pipeline and providing new data to be processed in the pipeline. The application then resumes the pipeline to allow it to take another turn.

A *fitting* stage represents a point in the pipeline where the calling program can inject or extract records. In order that multiple fittings can be supported, the fittings are given an identifier, which can be one to eight characters. Case is respected in fitting identifiers.

The fitting identifier is supplied as the argument to the *fitting* stage and in the request parameter list. This lets the copipe support relate a request to a particular *fitting* stage.

The scope of a fitting identifier is the pipeline set. That is, all fitting identifiers must be unique within the pipeline set, which comprises the initial pipeline specification and all pipeline specifications added by ADDPIPE and CALLPIPE.

A *fitting* stage can either read data from the pipeline or write data into the pipeline, but it cannot do both. A *fitting* that writes data from the application into the pipeline is a first stage; it has the behaviour typical of an input device driver. A *fitting* that reads data from the pipeline into the application is not a first stage; it has the behaviour typical of an output device driver. (It also passes the record to its output stream, if it is connected, and it does not propagate end-of-file backwards.)

The pipeline can contain stages that wait for external events, as for example *tcpclient*. When it does, the application can run while such a stage waits. See "Using Stages that Wait for External Events" on page 44.

Starting a Copipe

A copipe is established if the pipeline request contains a `fitg` request token. Figure 8 shows the layout of the parameter list when a copipe request is issued on CMS.

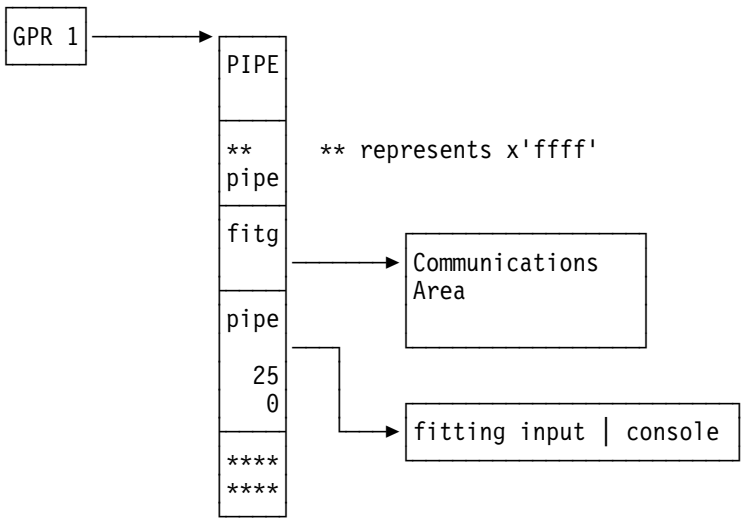
The parameter list could have been generated by this macro:

```
 PIPTPARM PREQ,CMS=YES,(FITG,S(COMM_AREA)),(PIPE,PIPELINE)
PIPELINE DC    C'fitting input|console'
```

The communications area would be defined by a PIPFTPRM macro in the work area:

```
 PIPFTPRM TYPE=INLINE,PREFIX=COMM_
```

Figure 8. Parameter List when Starting a Copipe (CMS)



Note: On CMS, you can use only registers 0 and 1 as a base for S-type references; and you must specify COPY=NO on the CMSCALL macro to be sure that the addresses passed to *CMS Pipelines* are the ones you have passed to CMS. Refer to "Using S-type References on CMS" on page 14 for further information.

Resuming a Copipe

Use the PIPRESUM macro to resume the pipeline.

Figure 9. Parameter list when Resuming a Copipe

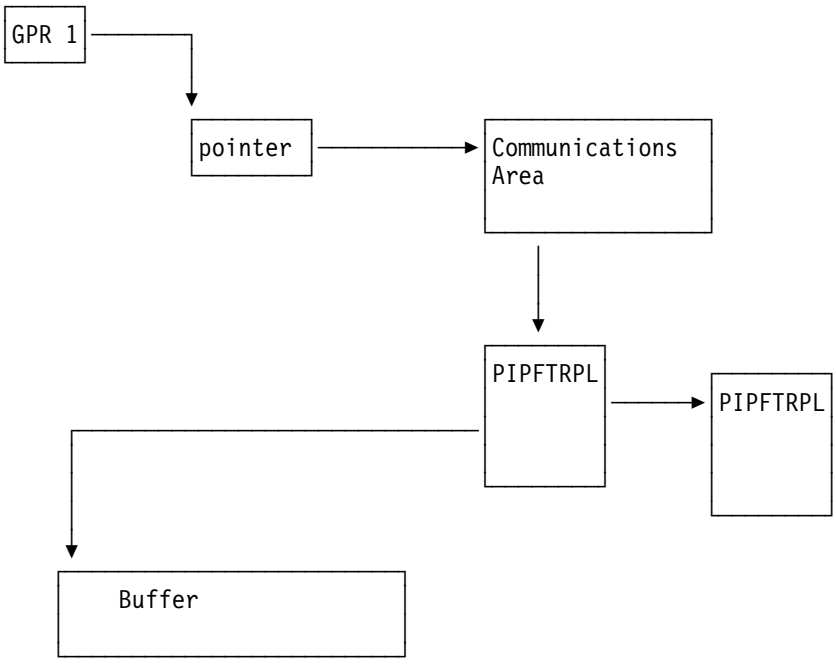


Figure 9 shows the parameter list layout. This structure is the same on CMS and z/OS.

The leftmost bit of the pointer to the communications area must be on to indicate that there is only one parameter on the resume operation. Return code -10 is set in general register 15 and the resume operation is suppressed, if the parameter list has more than one entry or if the communications area anchor does not point to the appropriate type of control block.

Terminating a Copipe

The application can force the copipe to terminate by issuing the PIPRESUM macro with the TERMINATE=YES keyword operand. This forces the copipe to terminate all active *fitting* stages. If this does not terminate the pipeline, all stages that are waiting for an external event are signalled to terminate, in the same way they would terminate if a record were passed to a *pipestop* stage. If this still does not terminate the pipeline, it will be forced to stall. The application can be sure that the pipeline has been terminated and that all resources have been released when it regains control after this final resume operation.

The application must ensure that the pipeline has terminated before it terminates itself. Resource leakage is likely if the pipeline is left running; on z/OS, some of the lost resources may be reclaimed at the end of the task, but others (*e.g.*, subpool 0) will not be reclaimed until the job step task terminates. On CMS, resources will not in general be reclaimed until ABEND recovery.

If the “done” status code has not been received, the application should issue the PIPRESUM macro to terminate, as described above. Such a call will always set the “done” status.

