SPANEX™

Span Service Routines Manual

Span Software Consultants Limited

Version: 06.0 Product Number: SPOS-001

Revision: 1st March 2015 Manual Ref: SPZ-03-017

CONTENTS Page

1 Introduction

This manual describes all currently available Span Service Routines.

Span Service Routines can be used from any language calling program but are primarily designed for use with assembler programs. Macros to aid in the use of these routines are provided for assembler-language users, and these are fully documented in the Span Macros Manual.

Span Manuals in this Series	Order No
Span Macros Manual	SPZ-02
Span Service Routines Manual	SPZ-03

1.1 Changes for Release 6.0

New options of the LINKAGE parameter of #SPANTRY allow the specification of the storage location of dynamic save areas.

New CPRIGHT2 option of the #SPZFLD macro for a mixed-case copyright notice value.

New format option for SPZPARSE fields that allows format-checking of character name values (ie alphabetic first character, followed by alpha-numeric characters).

All modules can now be located above the 16-Megabyte line.

1.2 Changes for Release 5.2

The #SPZFLD macro MOD option now supports dynamic change of the field offset in the output line.

1.3 Changes for Release 5.1

The #SPAMODE macro now supports the new 64-bit addressing mode introduced with z/OS.

The new BASE=NONE option of the #SPANTRY macro allows assembler programming to be performed without module base registers. Branching within the module is performed by relative jump instructions. This feature is supported by the #SPANXIT and #SPAMODE macros, and by a limited number of IBM macros.

The new CLEAR=YES option of the #SPANXIT macro clears a dynamic save area to zeros before releasing the storage. This ensures that sensitive information cannot be retrieved in response to a subsequent request for storage.

The new SETCONT= option of the SPZPARSE #SPZKWRD macro extends the number of values supported for the SET= parameter of keyword definitions. More possible values can thus be defined for parsing keywords.

1.4 Changes for Release 5.0

SPZPARSE now supports TSO- or IDCAMS-style command syntax, as an alternative to the traditional “keyword=value” style. Command and

parameter format is dynamically specified via a new parameter of the #SPZPARS macro.

The features provided in the #SPANTRY and #SPANXIT macros for MVS/ESA and OS/390 programming using the Linkage Stack may now be executed is ASC AR mode.

The #SPANTRY macro now allows dynamic save areas to be automatically set to binary zeros.

The new #SPZFIND macro is now required to be used instead of the standard FIND macro by users of the SPZQPAM service routine.

New DATE4 field type in the #SPZFLD macro supports for 4-digit year notation for Year 2000 compliance.

New SYSID field type in the #SPZFLD macro supports inclusion of the SMF System ID in program output.

New USERID field type in the #SPZFLD macro supports inclusion of the owning user ID in program output.

SPOUTPUT now supports the use of WTO tokens, to allow groups of console messages to be deleted with a single operating system DOM macro. This is implemented by means of the WTOTOKN option of the OPT= parameter of the #SPZSOB macreo, and by the WTOKEN= parameter of the #SPZSOB macro.

New version of the SOB used by SPOUTPUT eliminates all 3-byte addresses from the SPOUTPUT interface. All modules sharing a SOB must be assembled with the same version of the SPANEX macros, but SPOUTPUT supports existing modules compiled with all previous versions of the macros.

Minor editorial and technical changes have been made throughout this manual and the Span Macros manual.

1.5 Changes for Release 4.6

SPZSORT can now sort in-storage records up to 32767 bytes in length.

New features are provided in the #SPANTRY and #SPANXIT macros for MVS/ESA programming using the Linkage Stack.

The #SPANTRY macro now supports ESA-style module entry and exit linkage, and allows user-defined identification text to be added to a module entry point.

All invocation macros for Span Service Routine functions now support user programs running in AR mode on MVS/ESA systems.

New “Double-underline” feature is provided by SPOUTPUT formatting.

1.6 Changes for Release 4.5

The #SPAMODE macro now supports ARMODE switching for MVS/ESA systems.

The RUN=R4FSCR option of the #SPZSOB macro is now the default. Old-format full-screen output from SPOUTPUT is no longer supported for new programs. User programs assembled with Release 4.4 and earlier versions of the Span Macros can continue to use the earlier support - if these programs are assembled with Release 4.5 of the macros, the improved support will automatically be included.

The #SPZFLD macro now allows the screen high-lighting and colour attributes of fields on a 3270 screen to be modified dynamically.

SPOUTPUT now supports the BMARG= option of the #SPZSOB macro for full-screen output. This provides a "conditional end-of-screen" function to ensure that groups of output lines are kept together on the screen.

SPZDIRD now supports Partitioned DataSet Extended (PDSE) directories for MVS/ESA systems.

Support is now provided for versions of Span Software programs and user programs that are specific to the MVS/XA and MVS/ESA environments.

This page intentionally left blank.

2 SPOUTPUT Service Routine

2.1 SPOUTPUT Introduction

SPOUTPUT is a generalized routine for processing of program message output. It can be used to produce single-line messages, large-scale reports, and full-screen VDU displays. Output can be directed to a large range of destinations, and a given message may be sent to one or more destinations in a single invocation of SPOUTPUT. Some flexible facilities are also offered that enable the content of messages to be dynamically constructed, including some standard pieces of information that may be automatically included by SPOUTPUT.

2.2 Input to SPOUTPUT

SPOUTPUT is controlled by input parameters that are passed to it each time it is invoked by the user program. A single control block contains all the standard parameter information; this control block is known as the SOB (System Output Block), and is initialized and maintained by means of the #SPZSOB macro (see the Span Macros Manual). The #SPZSOB macro with the "TYPE=CALL" or the "TYPE=LINK" option is used to invoke SPOUTPUT, and this ensures that the correct parameter address is passed to the service routine. SPOUTPUT is invoked once for each output message line, and, depending on the output destination, the output will be issued immediately or may be buffered for later output (particularly for VDU terminals).

2.3 SPOUTPUT Message Formats

Messages to be output by SPOUTPUT can be presented to the routine in one of two formats. The format is selected by means of the "MODE=" parameter of the #SPZSOB macro, and the format remains the same until it is altered by a change to the alternative format. The address of either of these message formats is passed to SPOUTPUT at each invocation by means of the "MSG=" operand of the #SPZSOB macro.

2.3.1 Stream Mode

In stream mode, each message line is presented to SPOUTPUT as a single string of printable characters preceded by a single-byte length field.

eg

```
MSG       DC   AL1(19),C'THIS IS THE MESSAGE'
```

This format is generated by the #SPMSG macro, and effectively implies that the user program performs all the message data formatting before calling SPOUTPUT.

2.3.2 Insert Mode

In insert mode, each message line is presented to SPOUTPUT as a series of message field descriptions. Each field description specifies text that is supplied by the calling program, or a code requesting SPOUTPUT to generate certain standard information (eg current time, date, jobname, etc) to be included in the message. The #SPZFLD macro (see Span Macros Manual) is used to build message field descriptions, and this macro also allows field attributes to be defined (particularly when output is destined for a 3270-type terminal).

Insert mode is required for the definitions of page titles, sub-titles and footings, and for screen headings. The #SPZSOB macro parameters that define these title lines assume that the area containing title data is formatted by means of #SPZFLD macros.

If user-supplied text is to be included in a message field, this text can be provided in three different ways: (1) the text may be "hard-coded" by specifying it on the initial coding of the #SPZFLD macro; (2) the text may be filled in at execution time within the expansion of the #SPZFLD macro itself (a user label may optionally be specified on the #SPZFLD macro to permit this); (3) the text may be obtained from another user area by indirect addressing. Indirect addressing is specified on the #SPZFLD macro when the "INDIRECT" option is specified, and is forced when any of the colour or extended high-lighting options are specified for full-screen output (see below), or when any of the automatic field position adjustment options are specified. Indirect addressing permits the content and length of the text to be dynamically modified by means of the "MOD"-type executable form of the #SPZFLD macro.

2.4 SPOUTPUT Output Destinations

One or more message output destinations should be selected by means of the #SPZSOB macro when calling SPOUTPUT. These destination options are set by means of the "OPT=" or "OPT2=" parameters of this macro, and remain set until altered by re-specification. The various output destination options are described below. Output sent to any physical destination can be processed by one of a series of user exit routines, one for each output type, that may be specified by the user on the #SPZSOB macro.

2.4.1 PRINT

The output is directed to a printer-type listing, typically a "SYSOUT" file, whose DDNAME is specified by the "DDNAME=" operand of the #SPZSOB macro. Page control is handled automatically by use of the "LINECNT=" operand of the #SPZSOB macro. Print line width defaults to 132 bytes, but any value up to 254 bytes may be specified by means of the "PRTCOLS=" operand of the #SPZSOB macro, before the first output data is generated. Page headings, sub-headings and footings are automatically added, if required, and are defined by means of the "TITLE=", "SUBTITL=" and "FOOTING=" operands, respectively, of the #SPZSOB macro; headings, sub-headings and footings may each contain any number of output lines. Page skipping can be unconditional by the use of the "PAGE=" operand of the #SPZSOB macro, or conditional on the number of lines

remaining on the current page ("BMARG=" operand). By default, all printed output, including titles, subtitles and footings, will be converted to Upper Case characters before being printed by SPOUTPUT. This function can be disabled by means of the "PRTFOLD=NO" option of the #SPZSOB macro. Printed output is never converted to Upper Case before being passed to the print User Exit (User Exit Type 1).