Once the application has received a “done” status code, it must not resume the pipeline.

Using Stages that Wait for External Events

To allow the application to coordinate its own WAIT macros with *CMS/TSO Pipelines*', the interface exposes the pipeline "wake up" ECB, which is posted when one of the waiting stages becomes ready to run. *CMS/TSO Pipelines* also exposes the count of stages that are currently waiting. The application should use the standard WAIT macro to wait on this ECB if it receives the NO_DATA status code, it has no other work to perform, and there are pipeline stages waiting for the wake-up ECB.

Because the count of stages waiting is decremented as they become able to run, the application can see NO_DATA and zero stages waiting if an event occurs after the pipeline dispatcher has determined that there is no more work, but before the application tests the counter. In this situation, the application should resume the pipeline once more. If NO_DATA is still set and the counter is still zero, the pipeline is truly not able to move further and the application must recover by supplying or consuming data, or it must terminate the pipeline. An alternative strategy is to use the WAITRPL flag and throw the problem over the fence to *CMS/TSO Pipelines*.

Using Fitting Stages to Inject and Extract Records from the Pipeline

The PIPFTRPL macro instruction generates the fitting request parameter list, which represents an action to be performed by a *fitting* stage. There must be a separate fitting request parameter list for each *fitting* stage in the pipeline set.

A request can be in one of the states listed below. An application would normally create an RPL in the Read or Write state, but it can even create an Idle RPL.

Fitting States and Transitions

The application and the *fitting* stages cooperate by taking turns at the work to be done. To describe in one variable both the application's view of the state of the fitting and *CMS/TSO Pipelines*' view of the state of the fitting, the fitting state field in the RPL has quite a number of potential values.

Some values indicate that the application is in control of the fitting, while others indicate that *CMS/TSO Pipelines* is currently in control of the fitting. The application must not change an RPL unless the fitting is in one of the states where it is indicated that the application may change the state.

Idle The application does not wish any action to take place; *CMS/TSO Pipelines* has no pending action to report. An idle request is ignored. The application can reactivate an idle RPL by taking it to state Read or Write, as appropriate, at some later time.

Ended The corresponding *fitting* stage has terminated. The application must not change the state of such an RPL. To improve efficiency, the RPL should be removed from the list.

Write The application has provided data to be written into the pipeline by the *fitting* stage. The corresponding *fitting* stage must be first in a pipeline.

The application should store the address of the record in the field that has the label &P.BUFFER; it should store the length of the record in the field labelled &P.BUFFER_LENGTH. The contents of the designated storage area must remain stable until the RPL reaches the W.Done state or the Ended state.

W.Resolving A Write request could not be resolved immediately, because no *fitting* stage with the specified identifier is active. Typically, this state is entered when the *fitting* stage is in a subroutine pipeline that has not yet been started.

W.Running *CMS/TSO Pipelines* is processing the request; the data are "in the pipeline". Input data must remain stable.

W.Done The output record has been consumed. The application can now provide the next record.

Read The application will receive the address and length of the next record on the input stream of the corresponding *fitting* stage. This *fitting* stage must not be first in a pipeline.

R.Resolving A Read request could not be resolved immediately, because no *fitting* stage with the specified identifier is active. Typically, this state is entered when the *fitting* stage is in a subroutine pipeline that has not yet been started.

R.Running *CMS/TSO Pipelines* is processing the request; data are not yet available.

R.Done Data are available in the request parameter list.

The record is provided by *CMS/TSO Pipelines* in a buffer that it has allocated (or one provided by a fitting writing the record into the pipeline). The address of the record is stored at the label &P.BUFFER; the length of the record is at label &P.BUFFER_LENGTH. The contents of the buffer can be relied upon to be stable only as long as the RPL remains in the R.Done state.

Consume The *fitting* stage should consume the record, but it should not prepare to read further.

Reject The *fitting* stage is unable to perform the Read or Write requested, because it is in the wrong position of the pipeline. A reason code is returned in the request parameter list when this state is entered. The Rejected state is also entered if the request parameter list contains a state that is incompatible with the state that the *fitting* stage is in. (That is, the application modified the state when the request parameter list was in a state where it was *CMS/TSO Pipelines*' turn to change the state.)

Terminate The corresponding *fitting* stage should terminate.

T.Resolving A Terminate request could not be resolved immediately, because no *fitting* stage with the specified identifier is active. Typically, this state is entered when the *fitting* stage is in a subroutine pipeline that has not yet been started.

Figure 10 on page 46 shows the valid state transitions. Transitions that are marked "Pipeline" in the third column are performed by *CMS/TSO Pipelines* while it is in control. Transitions that are marked "Application" are performed by the application while it has control. Several transitions can happen for a request while *CMS/TSO Pipelines* is in control; for example, the application might observe a state change directly from Write to W.Done, except possibly on the first Write.

Figure 10. Valid RPL State Transitions

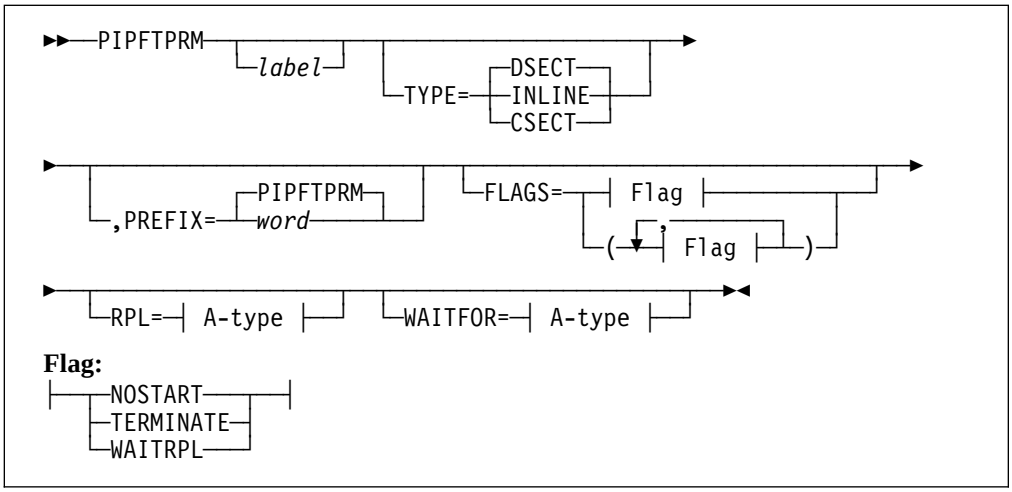
From	To	By	Remarks
Idle	Write	Application	The application wishes to write data into a *fitting* stage that is currently idle.
Idle	Read	Application	The application wishes to read data from a *fitting* stage that is currently idle.
Idle	Terminate	Application	The application wishes to terminate a *fitting* stage that is currently idle.
Write	W.Resolving	Pipeline	No *fitting* stage is running that has the specified ID.
W.Resolving	W.Running	Pipeline	The ID is resolved. The record is "in the pipeline".
W.Resolving	Reject	Pipeline	The *fitting* stage is unable to process the request, because it is not first in a pipeline and thus supports only Read requests.
W.Running	W.Done	Pipeline	The record has been consumed. The application can provide a new record.
W.Running	Ended	Pipeline	The *fitting* stage received end-of-file while writing the record into the pipeline. The stage has terminated.
W.Done	Write	Application	The application has provided the next record.
W.Done	Terminate	Application	The application will not provide further data. The corresponding *fitting* stage should terminate.
Read	R.Resolve	Pipeline	No *fitting* stage is running that has the specified ID.
R.Resolve	R.Running	Pipeline	No record is currently available on the input to the corresponding *fitting* stage.
R.Resolving	Reject	Pipeline	The *fitting* stage is unable to process the request, because it is first in a pipeline and thus supports only Write requests.
R.Running	R.Done	Pipeline	The address and length of the record are provided in the RPL. The input record has not been consumed by the *fitting* stage.
R.Done	Read	Application	The application has processed the contents of the record and is ready to receive a new one. The corresponding *fitting* stage should consume the input record and read another one.
R.Running	Ended	Pipeline	The corresponding *fitting* stage has received end-of-file on its primary input stream. The stage has terminated.
R.Done	Consume	Application	The application does not wish to process further data at this moment, but it has completed the processing for the record it read and the record should be consumed by the *fitting* stage.
Consume	Idle	Pipeline	The corresponding *fitting* stage has consumed the input record.
R.Done	Terminate	Application	The application does not wish to process further data. The corresponding *fitting* stage should terminate without consuming the record.
Terminate	Ended	Pipeline	The corresponding *fitting* stage has terminated.
Terminate	T.Resolving	Pipeline	No *fitting* stage is running that has the specified ID.
T.Resolving	Ended	Pipeline	The corresponding *fitting* stage has terminated.

Be Careful!

- A particular *fitting* stage can either read or write, but it cannot do both.
- Only one RPL may refer to any particular *fitting*.
- The FITTING_PTR field of the RPL is resolved to the address of a control block that represents the *fitting* stage; the application must not modify the contents.
- When a copipe is inactive, the only way to find its control block is through the anchor, which you have control over. If you lose the anchor or terminate your application before the pipeline has completed, you are likely to suffer storage leakage.
- If you resume the copipe after it has completed, results are unpredictable, but invariably unpleasant. Expect an ABEND as your reward.

Macros

PIPFTPRM—Generate Copipe Communications Area



label Specify the label for `type=csect`. Note that the label is not specified in column one of the card, as is customary.

`TYPE` Specify `DSECT` to generate a separate DSECT containing the definition of this parameter list. Specify `INLINE` to generate the definition inside some other section (DSECT or CSECT). Specify `CSECT` to generate a parameter list without internal labels (entirely without labels if *label* is omitted).

`PREFIX` Specify the prefix to be used for the labels that are generated when `type=DSECT` or `type=INLINE` is used.

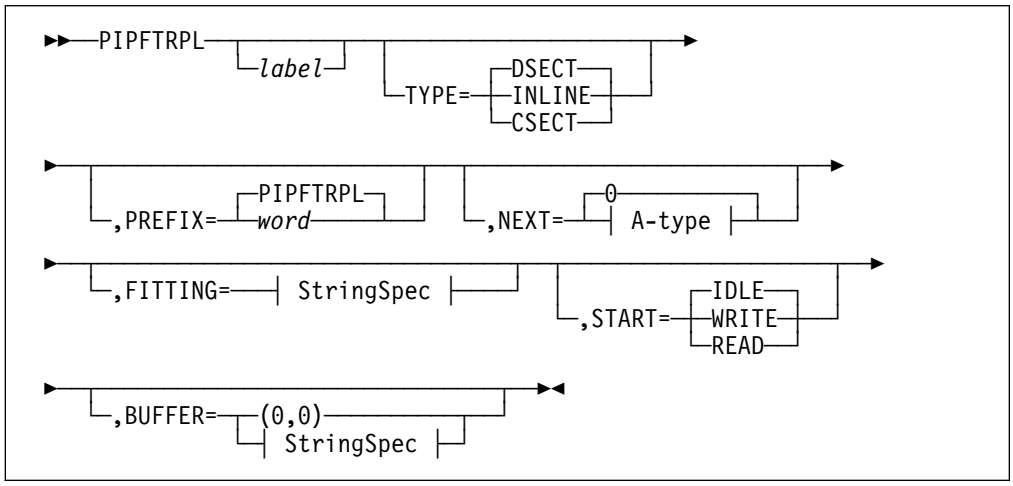
`FLAGS` Specify flag values to be assembled initially.

`RPL` Specify the address of the head of the list of RPLs.

`WAITFOR` Specify the address of the special RPL.

Refer to Figure 12 on page 50 for the definitions of the flags.

PIPFTRPL—Generate Fitting Request Parameter List



label Specify the label for TYPE=CSECT. Note that the label is not specified in column one of the card, as is customary.

TYPE Specify DSECT to generate a separate DSECT containing the definition of this parameter list. Specify INLINE to generate the definition inside some other section (DSECT or CSECT). Specify CSECT to generate a parameter list without internal labels (entirely without labels if *label* is omitted).

PREFIX Specify the prefix to be used for the labels that are generated when type=DSECT or type=INLINE is used.

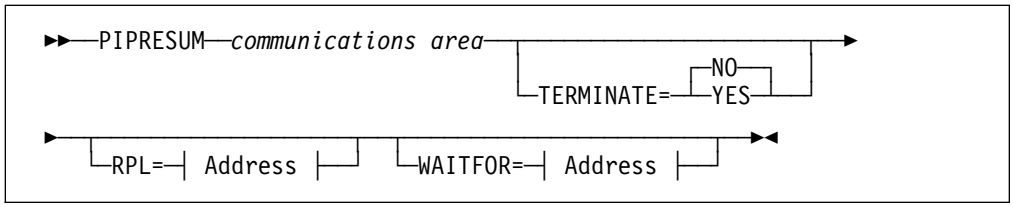
NEXT Specify the address of the next request parameter list in the chain.