2.4.2 TPUT

The output is sent to a TSO user, as specified by the "USERID=" operand of the #SPZSOB macro, or to the current TSO user if the "USERID=" operand is omitted. This output option is ignored if the user program is not running in TSO foreground.

2.4.3 FULLSCR

The output is formatted into 3270-type terminal pages and sent to a VDU, either TSO (by default) or to any other TP system if a user exit routine for full-screen output is supplied. Screen headings can be supplied by means of the "HEADER=" operand of the #SPZSOB macro; any number of heading lines may be specified. If TSO is being used and the terminal is not a VDU, this option is the same as "TPUT". If there is no full-screen user exit routine, this output option is ignored if the user program is not running in TSO foreground. See the discussion of full-screen processing below.

2.4.4 PUTLINE

The message is sent to the invoking TSO user by means of the PUTLINE TSO service routine. This has the advantage over TPUT of being effective when the TSO Terminal Monitor Program (TMP) is run in the background (MVS only). However, the TSO Command Processor Parameter List (CPPL) address must be supplied via the "CPPL=" operand of the #SPZSOB macro, and this address is known only to a program invoked as a TSO Command Processor.

2.4.5 SEND

The message is sent to the TSO user specified by the "USERID=" operand of the #SPZSOB macro via a "SEND" operator command with the "LOGON" option. Note that this is an authorized function, and will only be effective if the user program is an authorized program running under the control of Span Software's SPANEX program product. See the SPANEX General Usage Manual, Span Product Program section.

2.4.6 WTO

The output is sent to the console operator via a WTO (SVC 35), formatted as indicated by the "MSGTYP=" and "ROUTE=" operands of the #SPZSOB macro. If the "RUN=TEST" operand of the #SPZSOB macro is specified, the WTO will be suppressed.

2.4.7 WTONDEL

This option is similar to the WTO option, except that Descriptor Code 2 will be set, marking the message non-deletable on graphic consoles. Register 1 on return from the SPOUTPUT service routine will contain the WTO "MSG ID" to be used to "DOM" the message (see the Operating System Assembler Services manuals for a full description of the WTO facility).

2.4.8 WTOTOKN

This option is similar to the WTO option, except that additionally a WTO token may be associated with the message. The token is specified by means of the "WTOKEN=" operand of the #SPZSOB macro, and allows the message (or a group of messages issued with the same token) to be deleted from the operator console by means of an operating system DOM macro. If the message is to be non-deletable, the WTONDEL option must be specified in addition to the WTOTOKN option. See the Operating System Assembler Services manuals for a full description of the WTO facility.

2.4.9 WTOR

This option is similar to the WTO option, except that a WTOR (SVC 35) is issued instead of a WTO, and a reply will be required from the operator. The "WTORECB=", "REPAREA=" and "REPLEN=" operands of the #SPZSOB macro should be used to specify the other parameters required to define a WTOR request to the system (see the appropriate Operating System manual for a full description of the WTOR facility).

2.5 SPOUTPUT Logging

A log (copy) of all messages sent to any destination will be written to the SPOUTPUT Log DDNAME (default is "SYSLOG", but any valid DDNAME may be used by specification on the #SPGLBL macro) as long as the "LOG=YES" option of the #SPZSOB macro is in effect. All page headings and footings sent to the PRINT dataset will also be logged, but the log dataset will not be paged correctly if it is printed. Also the log dataset is limited to a line length of 132 bytes, and print output of a greater line length will be truncated.

2.6 SPOUTPUT Full-Screen Output

SPOUTPUT provides powerful facilities for full-screen output to 3270-type terminals. Full-screen output for TSO is built-in to SPOUTPUT, but full-screen output via any TP system can be implemented by coding a user exit routine to handle the actual input/output operations. SPOUTPUT will provide this user exit routine with a complete 3270 data stream, including all terminal buffer address characters, attribute bytes, etc. The specification of this user exit routine is controlled by means of the "RTN4=" parameter of the #SPZSOB macro.

The remainder of this discussion of full-screen output assumes that either TSO or a valid user exit routine is being used for screen input/output operations.

Both stream mode and insert mode are supported for full-screen output, but screen display attributes can be specified only for insert mode. Note that, if screen headings are specified by means of the "HEADER=" parameter of the #SPZSOB macro, then each line of the heading must be defined in insert mode. Default screen field attributes are unprotected, high-lighted for screen headings; and unprotected, non-high-lighted for other data.

The full range of supported field attributes is described in the definition of the #SPZFLD macro in the Span Macros manual.

The implementation of full-screen formatting was changed significantly with release 4.0 of the Span Macros and Services package. Prior to this release, each line of display on the screen was composed of a single field on the 3270, regardless of how many #SPZFLD macros comprised each line of data. This mode of output is still supported for compatibility reasons, but it is not now recommended and will be withdrawn in a future release. The new implementation permits the definition of multiple fields on each screen line, and allows very flexible definition of colour and extended high-lighting attributes as well as traditional 3270 functions.

In order to ensure that SPOUTPUT uses the new field formatting technique, the "RUN=R4FSCR" option of the #SPZSOB macro should be specified. This is now the default, and is forced for any module assembled with Release 4.5 or later of the Span Macros. If there are any user programs using the old format macros (#SPSOB, #SPFIELD) these must now be adapted to use the new options. Note that if more than one "RUN=" option is required, all options must be specified each time a #SPZSOB macro is executed that specifies any "RUN=" parameter, although the R4FSCR option now never needs to be specified.

In order to invoke SPOUTPUT full-screen formatting, "OPT=FULLSCR" should be specified on the #SPZSOB macro. It is important to note that, because of the buffering technique used by SPOUTPUT, it is necessary to issue a CALL- or LINK- type #SPZSOB macro with the parameter "OPT=(FULLSCR,CLOSE)" after the last output line has been sent, in order to flush the last screen buffer out to the terminal.

Support is provided in SPOUTPUT for the 3270 Extended Data Stream, which is used when extended high-lighting or colour features are being used. Any of these attributes may be specified on the #SPZFLD macro, and the appropriate screen commands will be generated by SPOUTPUT. If a user program is to be written that needs to be used with 3270 screens that do not support these extended functions as well as with those that do, then the "RUN=NOEDS" option of the #SPZSOB macro should be used when the Extended Data Stream is not required. Use of this option is then the only difference in the user program between the two types of terminal, and SPOUTPUT will not generate extended feature commands if the "NOEDS" option is in effect.

In order to obtain the maximum benefit from colour terminals, it is desirable to avoid the overhead on the 3270 display of the definition of "attribute bytes". These bytes, a legacy of early designs of 3270 terminal, occur on the screen every time that there is a change of "attribute" of the displayed data, and appear to the

user as a blank character that cannot be typed over. Extended data stream 3270 terminals can avoid most of these attribute bytes within data fields by the use of a series of new screen control commands. SPOUTPUT will support screen displays with changing screen attributes, colours and high-lighting options without attribute bytes, and the use of this feature is specified by means of the "CONTIN" option of the #SPZFLD macro. Any data field specified by a #SPZFLD macro with the "CONTIN" option is taken by SPOUTPUT to be a continuation of the previously-defined field, and no attribute byte is generated at the start of the field. However, all colour and extended high-lighting specifications are honoured, so that any change may be made to the appearance of the display. A #SPZFLD macro specifying "CONTIN" but with no 3270 attributes specified will cause the display to revert to the attribute values defined by the previous #SPZFLD macro in the message that did not specify "CONTIN". A #SPZFLD macro may be generated as a "TEXT" field but with zero length (specified by means of a "LENGTH=0" parameter) if it is required only to change or reset screen attributes. Note that a field specified with "CONTIN" does not necessarily have to be adjacent to the previously specified field; "CONTIN" can even be specified for the first #SPZFLD macro in a line of output data, in order to inherit the 3270 field from the previous output line, but the calling program is then responsible for ensuring that screen "page" breaks occur at the correct point. Attributes will not be maintained across a screen page break if the first output #SPZFLD macro for the new screen page specifies "CONTIN". Screen paging can be forced by means of the "PAGE=THROW" option of the #SPZSOB macro. Note that standard 3270 attributes (eg high-intensity, protected) will be ignored for #SPZFLD macros with the "CONTIN" option.

SPOUTPUT requires that each "message" or call to SPOUTPUT should represent a single screen output line. The right-hand end of an output line may be lost if it extends beyond the physical edge of the screen and wraps around to the following line, as it will be over-written by the next output line generated by SPOUTPUT; if the calling program requires to send an output line that is wrapped from one screen line to the next, it must split the data and call SPOUTPUT twice. It should also always be remembered that SPOUTPUT automatically adjusts its output to fit the dimensions of the screen that is in use. Related groups of lines can be kept together across screen page breaks by the use of the BMARG= option of the #SPZSOB macro.

2.7 SPOUTPUT User Output Exit Routines

SPOUTPUT supports user exit routines to perform actual input/output operations, if this is required. If no user exits are defined, SPOUTPUT will perform all its own I/O to standard destinations.

There are currently six user exit routines available, of which two are undefined. The other routines are shown below:

Exit Type 1	used for print output. This routine is called once for each line of print output, and once for each line of heading or footing data. Each output line is up to 254 bytes long, and the first character is an ASA printer control character. Output is not converted to Upper Case before being passed to this user exit - the exit must perform this function if it is required.

Exit Type 2	used for WTO/WTOR output. This routine is called for each WTO or WTOR message to be issued. The output line passed to this routine is a List-form WTO or WTOR macro, in exactly the same format as required by the Operating System SVC 35.
Exit Type 3	used for line-mode terminal output (a single line message at a time). This routine is called each time SPOUTPUT is requested to perform a TPUT or PUTLINE output operation. The output line passed to this routine is formatted, and the length of the data is passed separately.
Exit Type 4	used for full-screen mode terminal output. This routine is called each time SPOUTPUT has constructed a full screen's worth of data in its buffer, or when the user program has forced a new terminal page. The routine is passed a buffer containing a complete 3270 data stream, and starting with an "ESC,WRITE,WCC" sequence (see 3270 programming manual).

Note that all of these user exit routines may be called for other reasons than simply to perform output. These reasons are identified in the parameters passed to the routines by SPOUTPUT, which are detailed below.

User output exit routines parameter list

On entry to any user output exit routine, register 1 will point to the following parameter list:

+0	Address of SPGLBL parameter block (mapped by #SPGLBL macro)
+4	Address of user SOB for this output (mapped by #SPZSOB macro)
+8	Address of message to be output
+12	Length of message to be output
+16	Single-byte option request from SPOUTPUT:

- 0 Reserved
- 4 Get VDU screen size (Exit Type 4 only)
  exit returns: R0 = lines/screen
  R1 = characters/line
  R0 = 0 if not a VDU
- 8 Issue message (normal output request)
- 12 Issue CLOSE for this output destination
- 16 Clear screen (Exit Type 4 only)
- 20 Issue OPEN (Exit Type 1 only)
  exit returns R15 = 8 if open failed - Type 1 exit will not be called again

Return codes from exit routines:

R15 = 0	Successful output, SPOUTPUT will not perform any physical output
R15 = 4	Output not done by exit routine, SPOUTPUT will perform normal output technique
R15 = 8	(returned from OPEN request only) OPEN failed, bypass all future print output

2.8 Invoking SPOUTPUT

SPOUTPUT may be invoked by means of a CALL (or branch instruction) or by means of an Operating System LINK (SVC 6) operation. Both of these options are selected by means of the #SPZSOB macro.

SPOUTPUT is re-entrant, but each task must supply its own SOB if it is to be used re-entrantly. The CSECT SPGLBL must be available to the SPOUTPUT routine; it may either be link-edited with SPOUTPUT, or, if this is not done, SPGLBL must be available as a separate load module at execution time. If SPGLBL is link-edited with SPOUTPUT, the resulting load module is not re-entrant. The SPGLBL CSECT is obtained by assembling the #SPGLBL macro.

2.8.1 CALL

The #SPZSOB macro should be used, with the "TYPE=CALL" operand specified. Optionally, the entry point address of SPOUTPUT can also be specified by means of the "OUTEP=" parameter. If this is not done, the #SPZSOB macro will generate a V-constant for SPOUTPUT, which must then be link-edited together with the user program; if "OPT=SEND" is to be used, module SPSEND must also be included in the load module. For user programs designed to be run under SPANEX, the "OUTEP=SPXMOUTR" parameter should be specified to make use of the common copy of SPOUTPUT. Note that SPOUTPUT must reside below the 16-Megabyte line in MVS/XA and MVS/ESA systems. SPOUTPUT may be entered in 31-bit addressing mode if required.

2.8.2 LINK

A separate load module of SPOUTPUT must exist (including the SPSEND module).

The #SPZSOB macro should be used, with the "TYPE=LINK" operand specified, to invoke SPOUTPUT. This option may also be used by user programs running under SPANEX, and the LINK will be intercepted and the SPANEX copy of SPOUTPUT will be used. Note that SPOUTPUT must reside below the 16-Megabyte line in MVS/XA and MVS/ESA systems. SPOUTPUT may be entered in 31-bit addressing mode if required.

2.9 DCB Information

2.9.1 PRINT

The print DCB (default DDNAME "SYSPRINT") has the following default attributes:

```
RECFM=FBA,LRECL=133,BLKSIZE=1330
```

If the print line width is varied by means of the "PRTCOLS=" option of the #SPZSOB macro before the print DCB is opened, then the DCB attributes will be:

```
RECFM=FBA,LRECL=(prtcols+1),BLKSIZE=nnn
```

where nnn is the largest multiple of the LRECL that is less than or equal to 6233. If a block size is specified via JCL or if SPOUTPUT print output is directed to an existing dataset, then the block size will be honoured, provided that it is an exact multiple of the record length generated as a result of the PRTCOLS option (ie prtcols+1). Otherwise the block size will be overridden by SPOUTPUT to be the largest multiple of the LRECL that is less than or equal to 6233.

2.9.2 LOG

The log DCB (default DDNAME "SYSLOG") has the following attributes:

```
RECFM=FB,LRECL=132,BLKSIZE=1320
```

2.10 SCP Compatibility

SPOUTPUT will support all of the Operating Systems supported by Span Service Routines without modification or re-assembly, as dynamic checks are included to ensure compatibility. If SPOUTPUT is to be re-assembled, however, as with all Span Software Products, it is necessary to set the Global symbol &#SCP in the #SPXGLOB source module to the SCP that is to be used. If SPOUTPUT is to be run on an MVS/XA or MVS/ESA system, it must reside below the 16-Megabyte line.

2.11 SPOUTPUT Messages and Codes

SPOUTPUT does not produce any messages of its own, and merely issues the data passed to it by its calling program.

SPOUTPUT can, however, issue an Abend User 99, and this occurs when an invalid user SOB control block is passed as a parameter. If all invocations of SPOUTPUT are made by means of the #SPZSOB macro, this is unlikely to occur in normal circumstances.

This page intentionally left blank.

3 SPZDIRD Service Routine

3.1 SPZDIRD Introduction

The SPZDIRD service routine reads all entries in a Partitioned DataSet (or Partitioned DataSet Extended) directory into a virtual storage table. Two output formats are available:

fixed - the calling program requests a constant number of user halfwords of directory information for all members, regardless of the quantity of data actually held in the directory;

variable- each directory entry is returned in its entirety.

For both of these format options, the directory data may be obtained in the form in which it is physically held in the directory, or in the form in which directory data is returned by the BLDL macro (with two extra bytes inserted after position 11 of each directory entry - the first of these bytes indicates the concatenation number of the library from which the entry came, and the second byte is always zero).

If multiple Partitioned DataSets are to be read, that are concatenated together to a single DD statement, then the member lists may be optionally sorted and merged together by SPZDIRD; in this case duplicate member names are removed and only the first occurrence of a duplicate is returned to the calling program. SPZSORT is used to perform the sort operation, and must always be link-edited together with SPZDIRD. When datasets are concatenated together, any combination of PDS and PDSE datasets may be used.

The storage area which is to contain the directory is dynamically obtained by SPZDIRD, and the calling program is responsible for FREEMAINing it when it is no longer required.

3.2 SPZDIRD Input and Invocation

SPZDIRD requires a 3- or 4-word parameter list, as shown below:

Word 1 - the address of the 8-byte area containing the DDNAME of the PDS or PDSE whose directory is to be read. Note that concatenated input datasets are supported, but that if a DDNAME is supplied that refers to a series of concatenated PDSs and/or PDSEs, directory entries will be read from the first data set only unless the concatenation option is set in Word 4 of the parameter list (see below).

Word 2 - the address of a fullword containing the number of user data bytes to be read for each directory entry (for fixed-format directory - note that a number greater

than the actual length of a directory entry may be specified), or -1 (for variable format directory).

Word 3 - used by SPZDIRD to return the in-storage directory (see section 3.3, 4.3). This word must be initialized to -1 (X'FFFFFFFF') if the fourth (optional) word of the parameter list is present, and must be set to any other value if a 3-word parameter list is supplied.

Word 4 - contains a series of option flags:

X'40000000'	specifies that support is required for concatenated input datasets;
X'20000000'	specifies that, if concatenated input datasets exist, the member lists are to be sorted and merged together, with duplicate member names discarded. Note that, for this option, fixed-format output must also be specified; this option will be ignored if variable-format output is requested (see Word 2 of the parameter list);
X'10000000'	specifies that directory data is to be returned in "BLDL" format, ie each entry contains member name (8 bytes), TTR (3 bytes), Concatenation number (1 byte), zero (1 byte), user data (0-n bytes). Without this option, the concatenation number and zero bytes are omitted, and, for concatenated input datasets, there is then no means for the calling program to determine which dataset contains which members;
X'08000000'	specifies that the returned directory table is to be built above the 16-Megabyte line (MVS/XA or MVS/ESA systems only). The calling program must be running in 31-bit mode if this option is used. The maximum size directory table supported is 16 Megabytes. This option is ignored if it is used in a non-XA MVS system.