FITTING Specify the identifier of the *fitting* stage you wish to reach. This is specified as the argument of the *fitting* stage where it is defined in the pipeline specification or in the ARGS= operand of the PIPSCSTG macro that defines the stage. Fitting names must be unique across the pipeline set. If you omit this keyword operand, you must store the fitting name into the RPL before you make it available to *CMS/TSO Pipelines*.

START Specify the initial state for the request parameter list. You can specify IDLE, READ or WRITE.

BUFFER Specify the data buffer for a WRITE request.

PIPRESUM—Resume the Pipeline



communications area — Specify the address of the PIPFTPRM macro that defines the communications area as the only positional operand. This operand is specified as an "Address" type. This operand is required.

TERMINATE — Specify YES to force the pipeline to terminate.

RPL — Specify the address of a PIPFTRPL macro. This RPL must be the first on the list of requests. Omit this operand to use the same RPLs as were used in the previous resume operation.

WAITFOR — Specify the address of an RPL that must be ready before the application can resume.

This macro requires a work area to build its parameter list. It assumes that the macro is issued within a PROC/PROCEND procedure. If you do not use these constructs, you must code the PIPRESUM macro expansion "by hand".

Chapter 13. Copipe Data Areas

This chapter shows the layout of the data areas that are used to communicate with the *CMS/TSO Pipelines* copipe support described in this part of the book.

The symbol "&P." stands for the prefix as specified in the PREFIX= operand of the macro that defines the data area. The default is the name of the macro.

PIPFTPRM—Communications Area

Figure 11. PIPFTPRM Data Area

Label	Offs	Len	Description
&P	0		Beginning of data area.
&P.LEVEL	0	1	Communications area level. Assembled as X'01'.
&P.REQTYPE	1	1	Request type. Flags.
&P.STATUS	2	1	Status of the pipeline. Scalar.
	3	1	Unused. Must be zero.
&P.RPL	4	4	First RPL on the RPL list.
&P.WAITFOR	8	4	An RPL that must be ready before the application can proceed. The bit &P.WAITRPL must be on to activate this facility.
	12	4	Unused. Must be zero.
&P.ANCHOR	16	4	The pipeline's anchor. Do not modify this field.
&P.RESUME	20	4	The resume entry point.
&P.AECB	24	4	Pointer to the pipeline's resume ECB.
&P.AWAITCOUNT	28	4	Pointer to a fullword where the pipeline maintains the count of stages that are waiting for external events. The application can wait on the ECB if the counter is positive. (It can also wait if this counter is zero, but then it will never wake up.)

&P.REQTYPE

Figure 12. Values for

Label	Val	Description
&P.NOSTART	01	Do not start the pipeline after it has been scanned. The scanner will resume the application without starting any stages. This bit can be set in the PIPFTPRM macro.
&P.TERMINATE	02	Terminate the pipeline. First signal all *fitting* stages to stop. If this does not terminate the pipeline, it is stalled. This bit can be set by the PIPRESUM macro.
&P.WAITRPL	04	The designated RPL must be ready or terminated before the application is resumed. This bit can be set by the PIPFTPRM macro and by the PIPRESUM macro.

&P.STATUS

Figure 13. Values for		
Label	**Val**	**Description**
&P.READY	0	The pipeline has run all stages that can run. *fitting* stages have run and changed state. Thus, productive work has been accomplished since the pipeline was resumed.
&P.DONE	1	The pipeline has completed.
&P.NO_DATA	2	No *fitting* stage changed state. No productive work has been accomplished since the pipeline was resumed. The application should inspect the RPL states and supply or consume data, or it should wait on the ECB provided (assuming the wait count is positive).

PIPFTRPL—Fitting Request Parameter List

Figure 14. PIPFTRPL Data Area			
Label	**Offs**	**Len**	**Description**
&P	0		Beginning of data area.
&P.NEXT	0	4	Pointer to next element.
&P.STATE	4	1	Current state.
&P.PREVSTATE	5	1	State last time the copipe was resumed. The application should not modify this field.
&P.EXITSTATE	6	1	State last time the copipe returned control to the application. The application should not modify this field.
&P.REASON	7	1	Reason code if the transition was rejected.
&P.FITTING	8	8	The identifier for the *fitting* stage with which this RPL should be associated. When the fitting has been resolved, the first word contains binary zeros; the second word contains a pointer. The application should not modify this field while the RPL is not idle.
&P.BUFFER	16	8	The first word contains a pointer to the associated data; the second word contains its length. The application sets this before setting the state to Write; the copipe sets this before it returns R.Done.

&P.STATE

Figure 15 (Page 1 of 2). Values for		
Label	**Val**	**Description**
&P.IDLE	0	Not active. Ignored by the copipe.
&P.ENDED	1	Not active. Ignored by the copipe.
&P.REJECT	2	Transition rejected by the copipe.
&P.WRITE	3	Application data are ready for the *fitting* stage to write into the pipeline.

Figure 15 (Page 2 of 2). Values for		
Label	**Val**	**Description**
&P.WRESOLVING	4	The copipe is waiting for the specified *fitting* stage to start. If this state persists, the fitting identifier is most likely misspelled.
&P.WRUNNING	5	Data in pipeline. The *fitting* stage is waiting for the output record to be consumed.
&P.WDONE	6	An output record has been consumed. The application can now provide another record.
&P.READ	7	The application is ready to accept data from the pipeline.
&P.RRESOLVING	8	The copipe is waiting for the specified *fitting* stage to start. If this state persists, the fitting identifier is most likely misspelled.
&P.RRUNNING	9	The *fitting* stage is waiting for a record to arrive at its primary input stream.
&P.RDONE	A	An input record is ready; its address and length are stored in the RPL. The application should move to state Read to consume the record in anticipation of reading another record.
&P.CONSUME	B	The *fitting* stage should consume the current input record and move the RPL to the idle state.
&P.TERMINATE	C	The *fitting* stage should terminate without consuming the current input record.
&P.TRESOLVING	D	Waiting for the *fitting* stage to start.

&P.REASON

Figure 16. Values for		
Label	**Val**	**Description**
&P.XITION_OK	0	No error.
&P.NOT_CAPABLE	1	The *fitting* stage is not at the place in the pipeline where it can perform the requested read or write operation.
&P.NOT_RESOLVED	2	The requested *fitting* stage is not yet started.
&P.INVALID_XITION	3	The *fitting* stage is in a state whence it cannot go to the one requested.

Part 5. Porting *CMS/TSO Pipelines* to other Environments

Chapter 14. System Services Vector

CMS/TSO Pipelines encapsulates the interface to some of the services of the underlying operating system in a layer of *system service routines*. For example, all requests for storage eventually reach the one routine that issues the appropriate storage management function.