SPZDIRD may be invoked via CALL or LINK, with register 1 pointing to the above parameter list. SPZDIRD must reside below the 16-Megabyte line in MVS/XA and MVS/ESA systems.

3.3 SPZDIRD Output

Output from SPZDIRD is a variable-length block of storage with the following format:

Word 1 - number of entries in the directory

Word 2 - length of this output block (length to be FREEMAINed when no longer required).

Word 3 to end - actual directory entries. Note that the last directory entry in the list has a member name field of 8X'FF', and signifies the end of the directory.

This block of storage is pointed to by Word 3 of the input parameter list on exit from SPZDIRD.

See the description of the SPZQPAM service routine in this manual for an example of the use of SPZDIRD.

3.4 SPZDIRD Return Codes

SPZDIRD will return one of the following values in Register 15:

0 - normal completion.

4 - insufficient storage available to hold the directory. Increase the region or partition size.

8 - there is a format error in a directory, eg imbedded file mark.

12 - there is no DD statement for the DDNAME supplied.

16 - an extent violation occurred while reading a directory.

20 - an I/O error occurred while reading a directory.

24 - one of the datasets referred to is on a non-DASD device.

For return codes 4 and 8, the directory that has been read up to the point of error is returned to the calling program; for return codes 12 to 24, no directory information is returned.

This page intentionally left blank.

4 SPZFDUMP Dump Output Formatting Service Routine

4.1 SPZFDUMP Introduction

SPZFDUMP is a simple routine that formats a print line in standard "dump" output format. Each call to SPZFDUMP produces a print line that represents 32 bytes of data, in a format that includes the main storage address or offset, a hexadecimal dump of the data (grouped into blocks of four bytes for readability), and a clear-text representation of the data.

This facility can be very useful in all programs that need to display blocks of data in a conveniently readable format.

4.2 SPZFDUMP Input and Invocation

SPZFDUMP requires a 4-word parameter list, as shown below:

Word 1 - the address of a 32-byte area that contains the data that is to be formatted for printing.

Word 2 - the address of the output area which will contain a print line after return from a call to SPZFDUMP. This area must be a minimum of 117 bytes long. The formatted print line that is returned contains 117 characters of data, and does not include any print control character.

Word 3 - the address or offset to be printed in the left-hand column of the output data. For example, this can be a virtual storage address (for a program that dumps virtual storage), or can be an offset value (for a program that is formatting the contents of a buffer for display. The value in this word will typically increase by 32 for each call to SPZFDUMP in a program that is formatting an area of more than 32 bytes of data for printing.

Word 4 - the length of data to be formatted. A value in this parameter of 0 or 32 is interpreted to mean that a full print line formatting 32 bytes of data is to produced. A value of less than 32 is used for the last line of data, if the total data length is not a multiple of 32 bytes. The final print line formatted in this case will be partially filled in, according to the number of bytes that remain to be printed.

4.3 SPZFDUMP Output

Output from SPZFDUMP is the output area addressed by Word 2 of the parameter list. This area contains a print line of 117 characters, that may be output by the calling application.

SPZFDUMP produces no messages or return codes. Any abend within the SPZFDUMP module will be as a result of invalid parameters.

5 SPZPARSE Parameter Analysis Service Routine

5.1 SPZPARSE Introduction

SPZPARSE is a generalized facility for the analysis and decoding of control and parameter information from any source. SPZPARSE can be passed a string of parameters to be analysed, or can itself perform all I/O operations on Sequential or Partitioned datasets, or from CA-LIBRARIAN or CA-Panvalet libraries. SPZPARSE can be invoked by any program, foreground or background, to analyze any positional, keyword, keyword=value or keyword(value) parameters.

The SPZPARSE facility comprises a set of assembler-language macros and the SPZPARSE module itself. The macros may be coded in any assembler language program, and create tables defining all commands, parameters and operands that the user program wishes to accept as input. SPZPARSE interprets, verifies and converts user parameter data according to the requirements of the user program, any errors being reported by error messages. The parameter data may be obtained either by SPZPARSE from an input dataset, or may be provided as an in-storage string by the calling program.

Thus, any assembler program that requires input parameter information (eg a user-written utility program) need not include code to read and interpret parameters or control statements - it need use only the Span Software SPZPARSE Service Routine macros. This considerably reduces the quantity of tedious and error-prone coding for such programs. The SPZPARSE macros (#SPZPARS, #SPZCMD, #SPZKWRD, #SPZPEND, #SPZPMAP, #SPZPOSN, #SPZSUBP) are documented fully in the Span Macros manual.

5.1.1 Example of Use

A program is to be written that is designed to performs the functions of the IBM utility IEBUPDTE. One input record is required to define the member of a partitioned dataset to be updated. This record may be of the form:

```
./ CHANGE NAME=xxxx,LIST=ALL,SEQFLD=nnn
```

The writer of this utility must provide code to interpret these records, and any other permissible control statements. Perhaps the “NAME” value is to be stored in a field named “INNAME”, the sequence field value is to be set up as a packed field named “INSEQ”; if “LIST=ALL” is specified, bit 0 of a field named “INOPTS” is to be set on. Of course other types of input record must also be processed, but for the command type “CHANGE” the following code in the user program will completely process the input data using SPZPARSE:


```
            #SPZPARS EODAD=END00,ERROR=ERR00,DATA=DAT00,
                      SOB=USRSOB,MF=(E,INTL)
                   .
                   .
                   .
            #SPZCMD  CHANGE,RTN=CHG00
            #SPZKWRD NAME,FIELD=INNAME,REQD=Y
            #SPZKWRD LIST,FIELD=INOPTS,SET=(ALL,OI,X'80')
            #SPZKWRD SEQFLD,FIELD=INSEQ,FMT=P
            #SPZPEND
                   .
            #SPZSOB  TYPE=STATIC,NAME=USRSOB
INTL        #SPZPARS MF=L
                   .
                   .
INNAME      DS   CL8
INSEQ DC    PL4'0'
INOPTS      DC   X'00'
```

Note that the input record will be validated, and any relevant error messages will be produced. If the information was continued onto two or more records, these are read and interpreted before control is returned to the user program. In the coding example above, control would be given to routine "CHG00" if a "CHANGE" command was found, to routine "END00" at end-of-file on the input dataset, to routine "ERR00" if an error is encountered in the input, and to routine "DAT00" if a data record (non-command) is read.

5.1.2 Definitions of Terms

```
./NEXT      FIND      ABC,LINE=(2,3),LIST=YES,ALL
```

Records in the format shown in this example can be considered to consist of several parts.

The characters "./" are termed the "identifier", and identify this record as being a control record.

"NEXT" occupies the "label" field

"FIND" is the "command"; it defines the function to be performed as a result of this control statement.

"ABC" is a positional parameter, and "ABC" is the value of the parameter rather than a specific keyword.

"LINE" and "LIST" are "operands", and are of the form "keyword=value". Eg "LIST" is the keyword, and "YES" is the value.

The characters "2" and "3" occupy "sub-parameter" positions 1 and 2, respectively, of the operand "LINE".

"ALL" is a keyword operand, but one which does not require a value to be assigned to it.

SPZPARSE handles data that contains any or all of these components. The use of identifiers and labels is optional, and rarely used. Also a null command is permitted, enabling parameters that are not command operands to be processed. Considerable flexibility is allowed in the format of data input to SPZPARSE in the areas of start columns, continuation statements, and in the use of quotation marks and parentheses.

5.2 SPZPARSE Standard Facilities

The programmer using SPZPARSE specifies the commands, parameters and keywords that he wishes to accept by means of macro statements in his program. These statements also specify the fields in his program that will receive the parameter values input for the various operands, any data format conversion that is to take place, and the address of the routine in the user program that is to be given control for each different command being processed.

An executable macro statement defines the options required for the printing of the input records as they are processed, whether data records and/or labels are permitted, and identifies the set of definition macros that is to be used for interpretation of the input. Output of the parameter information, and of any generated error messages, is performed by the Span Software SPOUTPUT Service Routine, and the user program may pass the address to SPZPARSE of the SOB to be used for this purpose (see the description of the SPOUTPUT routine elsewhere in this manual).

The user program executes a macro which is equivalent to a READ statement (if an input dataset is to be processed). This macro returns control to the routine in the user program that will handle the function required by the command found in the control information or control statement, or to the routine that will handle a specific condition (eg data record, end-of-file). All the parameter values for the command specified will have been interpreted, checked for validity, converted to binary, packed decimal, hexadecimal or a bit setting as necessary, expanded to the correct length and stored in the pre-determined fields in the user program. Error messages will have been produced for any values found to be in error, or for missing or undefined keywords and commands, and the control card will have been printed.