Not only does this make it easier for the author to maintain *CMS/TSO Pipelines* for multiple platforms, it also enables vendors to support *CMS/TSO Pipelines* in their own operating systems by replacing these system services with their own code.

CMS/TSO Pipelines reaches the system service routines via the "system services vector". A user can replace parts or all of the system services vector by specifying the `sysv` parameter token. The token contains a pointer to a variable length vector of fullwords. This vector should be generated by the macro FPLSYSSV. The vector that is actually used by *CMS/TSO Pipelines* is constructed by merging the user specified vector with the current vector or with the system defaults, which are supplied by *CMS/TSO Pipelines*.

The first fullword of the system services vector contains the count of entries in the vector (excluding this first word). That is, a null vector consists of a single fullword of binary zeros.

Terminology

The system services vector is designed with the CMS multitasking model in mind, but *CMS/TSO Pipelines* makes no assumptions about there being processes and threads. In this chapter, the term "task" will refer to the individual dispatchable unit supported by the underlying host system.

A task is identified by a sixty-four bit number. The first fullword of this is arbitrarily called the process ID; the second fullword is equally arbitrarily called the thread ID. A task ID is a unique combination of process and thread IDs.

The term *preemptive task switching* describes a multitasking dispatcher that can switch control between tasks at arbitrary times, for example as a result of an interrupt. OS/390 uses such a dispatcher. The action of stopping a task temporarily is also called preempting it.

In contrast, both CMS Multitasking and *CMS/TSO Pipelines* use a *coroutine* style of dispatching; both switch between tasks only when a task performs an overt action that allows a task switch (such as suspending itself or creating another thread).

But *CMS/TSO Pipelines* supports being run by an operating system that uses preemptive task switching. Thus, it calls a locking service to serialise access to global resources.

On CMS, the PIPMOD command is used to initialise *CMS Pipelines*. This command is normally issued by the PIPE bootstrap module. On z/OS, the initialisation is performed as part of the first PIPE command (there is no separate bootstrap module).

Addressing Mode, Supervisor State, Enablement

CMS/TSO Pipelines runs in 31-bit addressing mode; branch entries to it must be in 31-bit mode.

CMS Pipelines switches to user key when it processes the PIPMOD command; and it switches back to key zero when it returns. The PIPE nucleus extension is a user nucleus extension; branch entries to it must be in user key. Both commands must be entered in supervisor state.

TSO Pipelines runs unauthorised; that is, in problem state and key 8.

CMS/TSO Pipelines itself does not change the supervisor/problem state nor does it enable interrupts, but REXX does; thus, any REXX stage will enable for interrupts. It is unspecified which built-in programs are in fact implemented as REXX programs, be that completely or in parts.

! In general, *CMS/TSO Pipelines* runs in primary space mode without referencing or
! changing the access registers; it is unlikely that it will work if it is entered in access
! register mode and the access registers do not all contain binary zeros.

! A few stages, such as *buffer*, *instore*, *outstore*, and *storage* supports an ALET operand to
! specify that data reside in the specified data space. As a result, these stages modify the
! addressing mode and access registers; they restore the addressing mode before calls to the
! pipeline dispatcher, but they do not save or restore the access registers.

Anchors

CMS/TSO Pipelines accepts, from the system service routines, a global anchor and an anchor for each task. These anchors are for the sole use of the system service routines; *CMS/TSO Pipelines* treats them as “magic cookies”.

The second word of the system services vector is the global anchor. The contents of this word (which are zero by default) are provided when *CMS/TSO Pipelines* calls a service and it does not know which task is associated with the call.

When *CMS/TSO Pipelines* knows for which task it is calling a service, it supplies the task’s anchor.

Nota bene: Whether *CMS/TSO Pipelines* knows is not necessarily the same as whether *CMS/TSO Pipelines* ought to know. Also note that *CMS/TSO Pipelines* does not serialise access to the anchors.

Register Conventions

When *CMS/TSO Pipelines* calls a system service, it supplies parameters in general registers 0 and up.

It usually supplies a standard type-1 save area of eighteen fullwords in general register 13, but see the Prologue and Epilogue services for exceptions. See also Enqueue and Dequeue.

It supplies the return address in general register 14.

It supplies the anchor (either the global one or the task one) in general register 15.

Note that *CMS/TSO Pipelines* provides no base register for the system service routine.

The system service routine must return with general registers 2 through 13 unchanged (except as noted for the storage management services). Where no return value is specified for general register 0 or 1, the contents of the registers need not be restored.

System Services

The various services are described in the following sections, grouped into the major categories,

- Task Management
- Storage Management
- Resource Management
- Exit Management

The first word of the heading for each section, which is the short name for the service, is also the name of the keyword operand that specifies the entry point for the service when you code the FPLSYSSV macro to build your system services vector.

Task Management

TaskID—Return Process and Thread ID

CMS/TSO Pipelines calls the TaskID service to obtain a unique identifier and an anchor for the current task. *CMS/TSO Pipelines* uses this information to determine whether or not a PIPE recursion is from a known thread by comparing the values with values returned previously. It also passes the information to the Resume service; it does not interpret or use the values returned in any other way.

Input:

R13 Standard type-1 save area.

R14 Return address.

R15 The global anchor.

Output: Registers 15 and 0 set to a unique identification of the task.

R0 The thread ID.

R1 The anchor associated with the task.

R15 The process ID.

Suspend—Suspend the Running Task

CMS/TSO Pipelines calls the Suspend service when it has no stages that can run and at least one of its stages is waiting on an ECB through the PIPWECB macro instruction. (The pipeline set would be stalled if no stages were waiting on an ECB.)

Input:

R1 The address of a CMS WAITECB parameter list (three doublewords), which is followed immediately by a standard OS ECB. The storage area is private to the task.

R13 Standard type-1 save area.

R14 Return address.

R15 The task's anchor.

Output: None.

Resume—Resume a Task

CMS/TSO Pipelines calls the Resume service when an asynchronous exit has posted a pipeline ECB on which a stage is waiting. Thus, the stage can resume after a PIPWECB macro. This service can also be called from a different thread as the result of a PIPPOST macro instruction.

Input:

R0 The address of a CMS WAITECB parameter list (three doublewords), which is followed immediately by a standard OS ECB (that is, the contents of general register 1 on the corresponding call to the Suspend service).

R1 The thread ID to be resumed. This was returned in general register 0 by the TaskID service.

R2 The process ID to be resumed. This was returned in general register 15 by the TaskID service.

R13 Standard type-1 save area.

R14 Return address.

R15 The task's anchor.

Output: None.