5.2.1 Coding Example

A program is to accept a “CALL” control statement, which must specify a module name as a parameter. This module name is to be expanded to fill a field 8 bytes in length. Optionally, a “TIMES” parameter may be specified, and this is to be converted to a half-word binary value. The program is also to accept a “STOP” control statement, which has no operands.

In executable code, where a “READ” macro instruction would normally be issued, the programmer codes:

```
INT00 #SPZPARS ERROR=INT00,SOB=USRSOB,CMDID=X,
                MF=(E,INTL)
      (place code here for end-of-file condition)
```

The addresses of the routines to handle the “CALL” and “STOP” control statements are specified on the #SPZCMD macros, which are placed in an area of non-executable code in the program. The set of definition macros for this parsing operation is identified by the “ID=” parameter of the first #SPZCMD macro, and is referred to by the “CMDID=” parameter of the executable #SPZPARS macro. This permits multiple sets of command and operand definitions to be used for different purposes within the same program.

The non-executable macros needed for this example may be:

```
           #SPZCMD  CALL,RTN=CAL00,ID=X
           #SPZKWRD MODULE,FIELD=MODNAME,REQD=Y
           #SPZKWRD TIMES,FIELD=TIMESPEC,FMT=B
           #SPZCMD  STOP,RTN=STP00
           #SPZPEND

INTL       #SPZPARS MF=L
TIMESPEC   DS   H
MODNAME    DS   CL8
```

When the program is executed, and a control card:

```
           CALL      MODULE='AA',TIMES=16
```

is encountered, control is returned to routine “CAL00”, with the field MODNAME containing “AA” and 6 blanks, and the field TIMESPEC containing hexadecimal “0010” (binary 16). Note that the quotation marks specified on the control statement do not appear in the MODNAME field.

If a control statement:

```
           CALL      TIMES=1A,MOD=AA
```

were specified, an error would be recognized by SPZPARSE, and control would be returned to “INT00”, because this is the error address specified on the #SPZPARS macro. Messages indicating that the “TIMES” value is non-numeric, that “MOD” is an invalid keyword, and that the “MODNAME” parameter is omitted, would be issued to the SOB “USRSOB” following the output of the control statement itself.

The programmer may also specify that sub-parameters are permitted. For example, an operand of the form:

```
           LINES=(6,(5,5))
```

can be specified, so that one sub-parameter will contain the value “6”, and another the value “5,5”. Note that only one level of sub-parameter nesting is allowed; ie the value “5,5” cannot be divided further.

Positional (ie non-keyword) parameters may also be processed by SPZPARSE, and it is possible to define keyword parameters that do not require values (eg to use “./ CHANGE LIST” instead of “./ CHANGE LIST=YES”).

Full details of all SPZPARSE macro parameters will be found in the Span Macros manual.

5.3 SPZPARSE Validation and Format Conversion

The validation and conversion options for a particular command operand are controlled by the “FMT=”, “VALID=”, “REQD=”, “SET=”, “SETYN=”, “VALUE=” and “ALIAS=” parameters of the #SPZKWRD, #SPZPOSN and #SPZSUBP macro statements.

The “FMT=” parameter defines the conversion to be applied to the input data, and, by implication, certain validation criteria. The “FMT” codes are:

C	-	character format, no validity checking or conversion (this is the default option)
CN	-	character name data, where the first character of the value must be alphabetic (A-Z) or one of the characters #, @ or $ (the primary currency symbol)
N	-	character numeric values only, right justified, padded on the left with zeroes
P	-	numeric values only, convert to packed decimal
B	-	numeric values only, convert to binary
X	-	0-9, A-F only, convert to hexadecimal, left-justified but byte aligned (eg '01A' would result in a 2-byte field containing '01A0' hexadecimal).

Note that valid numeric values include leading and trailing “+” and “-” signs, but only one of these is accepted per input value.

The “VALID=” parameter defines a list of valid values that may be specified for the operand in question. This list of values is checked before any format conversion takes place.

The “REQD=” parameter indicates whether or not this operand is required for every occurrence of the particular command in the input stream.

The “SET=” parameter is used to convert literal parameter values into bit settings. It takes the form of a list of entries, each specifying a valid value, a System/390 “immediate” instruction operation code (OI, NI, XI, MVI) and a one-byte mask value. If the input operand corresponds to one of the values specified in the “SET=” list, the instruction constructed from the operation code and the mask value is executed upon the receiving field defined in the definition macro. For example, if a #SPZKWRD macro statement specifies:

```
SET=((FRED,MVI,X'00'),(BILL,OI,B'10000000'))
```

and an input value of “BILL” is found, the destination field will be modified by the following assembler instruction:

```
OI   field,B'10000000'
```

The "SETYN=" parameter is a special form of the "SET=" parameter. Here, the only permitted values for the operand are "Y", "YES", "N", "NO". A value of "Y" or "YES" causes an instruction of the form:

```
OI   field,bytevalue
```

to be executed, hence setting specified bits in the field ON; and a value of "N" or "NO" causes an instruction of the form:

```
NI   field,255-bytevalue
```

to be executed, hence setting specified bits in the field OFF.

eg

```
SETYN=B'10000000'
```

sets the bit 0 of the field to "1" if "Y" or "YES" is specified; and bit 0 of the field to "0" if "N" or "NO" is specified. The remaining bits of the field remain unmodified.

The "VALUE=" parameter specifies whether or not a "value" is required for a given keyword operand. If "VALUE=YES" is specified, the keyword in question must, if entered, be followed by an "=" sign and a value. If "VALUE=NO" is specified there must not be a value, and the keyword has effect merely by its presence or absence. "VALUE=OPT" specifies that there may or may not be a value entered, and the user program must test the receiving field to determine whether or not there was a value specified in each case.

The "ALIAS=" parameter allows a second spelling or an abbreviation of a keyword to be used interchangeably with the normal version of the keyword.

5.4 SPZPARSE - Values Returned to the Calling Program

When control is returned to the calling program following the execution of the #SPZPARS macro, all parameters associated with one command will have been examined. Error messages will have been issued as required and the following fields will have been set up:

In user code:

The destination fields for valid parameters values will contain the values specified, adjusted to the length of the field or to the length specified on the #SPZKWRD or #SPZPOSN macro. Format conversions will have been performed, in accordance with the "FMT=" operand of the #SPZKWRD or #SPZPOSN statement. Values input as null (eg "LINES=,") will be represented as null fields, ie containing blanks or zero according to the field format. No modification will be made to "SET" fields if they are input as null.

In the code generated by #SPZCMD statements:

The macro #SPZPMAP will generate DSECTs to be used for the analysis of information passed to, and returned from, SPZPARSE. A "USING" statement on the symbol "DPFXDS" will give addressability to the fields defined at the start of a definition macro set (built by the first #SPZCMD macro of a set). Field "DPFXOPI" will contain the two-byte offset from the start of the macro set to the command entry for the command found; a "USING" statement on symbol "DOPDS" will map the fields for this command entry. "DOPLL" is a one-byte field containing the length of the command minus one, field "DOPL" contains the command literal itself. It is not usually necessary to reference these fields.

In the code generated by #SPZKWRD and #SPZPOSN statements:

The one-byte field "DODUFLG" (in DSECT "DODDS") defined by the label field of a #SPZKWRD or #SPZPOSN statement (if any label was specified) is set to indicate the status of the parameter described by this macro:

X'00' parameter not found
X'04' parameter found, and moved to user field if a "keyword=value" type
X'08' parameter found, but was in error, and not moved to user field

The one-byte field "DODULEN" is set to the decremented length (length-1) of the data as it appears in user storage. If the format of the parameter was converted to binary or packed decimal, or was used to cause bit settings, this field contains the actual length of the user data area. If the format of the parameter was not converted, or was converted from hexadecimal representation, the length is the length of the input data excluding any padding characters added at the right hand end of the field.

In register 1:

Register 1 contains the address of the last physical input record read. This is significant only if the record was a "data" record.

5.5 SPZPARSE Advanced Functions

5.5.1 The "TYPE=MOD" Option

The macros #SPZKWRD, #SPZPOSN and #SPZSUBP permit the user program to modify dynamically the way SPZPARSE will process commands and their operands. The options generated by each one of these macros can be changed by the executable form of the macro, which is designated by specifying the "TYPE=MOD" operand of the macro, and coding the macro in the executable part of the program. Values and options that may be dynamically changed include: required/not-required ("REQD=" parameter), acceptable/not-acceptable ("ACCEPT=" parameter), field address ("FIELD=" parameter), "SETPRES" field address ("PRESFLD=" parameter), and data format ("FMT=" parameter). This facility is used to permit re-entrant programs to use SPZPARSE facilities, and also in cases where addresses and options are known only at execution time.

eg Required/Optional parameter control:

A parameter can be set as “required” by coding, for example:

```
#SPZKWRD  TYPE=MOD,NAME=fieldname,REQD=Y
```

and set “not-required” by coding:

```
#SPZKWRD  TYPE=MOD,NAME=fieldname,REQD=N
```

5.5.2 Print Control

If it is necessary to vary the SPZPARSE print options during the execution of the user program, this can be done by respecifying the values of the “FLAG=” and “PRINTD=” operands of the #SPZPARS macro on any occasion this macro is used.

5.5.3 Obtaining the same data in different formats

To obtain the value of a particular operand in more than one format in one pass, the following technique may be used. The parameter should be defined by a #SPZKWRD macro in the usual way. A single #SPZSUBP macro should then be specified, indicating the alternative data format.

eg A field is required in both character numeric and packed decimal format. Macros used are:

```
#SPZKWRD  KEYWD,FIELD=CHARVAL,FMT=N
#SPZSUBP  FIELD=PACKVAL,FMT=P
```

5.5.4 Using SPZPARSE to interpret a data string

SPZPARSE usually obtains its data from 80-byte input records. It is possible, however, to provide the data to be interpreted from the calling program in an in-storage buffer. The “STRING=” operand of the #SPZPARS macro is used to specify the address of this character string. The “STRLEN=” parameter specifies the length of this data (maximum is 255 bytes) if this cannot be obtained from the assembled length of the data field. “Register” notation may be used for either of these parameters of the #SPZPARS macro in its execute form, and the “STRING=” parameter must always be specified if an in-storage string is to be interpreted. The string data is interpreted in the usual way by SPZPARSE, but the string is not output to the user's SOB.

This technique is applicable to the interpretation of JCL EXEC statement parameter strings, strings derived from the interpretation of sub-parameters, replies to console messages, etc.

5.6 SPZPARSE Macro Parameters

Parameters of the SPZPARSE macros are described in full detail in the Span Macros manual. However, some general notes on these macros are presented here.

The five keyword definition macros, #SPZCMD, #SPZKWRD, #SPZPEND, #SPZPOSN, #SPZSUBP, and the "list" form of the #SPZPARS parameter macro, generate non-executable non-reentrant code.

The "execute" form of the #SPZPARS macro generates executable code and is equivalent to a READ macro when an input control statement dataset is being used. Note that the user program does not need to provide a DCB or any DCB information to SPZPARSE; SPZPARSE will perform all management of I/O control blocks, OPEN and CLOSE, provided that the user program continues to process input statements until the end-of-file condition is raised.

There can be more than one executable #SPZPARS macro in an assembly; and more than one set of #SPZCMD, #SPZKWRD, #SPZPEND, #SPZPOSN, #SPZSUBP macros, provided that each set of these macros is begun with a #SPZCMD macro with a unique single-character value for the "ID=" operand. The "CMDID=" operand of the #SPZPARS macro is then used to specify which set of command definition macros is being used for a particular invocation of the SPZPARSE service routine.

There must be one #SPZCMD macro for each valid command. The #SPZPOSN and #SPZKWRD macros that define valid positional and keyword operands for a command must immediately follow their corresponding #SPZCMD macro. Similarly, #SPZSUBP macros that define valid sub-parameters must follow the #SPZKWRD macro defining the operand which they sub-divide. All positional parameters must be defined before keyword parameters for each command.

A #SPZPEND macro, or a #SPZCMD macro with a new unique "ID=" value, must be coded following the last #SPZKWRD, #SPZSUBP or #SPZPOSN macro for the last command of a set of definition macros. No statement other than SPZPARSE macros should appear between the first #SPZCMD macro and its corresponding #SPZPEND macro.

The #SPZPMAP macro should be used in each user program. This macro provides all the SPZPARSE DSECTs, and these are used in some cases by the other SPZPARSE macros.

5.7 SPZPARSE Rules for Coding Input

5.7.1 "Standard" Format

a) If an "identifier" is required, this must appear in columns 1-2 of input records. Note that it is unusual for identifiers to be required.

b) Comments are indicated by an asterisk in column 1 (in column 3 if an identifier is required).

c) The "label" field, if present, must start in column 1 (in column 3 if an identifier is required), and must be followed by one or more blanks.

d) The command (if required) must appear next on the record (following any identifier and any label), in any column before column 70. The command must appear completely on one record and must be followed by at least one blank.

e) Positional parameters should precede keyword parameters, for ease of understanding. If a required positional parameter is expected, the first parameter encountered will be taken as this positional parameter, regardless of its format. Non-required positional parameters will be recognized anywhere in the input as long as their content does not equal the value of one of the keyword parameters permitted for the command - a valid keyword parameter will be taken as that parameter rather than as a non-required positional parameter.

f) Keyword parameters may be coded in any order. Each keyword must be completely contained in one record, but operand values may be split across records.

g) Each parameter except the last must be terminated by a comma, and the next keyword must be coded in the next field with no intervening blanks, or on the next record. The last parameter of a command must be followed by a blank. The columns up to column 72 remaining after the last parameter may be used for comments.

h) The value of a "keyword=value" parameter must be coded immediately following an "=" sign following its keyword. If the parameter contains commas or blanks, quotation marks or parentheses, the parameter must be enclosed in either quotation marks or parentheses. Any quotation marks within quotation marks must be coded as two adjacent quotation marks, and brackets within brackets must be paired. The parameter may be continued onto another record by coding the parameter up to and including column 71, and placing a non-blank character in column 72. The next parameter may continue in any column of the next record, but if the parameter is enclosed in quotation marks or brackets the first character must be the enclosing character.

i) If the parameter is divided into sub-parameters, the following rules apply:
   - if only the first sub-parameter is to be coded, it may be coded as if it was the main parameter.
   - sub-parameters must otherwise be enclosed in quotation marks or brackets. Sub-parameters of the same parameter must be separated by commas, and omitted sub-parameters must be indicated by a comma.

j) Data passed to SPZPARSE via the "STRING=" parameter of the #SPZPARS macro must conform to the above rules, but may not be continued onto another record. The maximum length of the string is 255 characters.

k) If a command has no parameters, any comments on the record must be preceded by a full stop.

5.7.2 "AMS" or "TSO" Format

a) "Identifiers" and "labels" are not supported with these formats of control statement.

b) Comments may be specified at any point in an input stream, and are indicated by a sequence "/* ... */". All data from the "Slash-asterisk" is ignored until an "Asterisk-slash" is found.

c) The command (if required) must appear first on the record, in any column before column 70. The command must appear completely on one record and must be followed by at least one blank.

d) Continuations of a command are indicated by coding a "-" or a "+" character as the last non-blank character of a record. If a "+" is coded, all non-blank data before the first significant character of the following record is ignored - this allows parameters values to be input that are longer than can be contained on a single input record.

e) Positional parameters should precede keyword parameters, for ease of understanding. If a required positional parameter is expected, the first parameter encountered will be taken as this positional parameter, regardless of its format. Non-required positional parameters will be recognized anywhere in the input as long as their content does not equal the value of one of the keyword parameters permitted for the command - a valid keyword parameter will be taken as that parameter rather than as a non-required positional parameter.

f) Keyword parameters may be coded in any order. Each keyword must be completely contained in one record, but operand values may be split across records.

g) Each parameter must be terminated by one or more blanks. The last parameter of a command is indicated by the lack of a continuation character.

h) The value of a "keyword(value)" parameter must be coded within parentheses following its keyword. Any quotation marks within quotation marks must be coded as two adjacent quotation marks, and brackets within brackets must be paired. The parameter may be continued onto another record by following the rules for continuations given above. The next "keyword(value)" parameter may directly follow the closing bracket of the previous parameter, with no intervening blank. The closing bracket may be omitted on the last "keyword(value)" parameter of a command.

i) If the parameter is divided into sub-parameters, the following rules apply:
   - if only the first sub-parameter is to be coded, it may be coded as if it was the main parameter.
   - sub-parameters must otherwise be enclosed in quotation marks or brackets. Sub-parameters of the same parameter must be separated by blanks or commas, and omitted sub-parameters must be indicated by a comma.

j) If the FORMAT=TSO option is used, any non-ambiguous abbreviation may be specified for any keyword parameter.

k) Data passed to SPZPARSE via the "STRING=" parameter of the #SPZPARS macro must conform to the above rules, but may not be continued onto another record. The maximum length of the string is 255 characters.

5.8 Invoking SPZPARSE

SPZPARSE is invoked by means of the #SPZPARS macro instruction, which may specify the entry point address of SPZPARSE if required.

SPZPARSE is re-entrant. The CSECT SPGLBL must be available to the SPZPARSE routine; it may either be link-edited with SPZPARSE (and SPOUTPUT, if required), or the address of the SPGLBL module may be passed to SPZPARSE via the "GLBLAD=" parameter of the #SPZPARS macro, or, if neither of these is done, SPGLBL must be available as a separate load module (named SPGLBL) at execution time. If SPGLBL is link-edited with SPZPARSE, the resulting load module is not re-entrant. The SPGLBL CSECT is obtained by assembling the #SPGLBL macro.

If the input to SPZPARSE is on a CA-LIBRARIAN Master (specified by the LIBR=LIBR option of the #SPZPARS macro), then the external subroutine SPZPLIB is used by SPZPARSE to access CA-LIBRARIAN. This must be installed as a separate load module, link-edited with the FAIR (File Access Interface Routine) as documented in the appropriate CA-LIBRARIAN manual. Note that SPZPLIB is not re-entrant and so cannot be placed in the Link Pack Area.