Storage Management

Storage management services are called in cases where no work area is available; that is, storage management is called precisely to allocate a work area. As a result, the two storage allocation services and the release service do not follow standard type-1 save area conventions.

There are three types of storage management services:

- Allocate and return task storage. Task storage is returned before the end of the task or process that allocates the storage.
- Extend task storage.
- Allocate and return permanent storage. Permanent storage is allocated by one task or process and possibly returned by some other process; or not returned at all, depending on the operating system. On CMS, dropping the PIPMOD nucleus extension causes all permanent storage to be returned; on ABEND recovery *CMS Pipelines* notes that its permanent storage is no longer allocated. On z/OS, permanent storage is recovered by the operating system when the job step task terminates.

When the task storage managers are called from prologue and epilogue code, general register 13 will point to the save area provided by the system. The service routines may

use general registers 2 through 6 as additional work space; the storage allocation routines must restore those registers they use from the save area before returning; the storage release macro does not need to restore those registers.

General registers 7 through 13 must not be modified.

The extend service and the permanent storage management services are never called from a prologue; as a result they use standard calling conventions.

Below—Allocate Storage Below the 16M Line

CMS/TSO Pipelines calls the Below service to allocate storage below the 16-megabyte line. This storage is typically used for a parameter list for a 24-bit system service (*e.g.*, a Data Control Block).

Input:

R0 The number of doublewords required.

R13 Standard type-1 save area. Registers have already been saved into this save area; thus, the service routine must not store into it. But it must restore registers 2 through 6 from this save area, if they are used as work registers.

R14 Return address.

R15 The global anchor.

Output:

R0 The number of doublewords actually allocated. This number may be larger than the number requested. Except when the Extend service has been used successfully, *CMS/TSO Pipelines* returns the amount of storage actually allocated (rather than the amount requested). When the storage is allocated as part of a buffer, *CMS/TSO Pipelines* may use any additional storage that the storage manager might provide.

R1 The address of the first byte of the storage area allocated.

R15 Return code. 0 when storage is allocated; 1 when storage is not available.

Above—Allocate Storage Anywhere

CMS/TSO Pipelines calls the Above service to allocate an area of storage that may be allocated anywhere within the address space. Other than for the location of the area allocated, the Above service is identical to the Below one.

Release—Return Storage

CMS/TSO Pipelines calls the Release service to return an area of storage that was previously allocated by the Below or the Above service. It never splits an allocated area; if the Extend service is provided and an area has been extended successfully, *CMS/TSO Pipelines* will return all storage in an extended area as a single unit.

Input:

R0 The number of doublewords to release.

R1 The address of the first byte of the storage area to release.

R13 Standard type-1 save area. Registers have already been saved into this save area; thus, the service routine must not store into it.

R14 Return address.

R15 The global anchor.

Output: None.

Extend—Try to Extend an Allocated Area

CMS/TSO Pipelines calls the Extend service to allocate additional storage at the end of an already allocated block of storage.

Input:

R0 The number of doublewords to add to the allocation.

R1 The address where storage should be allocated.

R13 Standard type-1 save area.

R14 Return address.

R15 The global anchor.

Output:

R0 The number of doublewords actually allocated. This number may be larger than the number requested.

R15 The return code. 0 means that storage has been allocated. 1 means that storage is not available; this may be because the storage area is already in use or because the underlying system does not support allocation of storage at a particular address.

Persistent—Allocate Persistent Storage Anywhere

CMS/TSO Pipelines calls the Persistent service to allocate an area of storage that must last longer than the pipeline specification. The storage may be allocated anywhere within the address space.

Input and output register conventions for the Persistent service is identical to the Above one. General register 13 points to a standard type-1 save area for use by the service.

ReleasePersistent—Return Storage

CMS/TSO Pipelines calls the ReleasePersistent service to return an area of storage that was previously allocated by the Persistent service. *CMS/TSO Pipelines* never extends persistent storage. It never splits an allocated area. Input and output register conventions for the ReleasePersistent service is identical to the Release one. General register 13 points to a standard type-1 save area for use by the service.

Resource Management

Enqueue—Serialise Access to a Global Area

CMS/TSO Pipelines calls the Enqueue service when it needs to update its global data area in a way that cannot be done atomically by the System/390 hardware. That is, the Enqueue service is called when *CMS/TSO Pipelines* needs write access to its global area, for example to update a value that is larger than a doubleword.

The Enqueue service must ensure that only one task can enqueue on a particular address at any one time; it can assume that the area will be enqueued for a short period. The next system service call in the task will be for the Dequeue service.

Input:

R1 The address of a thirty-two byte area of storage for use by the Enqueue service. The same area is used on all tasks when they access that particular area of storage. Thus, the system service routine can build a semaphore or some other synchronisation mechanism in this area. This area is initialised to binary zeros.

R13 Standard type-1 save area. The forward pointer at offset 8 of this save area points to a 104-byte work area (which can be considered a standard type-1 save area with a thirty-two byte extension).

R14 Return address.

R15 The global anchor.

Output: None.

Dequeue—Release Exclusive Access to a Global Area

CMS/TSO Pipelines calls the Dequeue service when it has finished updating a global area; the area is now available to be enqueued by other tasks.

Input:

R1 The address of the thirty-two byte area of storage that was passed to the Enqueue service to obtain exclusive access to the area.

R13 Standard type-1 save area. The forward pointer at offset 8 of this save area points to a 104-byte work area (which can be considered a standard type-1 save area with a thirty-two byte extension).

R14 Return address.

R15 The global anchor.

Output: None.

Exit Management

Timer—Set TOD Clock Exit

CMS/TSO Pipelines calls the Timer service to establish or cancel an asynchronous exit that is to be called at some future time. A call to the timer service cancels any previous request (that is, the operating system must not stack these requests). The exit must be driven once (unless it is cancelled).

Input:

R0 If general register 0 is zero, the existing exit must be cancelled (if it is still established). General registers R1 through R3 are not used.

If general register 0 is nonzero, it contains the address of the exit to be driven.

R1 A “user word” to be provided to the exit in general register 1 when it is called.

R2 The leftmost word of the TOD clock value at which time the exit must be driven.

R3 The rightmost word of the TOD clock value at which time the exit must be driven.

R13 Standard type-1 save area.

R14 Return address.

R15 The global anchor.

Output:

R15 The return code. 0 means that the exit is established. 1 means that the current time is later than the TOD clock value specified (that is, the event has already occurred).

Entry Conditions for the Exit:

R1 The "user word" supplied in general register 1 when the exit was established.

The exit will return with general registers 2 through 13 unchanged. The exit may call the Timer service to establish a new event. The operating system must not drive the timer exit recursively.