If the input to SPZPARSE is on a PANVALET library (specified by the LIBR=PANV option of the #SPZPARS macro), then the external subroutine SPZPPAN is used by SPZPARSE to access PANVALET. This must be installed as a separate load module, link-edited with PAM (Panvalet Access Method) as documented in the appropriate PANVALET manual. Note that SPZPPAN is not re-entrant and so cannot be placed in the Link Pack Area.

5.8.1 CALL

Link-edit SPZPARSE and SPZPARS2 with the invoking program, and do not specify the "PARSEP=" operand of the #SPZPARS macro used to invoke SPZPARSE. Alternatively, use the "PARSEP=" operand to specify the address of a V-constant for SPZPARSE in the user program. For user programs designed to be run under SPANEX, the "PARSEP=SPXMKWSC" and "GLBLAD=SPXMGLBA" parameters should be specified to make use of the common copies of SPZPARSE and SPGLBL.

5.8.2 External SPZPARSE

Create a load module of SPZPARSE and SPZPARS2 (or include them in another load module) and use the "PARSEP=" operand of the #SPZPARS macro to pass the address of the SPZPARSE module, using register notation.

5.9 SPZPARSE SCP Compatibility

SPZPARSE supports all the Operating Systems supported by other Span Software products. However, if SPZPARSE is to be transferred from one Operating System to another, the module must be re-assembled, using the appropriate macro libraries. SPZPARSE is sensitive to the system control program being used because it invokes the #SPXQ SPANEX macro (see the Span Macros manual).

5.10 SPZPARSE Error Messages

Listed below are the standard forms of the error messages that may be issued by SPZPARSE describing coding errors in input commands. The message identifiers may be altered by the user program by means of the "MSGPREF=" parameter of the #SPZPARS macro, and, if this is done, this message documentation should be reproduced with the amended message identifiers.

```
SPZ001I  parameter PARAMETER REQUIRED BUT NOT PRESENT
```
Explanation: The specified parameter is required but is not present.
Where produced: Destinations are as specified on the user SOB.

```
SPZ002I  INVALID COMMAND OR OPERATION
```
Explanation: The command specified is not one of those permitted. NOTE: This message may appear against continuation statements of a statement containing a format error.
Where produced: Destinations are as specified on the user SOB.

```
SPZ003I  EXPECTED CONTINUATION NOT RECEIVED DUE TO END OF FILE
```
Explanation: End-of-file has been reached on a control statement dataset, but the previous control statement is to be continued.
Where produced: Destinations are as specified on the user SOB.

```
SPZ004I  COMMAND SHOULD HAVE NO PARAMETERS
```
Explanation: A parameter is specified for a command that does not have any valid operands.
Where produced: Destinations are as specified on the user SOB.

```
SPZ005I  INVALID CONTINUATION
```
Explanation: The parameter value extends past column 71 and no continuation mode is present.
Where produced: Destinations are as specified on the user SOB.

```
SPZ006I  value VALUE HAS NO TERMINATING APOSTROPHE OR UNPAIRED
           PARENTHESES
```
Explanation: A character string that was started by a quotation mark or bracket has no terminating quotation mark or bracket.
Where produced: Destinations are as specified on the user SOB.

```
SPZ007I  value LITERAL NOT TERMINATED WITH A BLANK OR COMMA
```
Explanation: The character following an ending quotation mark or bracket is neither a blank nor a comma.
Where produced: Destinations are as specified on the user SOB.

```
SPZ008I  value KEYWORD IS NOT PERMITTED FOR THIS COMMAND
```
Explanation: The specified keyword parameter is not a valid operand of this command.
Where produced: Destinations are as specified on the user SOB.

```
SPZ009I  parameter CONTAINS A NON-NUMERIC CHARACTER
```
Explanation: The specified parameter is required to be numeric but contains a non-numeric character.
Where produced: Destinations are as specified on the user SOB.

```
SPZ010I  value VALUE EXCEEDS MAXIMUM PERMITTED LENGTH
```
Explanation: The specified parameter value is too long.
Where produced: Destinations are as specified on the user SOB.

```
SPZ011I  parameter CONTAINS A CHARACTER THAT DOES NOT REPRESENT
           A HEXADECIMAL BIT PATTERN
```
Explanation: The specified parameter is required to be input in hexadecimal format, but contains a character that has no hexadecimal equivalent.
Where produced: Destinations are as specified on the user SOB.

```
SPZ012I  keyword VALUE IS NOT ONE OF THE PERMITTED VALUES
```
Explanation: The value of the specified parameter is not one of those permitted for this keyword.
Where produced: Destinations are as specified on the user SOB.

```
SPZ013I  LABEL FIELD EXCEEDS MAXIMUM LENGTH
```
Explanation: The label field on an input statement is too long. The maximum length of a label is 8 bytes.
Where produced: Destinations are as specified on the user SOB.

```
SPZ014I  parameter SUB-PARAMETER HAS NO TERMINATING APOSTROPHE
           OR UNPAIRED PARENTHESES
```
Explanation: A sub-parameter of the parameter specified is itself started with a quotation mark or bracket, but is not terminated with a quotation mark or bracket.
Where produced: Destinations are as specified on the user SOB.

```
SPZ015I  parameter SUB-PARAMETER NOT TERMINATED WITH A BLANK OR
           COMMA
```
Explanation: A sub-parameter of the parameter specified is enclosed in quotation marks or brackets, but the character following the ending quotation mark or bracket is neither a blank nor a comma.
Where produced: Destinations are as specified on the user SOB.

```
SPZ016I  parameter EXCESSIVE SUB-PARAMETERS
```
Explanation: Too many sub-parameters were input for the parameter specified.
Where produced: Destinations are as specified on the user SOB.

SPZ017I parameter CONTAINS DUPLICATE SIGN CODE
Explanation: The numeric parameter specified has more than one sign code in the input.
Where produced: Destinations are as specified on the user SOB.

SPZ018I value KEYWORD IS NOT PERMITTED
Explanation: The specified keyword is not defined for use in the command being processed.
Where produced: Destinations are as specified on the user SOB.

SPZ019I NO COMMAND OR OPERATION FOUND ON THIS STATEMENT
Explanation: This statement or command string could not be interpreted because no command field was found.
Where produced: Destinations are as specified on the user SOB.

SPZ020I value KEYWORD, PARAMETER VALUE NOT PERMITTED
Explanation: The specified keyword was entered with a value (following an "=" sign) but no value was expected for this keyword.
Where produced: Destinations are as specified on the user SOB.

SPZ021I PANVALET DD STATEMENT MISSING
Explanation: The DD statement specified by the INDD= parameter of the #SPZPARS macro was not present, and the CA-PANVALET input option was being used. The specified DD name must be allocated to the CA-PANVALET library. The end-of-file condition is set.
Where produced: Destinations are as specified on the user SOB.

SPZ022I PANVALET INPUT MEMBER NOT FOUND
Explanation: The CA-PANVALET input option was being used, but the member name specified by the MEMBER= parameter of the #SPZPARS macro did not exist in the CA-PANVALET library. The end-of-file condition is set.
Where produced: Destinations are as specified on the user SOB.

SPZ023I PANVALET OPEN ERROR
Explanation: A non-recoverable OPEN error occurred in CA-PANVALET. The end-of-file condition is set.
Where produced: Destinations are as specified on the user SOB.

SPZ024I PANVALET READ ERROR
Explanation: A non-recoverable READ error occurred in CA-PANVALET. A CA-PANVALET CLOSE is issued and the end-of-file condition is set.
Where produced: Destinations are as specified on the user SOB.

SPZ025I LIBRARIAN DD STATEMENT MISSING
Explanation: The DD statement specified by the INDD= parameter of the #SPZPARS macro was not present, and the CA-LIBRARIAN input option was being used. The specified DD name must be allocated to the CA-LIBRARIAN Master. The end-of-file condition is set.
Where produced: Destinations are as specified on the user SOB.

SPZ026I LIBRARIAN INPUT MEMBER NOT FOUND
Explanation: The CA-LIBRARIAN input option was being used, but the member name specified by the MEMBER= parameter of the #SPZPARS macro did not exist in the CA-LIBRARIAN Master. The end-of-file condition is set.
Where produced: Destinations are as specified on the user SOB.

SPZ027I LIBRARIAN OPEN ERROR
Explanation: A non-recoverable OPEN error occurred in CA-LIBRARIAN. The end-of-file condition is set.
Where produced: Destinations are as specified on the user SOB.

SPZ028I LIBRARIAN READ ERROR
Explanation: A non-recoverable READ error occurred in CA-LIBRARIAN. A CA-LIBRARIAN CLOSE is issued and the end-of-file condition is set.
Where produced: Destinations are as specified on the user SOB.

SPZ029I PDS INPUT MEMBER NOT FOUND
Explanation: Partitioned dataset input was being used, but the member name specified by the MEMBER= parameter of the #SPZPARS macro did not exist in the PDS. The end-of-file condition is set.
Where produced: Destinations are as specified on the user SOB.