Programming Notes

Enqueue

The Enqueue service should make no assumption that all enqueues will be for the same area; but clearly it will suffice to enter a "must complete" mode which prevents any task switching.

Storage Management

The implementation may elect to treat requests for persistent storage the same as requests for Above. The default CMS system service routine does so. The default TSO system service routine allocates persistent storage from subpool 132.

Suspend and Resume

The programmer should note that *CMS/TSO Pipelines* cannot avoid a race between a call to the Suspend service and the corresponding call to the Resume service, even on systems that do not perform preemptive task switching.

Global Anchor

If the operating system wishes to use a nonzero global anchor, it may be easier to issue two PIPMOD commands. The first one should be for a vector of two fullwords, where the second one contains the global anchor. The second PIPMOD command should then reference the system services vector that has been generated by the FPLSYSSV macro with the TYPE=INLINE operand.

Issuing two PIPMOD commands assumes that *CMS/TSO Pipelines* can use the default storage management for its global data area. If this is not the case, the operating system must build a system services vector that contains the anchor as well as addresses of the system service routines.

If reentrancy is not required, the operating system can generate the system services vector with the TYPE=CSECT operand and then store the anchor at the address FPLSYSANCHOR (assuming the default prefix is used).

If reentrancy (or read only code) is required, the operating system must build the system services vector in dynamically allocated storage. It can use FPLSYSSV TYPE=DSECT macro to define the system services vector in its work area. It can then copy the system services vector expanded by FPLSYSSV TYPE=INLINE into this area and then store the global anchor.

The *CMS Pipelines* PIPMOD Command

On CMS, the PIPMOD nucleus extension supports being invoked with a call type of "program" and a parameter token list that contains a `sysv` token, which can optionally be followed by a `usex` token.

CMS Pipelines will use the system services vector specified for all storage management when initialised in this way.

The `sysv` token specifies the address of the system services vector override. This service vector is used on all requests that are not related to a pipeline set. It also supplies the default when a PIPE command is issued on a thread that has no active pipeline.

The `usex` token is valid only through this interface; it is four fullwords. It is supported only as the first call after the PIPMOD nucleus extension is established (be that by NUCXLOAD or SEGMENT LOAD) or immediately after CMS ABEND recovery. The second fullword should contain binary zeros; the last two fullwords specify the name of the nucleus extension to be set up to create a pipeline set. The `usex` parameter token serves the function that is normally accomplished by the PIPMOD INSTALL command.

Figure 17. Parameter Token Summary

Type	Word 1	Word 2	Word 3
`sysv`	Address of the system services vector.		
`usex`	0	First four characters of nucleus extension.	Last four characters of nucleus extension.

Merging the PIPMOD INSTALL Token

When the `sysv` parameter token is specified, the words in the vector to which it points override the existing system services vector. Each word can have these values:

0 The entry is not modified. The default or existing entry is retained.

-1 The *CMS/TSO Pipelines* default is requested. This retracts an exit that was previously established.

other The entry is modified to use the specified address as the entry point for the system service routine.

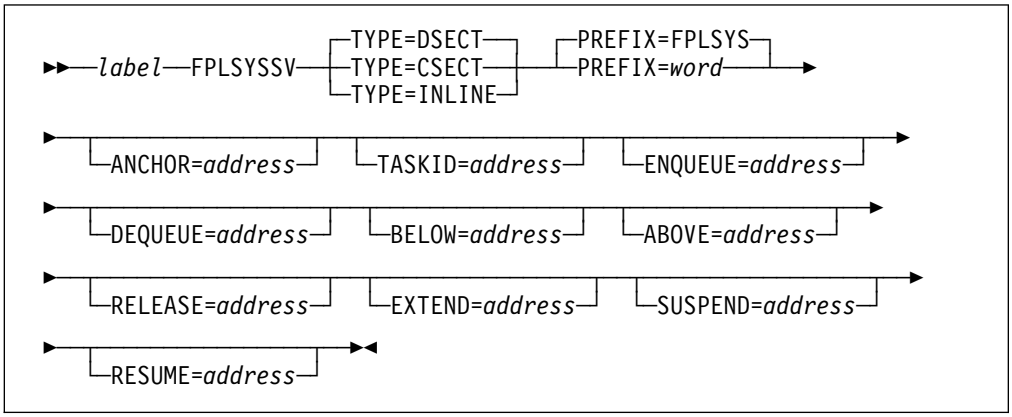
The vector specified with the PIPE command governs the pipeline set being created and recursions from within this pipeline set.

Macro

Use the macro FPLSYSSV to generate the services vector override that is specified in the `sysv` parameter token.

FPLSYSSV—Build a System Services Vector

The FPLSYSSV macro expands into a list of address constants. The TYPE keyword determines how the list of addresses should be generated.



label The label for the generated system services vector. This is required for TYPE=DSECT.

TYPE

- DSECT — The constants are in a DSECT and have labels attached. The *label* is used as name of the DSECT.
- CSECT — The constants are in the current section and have labels attached.
- INLINE — The constants are in the current section and have no labels attached. This is the recommended way of generating a system services vector.

PREFIX — The prefix to use when labels are attached to constants. The actual label is composed of the prefix and the keyword name for the routine. The unadorned prefix is generated for the count (the first word). You can specify a null prefix.

address Specify the label of the entry point for the routine; or zero to use the existing entry or default; or minus one to replace the existing entry with the default. Specifying minus one in effect retracts an existing override. The default for omitted keyword operands is zero.

Notes:

1. A certain amount of reasonableness is assumed in the use of the system services vector, but this is not enforced.
2. If the Enqueue service is provided, the Dequeue service must be provided as well.
3. If the operating system does not use the default storage management for the operating environment for which *CMS/TSO Pipelines* is generated, it must supply all four storage management services, or none.
4. Either both of the Suspend and the Resume services must be specified, or none.

Part 6. Sample Programs

Chapter 15. Hello World!