SPZ030I INPUT DD STATEMENT MISSING
Explanation: The required DD statement was not supplied for sequential or Partitioned Dataset input.
Where produced: Destinations are as specified on the user SOB.

SPZ031I parameter KEYWORD IS AMBIGUOUS
Explanation: The keyword parameter specified was an abbreviation of a valid keyword for this command, but the same abbreviation could apply to more than one possible valid keyword. Modify the command to use a more specific form of the desired keyword.
Where produced: Destinations are as specified on the user SOB.

SPZ032I value KEYWORD, PARAMETER VALUE IS REQUIRED
Explanation: The specified keyword was entered with no value (following an "=" sign or within parentheses). This keyword requires a value.
Where produced: Destinations are as specified on the user SOB.


```
SPZ033I  value DOES NOT BEGIN WITH A VALID ALPHABETIC
           CHARACTER: A-Z, #, @, $
```

Explanation: The specified keyword or positional parameter specifies an object name, and must begin with an alphabetic character or one of the characters #, @ or $ (the primary currency symbol).
Where produced: Destinations are as specified on the user SOB.

6 SPZQPAM Service Routine

6.1 SPZQPAM Introduction

SPZQPAM is the Span Software Queued PDS Access Method routine.

The following discussion of SPZQPAM assumes knowledge of assembler language programming and of the assembler macros that are used to perform Direct Access device input/output.

There are two well-known methods of reading multiple members of a single partitioned dataset during one execution of a program:

- use BPAM macros OPEN, FIND, READ, READ, . . . FIND, READ, READ, etc, CLOSE. This has the disadvantage that all synchronization and deblocking must be handled by the user program;
- use RDJFCB, then modify the JFCB with the member name required, use "OPEN TYPE=J", GET, GET, . . . CLOSE, and go back to modify the JFCB again for each member required. This requires the high over head of an OPEN and a CLOSE for each input member.

SPZQPAM provides a PDS access method that combines the advantages of both the above methods. The dataset is opened and closed once only, the #SPZFIND macro is used in exactly the same was as FIND, but instead of a READ and CHECK macro being used to read a physical record, a GET macro is used to access a logical record.

SPZQPAM will optionally provide automatic buffering services, and take advantage of operating system chained scheduling facilities, in order to give excellent read performance, particularly for large input members.

SPZQPAM supports any number of input libraries concatenated to the DD statement specified for its use.

6.2 SPZQPAM Method of Use

6.2.1 DCB Required

The calling program must set up a DCB specifying "DSORG=PO,MACRF=R". A valid DDNAME must be supplied before SPZQPAM is used. If SPZQPAM automatic buffering is required, the DCB should include the NCP parameter to specify the number of buffers to be used; NCP=10 is normally an adequate specification even for very large input PDS members.

6.2.2 OPENing the PDS

Instead of using a conventional OPEN macro, entry point SPZOPEN of SPZQPAM should be called with register 1 containing the address of the DCB concerned. If OPEN TYPE=J is required, then the high-order bit of register 1 should be set to one. The #SPZOPEN macro may be used to perform the PDS open (see the Span Macros manual for details). SPZOPEN is externally compatible with the OPEN macro, ie DCB field DCBOFLGS, bit X'10', will be set to one if OPENing was successful.

6.2.3 Positioning to a Member

Instead of using the conventional FIND macro, the #SPZFIND macro must be used, with exactly the same parameter format as for FIND, and with either option “C” or option “D”.

6.2.4 Reading data

Logical records are read sequentially by repeated use of the GET macro. The only form of GET supported is LOCATE mode, ie the only operand required is the DCB address. On return from GET, register 1 will contain the address of the next logical record. Reading of the current member may be terminated at any time by issuing a FIND macro, otherwise the calling program's EODAD routine will be entered when end-of-member is encountered.

6.2.5 Closing the PDS

Instead of the normal CLOSE macro, entry point SPZCLOSE of SPZQPAM should be called with register 1 containing the address of the DCB concerned. Alternatively, the #SPZCLOS macro may be used (see the Span Macros manual for details).

6.3 Including SPZQPAM in your program

As distributed, SPZQPAM consists of a single CSECT, named SPZQPAM, with three secondary entry points, SPZOPEN, SPZFIND and SPZCLOSE. These are two methods for including the object module of SPZQPAM in programs that require it:

- explicitly, with a linkage editor INCLUDE statement, naming SPZQPAM;
- implicitly, by giving the object module alias names of SPZOPEN and SPZCLOSE, and allowing the linkage editor to perform “automatic library call” from the object library.

SPZQPAM is re-entrant, but may be invoked by means of a LINK macro (SVC 6) only if it resides in the Link Pack Area.

6.4 Using SPZQPAM with SPZDIRD

Many data management programs are required to read the entire directory of a PDS and then to process some or all of the members in alphabetic sequence. Span Software Service Routines SPZDIRD and SPZQPAM can very easily be used to perform these functions.

For example, a program to print all members of a source PDS might look as follows:

```
          ........  )   <<<<<<<<<<program initialization code>
          LA    R1,DIRDPARM
          CALL  SPZDIRD
          LTR   R15,R15
          BNZ   DIRDERR           Errors in Directory
          #SPZOPEN SYSLIB         Open for SPZQPAM
          L     R2,DIRDDIRY       A(SPZDIRD Output)
          L     R3,0(,R2)         # members in directory
          LA    R4,8(,R2)         A(first member entry)
DIRLOOP   DS    0H
          ........  )   <<<<<<<<<<print member heading>
          MVC   TTR(4),8(R4)      TTRK of member from dir
          #SPZFIND  SYSLIB,TTR,C  Point to start of member
RECLOOP   GET   SYSLIB            Read one logical record
          MVC   OUTBUF(80),0(R1)  Move record
          PUT   SYSPRINT,CONCHAR  Print it
          B     RECLOOP
EOFLIB    LA    R4,14(,R4)        A(next member entry)
          BCT   R3,DIRLOOP        Do next member
          L     R0,4(,R2)         Length to be freed
          LR    R1,R2             A(directory area)
          FREEMAIN R,A=(1),LV=(0) Free it
          #SPZCLOS SYSLIB         Close the DCB
          ........  )   <<<<<<<<<<program termination code>
DIRDPARM  DC    A(SYSLIB+40)      A(DDNAME)
          DC    A(F0)             A(# user bytes)
DIRDDIRY  DC    F'-1'             Result, flags follow
          DC    X'50000000'       Concat, BLDL format
F0        DC    F'0'              # user bytes
SYSLIB    DCB   DSORG=PO,MACRF=R,DDNAME=SYSLIB,RECFM=FB,      X
                BLKSIZE=800,LRECL=80,EODAD=EOFLIB,NCP=10
TTR       DC    F'0'
SYSPRINT  DCB   DSORG=PS,MACRF=PM,DDNAME=SYSPRINT,RECFM=FBA,  X
                LRECL=133, . . .
CONCHAR   DC    C' '              Print control
OUTBUF    DC    CL132' '          Output record area
          END
```

6.5 SPZQPAM Implementation Restrictions

- GET MOVE mode is not supported.
- Undefined record format is not supported.
- CLOSE TYPE=T is not supported.
- FIND macro is not supported for OS/390 Version 2.4.0 or later, #SPZFIND must be used

- Look-ahead buffering is performed for the current member, with I/O being issued when the half the specified buffers (NCP DCB parameter) are empty.

7 SPZSORT Service Routine

7.1 SPZSORT Introduction

SPZSORT is an efficient in-storage sort routine, handling fixed-length records with a maximum size of 32767 bytes. Records must be input or generated in a single contiguous area of virtual storage. The sort key can be any single field within the record and sorting can be in ascending or descending sequence. SPZSORT is re-entrant, and may be called in 31-bit mode (MVS/XA and MVS/ESA systems) in order to sort a table that is located above the 16-megabyte line. The sorting algorithm used is a shell sort.

7.2 SPZSORT Input

The following 5-word parameter must be passed to SPZSORT in register 1:

Word 1	-	address of the area containing the records to be sorted.
Word 2	-	address of a fullword containing the length of each record.
Word 3	-	address of a fullword containing the number of records.
Word 4	-	address of a fullword containing the key offset within each record.
Word 5	-	address of a fullword containing the length of the key as a positive number for ascending sort, and as a negative number for descending sort.

SPZSORT may be invoked either by CALL or by LINK, and may reside above the 16-Megabyte line in MVS/XA and MVS/ESA systems.

7.3 SPZSORT Return Codes

SPZSORT will return one of the following codes in Register 15:

0	Sort successful
16	One of the following errors was detected in the parameters passed to SPZSORT: • Invalid record length specified (greater than 32767) • Key length zero • Key length + Key offset greater than record length

7.4 SPZSORT Output

The only output from SPZSORT is the input area, now containing the records in the requested sequence.

This manual is published by

Span Software Consultants Limited

Little Moss, Peacock Lane

High Legh

Knutsford

Cheshire

WA16 6PL

England

Tel: +44/0 1565 832999

Fax: +44/0 1565 830653

Email: spanex@spansoftware.com

www.spansoftware.com

to whom all comments and suggestions should be sent.