No collection of samples can be complete without the proverbial greeting. This example shows how to issue a pipeline that writes a literal string to the terminal as an encode pipeline specification.

```
PGMISAMP TITLE '        PIPE Command Programming Sample Program        *
                        COPYRIGHT IBM Danmark A/S'
                                                          EJECT
************************************************************************
*                                                                      *
* Hello, World.                                                        *
*                                                                      *
* Run an encoded pipeline to write a line on the terminal.             *
*                                                                      *
* Change activity:                                                     *
* 15 Nov 1995  New module by John P. Hartmann, DKIBMVM2(JOHN)          *
*                                                                      *
************************************************************************
                                                          SPACE 2
PGMISAMP CSECT ,
PGMISAMP AMODE ANY
PGMISAMP RMODE ANY
 DMSPDEFS ,                  Drag in system values
                                                          SPACE 1
 STM     14,12,12(13)
 BALR    12,0
 USING   *,12
 CMSCALL CALLTYP=PROGRAM,PLIST=PIPE
 L       14,12(,13)
 LM      0,12,20(13)
 BR      14
                                                          SPACE 1
 PIPTPARM PIPE,(ENCD,PSPEC),CMS=YES
                                                          SPACE 1
 PIPSCBLK PSPEC,TYPE=RUNPIPE,NAME='test1'
 PIPSCSTG TYPE=BEGIN
 PIPSCSTG TYPE=STAGE,VERB='literal',ARGS='Hello, World!'
 PIPSCSTG TYPE=STAGE,VERB='console'
 PIPSCSTG TYPE=DONE
                                                          SPACE 1
 END     PGMISAMP
```

Note the use of the DMSPDEFS macro, which defines several symbolic constants that are used by the other macros.

```
 load pgmisamp (start
►DMSLIO740I Execution begins...
►Hello, World!
►Ready; T=0.05/0.11 10:26:41
```

Chapter 16. Sample *spec* Function Package

This example shows a complete module. The function SUM takes 1 to 10 integer arguments and produces an integer result.

When this program is combined with FPLNXH and FPLNXG into a filter package (that contains no filters), *spec* will resolve the function name SUM to the function in this package (unless someone installed another filter package in front of this one).

```
FUN      TITLE '              User function test case for SPEC         +00010000
                              COPYRIGHT IBM Danmark ApS 2010'           00020000
 COPY    PGMID                                                         00030000
                                                    SPACE 2            00040000
********************************************************************** 00050000
*                                                                    * 00060000
* Test case  for a  filter package that  contains user-written       * 00070000
* functions for spec.                                                * 00080000
*                                                                    * 00090000
* Change activity:                                                   * 00100000
* 11 Apr 2010  New module by John P. Hartmann, CPHART(JOHN)          * 00110000
*                                                                    * 00120000
********************************************************************** 00130000
                                                    SPACE 2            00140000
FPLFUN   MODBEG FREETYPE=NONE                                         00150000
 DMSPDEFS VECTOR=R9,VECTOR2=YES                                       00160000
 FPLSPCTR PREFIX=C                                                    00170000
 COPY    FPLFUNTB                                                     00180000
 COPY    FPLFUNTE                                                     00190000
 FPLFUNTB ,                   Flags                                   00200000
 FPLFUNTE ,                   Define length                           00210000
                                                      EJECT            00220000
********************************************************************** 00230000
*                                                                    * 00240000
* Sum a series of binary numbers                                     * 00250000
*                                                                    * 00260000
********************************************************************** 00270000
                                                    SPACE 2            00280000
SUMIT    PROC  SAREA=HLLSTACK,ENTRY=NO,WORKBASE=R11                   00290000
 PBEGIN  ,                                                            00300000
 PIPVERV2 EP=SPVTI                                                    00310000
 LTR     R15,R15                                                      00320000
 PIPERM  1000,EXIT,COND=NOTPOSITIVE,SUB='FPLSPVTI'                    00330000
 LR      R7,R2                Move counter base                       00340000
 LR      R8,R3                                                        00350000
 LR      R6,R0                Result                                  00360000
 ZIP     R0                   Sum                                     00370000
 USING   C_SECT,R7                                                    00380000
 REPEAT  ,                                                            00390000
   CLI   C_SIGN,C_SIGNOMITTED                                         00400000
   CONTINUE COND=EQUAL                                                00410000
   CLI   C_SIGN,C_SIGNBIN                                             00420000
   PIPERM 728,EXIT,COND=NOTEQUAL,                                    +00430000
               SUB=('Not binary:',(C_SIGN,1,HEX))                     00440000
   A     R0,C_BINARY                                                  00450000
 UNTIL   INCR=(R7,C_LENGTH),BCT=R8                                    00460000
 LR      R7,R6                Get result                              00470000
```

```
 ST      R0,C_BINARY                                                    00480000
 MVI     C_SIGN,C_SIGNBIN                                               00490000
 DROP    R7                                                             00500000
 ZIP     R15                                                            00510000
 PEXIT   RC=(R15)                                                       00520000
 PROCEND ,                                                              00530000
                                                        EJECT           00540000
********************************************************************** 00550000
*                                                                    * 00560000
* Function table                                                     * 00570000
*                                                                    * 00580000
********************************************************************** 00590000
                                                      SPACE 2           00600000
 ENTRY   &MODULE.TB                                                     00610000
&MODULE.TB FPLFUNTB TYPE=CSECT                                          00620000
 FPLFUNTE SUM,SUMIT,RESULT=INT,MAX=10,                                 +00630000
               ARGS=(INT,(INT,OPT,SAME))                                00640000
 FPLFUNTN ,                                                             00650000
 MODEND  PRINT=GEN                                                      00660000
```

A gentle warning: You are not likely to succeed if you try just to pick a few lines out of the above program if your function deals with string arguments.

Generating the Filter Package

```
global txtlib fpllib
load fplnxh fplnxg fplfun
genmod funpack ( from fplnxh
```

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

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

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

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

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

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

```
IBM Corporation
Mail Station P3
2455 South Road
Poughkeepsie, New York 12601-5400
U.S.A.
Attention: Information Request
```

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Programming Interface Information

This book primarily documents information that is NOT intended to be used as Programming Interfaces of *CMS/TSO Pipelines*.

This book also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of *CMS/TSO Pipelines*. This information is identified where it occurs by an introductory statement to a chapter.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, or other countries, or both:

AT®	IBMLink	SQL/DS
BatchPipes®	IPDS	System/360
BookManager®	MVS	System/370
BookMaster®	MVS/DFP	System/390®
C/370	MVS/ESA	VM/ESA®
CT®	Notes®	VTAM®
DB2®	OS/390®	WebSphere®
DFSORT	Print Services Facility	z/Architecture
DRDA®	PROFS®	z/OS®
ECKD	RACF®	z/VM®
GDDM®	S/390®	
IBM®	SAA®	

Enterprise Systems Architecture/390®
Intelligent Printer Data Stream
Language Environment®

UNIX is a registered trademark of The Open Group in the United States and other countries.

Index

Special Characters

A

B

C

D

E

F

G

L

M

N

P

R

S

T

U

W

Program Number: 5741-A07

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

JH95-0068-03

Spine information:

CMS/TSO Pipelines PIPE Command Programming Interface *1.1.12*