3890/XP Series and 3890/XP Enhanced

SC31-4070-00

SPXServ Reference

3890/XP Series and 3890/XP Enhanced

SC31-4070-00

SPXServ Reference

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page ix.

Fourth Edition (January 1998)

This edition is a major revision of, and obsoletes, GC31-2704. Information in this manual is subject to change from time to time. Because this edition includes so much new or changed material, it contains no revision bars and should be read in its entirety. Before using this publication in connection with the operation of IBM systems, consult the *IBM System/370, 30xx, 4300, and 9370 Processors: Bibliography of Industry Systems and Application Programs*, GC20-0370, for the editions that are applicable and current.

IBM does not stock publications at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Department 78L, MG83/204, 8501 IBM Drive, Charlotte, NC, 28267, U.S.A. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Contents

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights or other legally protectable rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

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, MG39/201, 8501 IBM Drive, Charlotte, NC 28262-8563, U.S.A. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY, 10594 U.S.A.

Trademarks

The following terms, denoted by an asterisk (*) elsewhere in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

IBM	Operating System/2	OS/2
System/370	Personal System/2	PS/2
MVS	VSE	

The following terms, denoted by a double asterisk (**) elsewhere in this publication, are trademarks of other companies as follows:

Intel	Intel Corporation
Recognition UT 600/1000 XP	Recognition International Inc.

About This Manual

This manual describes the sort-program-execution-services dynamic link library (SPXSERV.DLL) function requests.

The sort-program-execution-services (SPXServ) functions support all models of the IBM* 3890/XP Series Document Processor:

- IBM 3890/XP Document Processor
- IBM 3891/XP Document Processor
- IBM 3892/XP Document Processor.

In addition, this manual describes the SPX32 functions that are available with the IBM 3890/XP Enhanced Document Processor. Additional functions available in the 3890/XP Enhanced include:

- Faster execution of sort programs.
- High resolution timing functions.
- Easier memory manipulation via flat 32-bit addresses vs. segmented addresses in older processors.
- Documented interface to the 3890/XP Enhanced user interface.
- Enhanced Graphical User Interface (GUI) allowing selection of function via menu.
- Easier to use help system.
- Error recovery from traps in user code.

Who Should Read This Manual

This manual is for customer programmers who write user Sort programs in OS/2*-supported languages. It assumes that you are familiar with the *IBM 3890/XP Document Processor General Information Manual*, GA34-2012, and the *IBM 3890/XP Enhanced Document Processor Programming Guide*, SC31-4069.

Terms That You Should Know

Multiple Virtual Storage (MVS*) Support means the 3890/XP MVS Support licensed program.

Virtual Storage Extended (VSE*) Support means the 3890/XP VSE Support licensed program.

Document processor means all models of the 3890/XP Series document processor.

Routing number means the routing-and-transit field on MICR documents.

Code line data matching was formerly called image matching or image processing.

OS/2 language means a Sort program written in a language that works with the IBM Personal System/2* (PS/2*) under Operating System/2* (OS/2).

---

* Trademark of IBM

Control Program means the 3890/XP and 3890/XP Enhanced microcode and related modules.

How This Manual Is Organized

This manual consists of three chapters:

- Chapter 1, "SPXServ," describes the SPXSERV.DLL library and the SPXServ function calling conventions.
- Chapter 2, "Alphabetic Reference," describes the SPX functions.
- Chapter 3, "Alphabetic Reference (Spx32 Functions)," describes the SPX32 functions.

This manual also contains a glossary and an index.

Related Publications

For other information that is related to the 3890/XP Series and 3890/XP Enhanced Document Processors, see the following publications:

- *IBM 3890/XP Document Processor General Information Manual*, GA34-2012
- *IBM 3890/XP Enhanced Document Processor Programming Guide*, SC31-4069
- *IBM 3890/XP Series Stacker Control Interface Reference*, SC31-2703
- *IBM 3890/XP High-Level Language Services Program Reference Manual*, SC31-2653 (World Trade only)
- *IBM 3890/XP Series Document Processor Operator's Guide*, GA34-2013
- *IBM 3890/XP Enhanced Document Processor Operator's Guide*, GA34-2212
- *IBM 3891/XP and 3892/XP Document Processors Operator's Guide*, GA34-2050
- *IBM 3890 Document Processor Machine and Programming Description*, GA24-3612.

Chapter 1. SPXServ

SPXServ is a dynamic-link library (DLL) supplied by IBM and installed as part of the 3890/XP Series Control Program or 3890/XP Enhanced Series Control Program. It contains functions that give you access to document processor data and facilities. The illustration below shows how the SPXServ functions serve as the gateway by which OS/2 language Sort program modules can read or write to the data areas in program storage and in auxiliary storage.

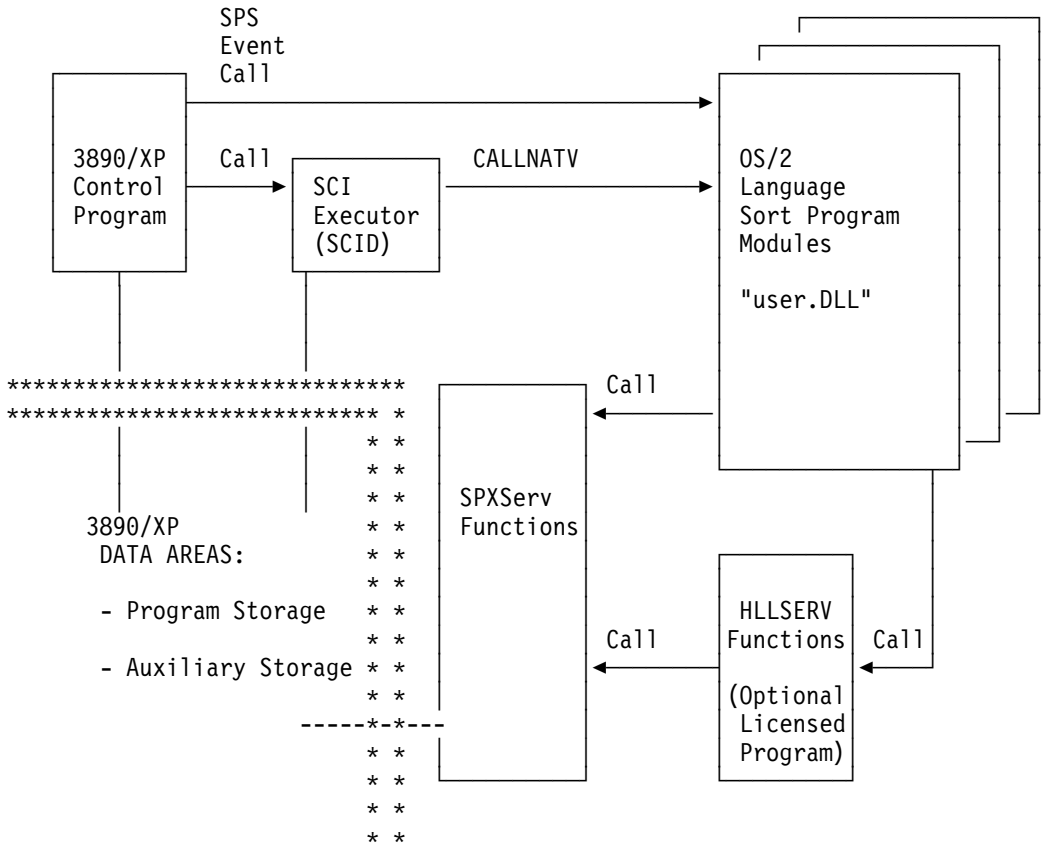

Figure 1-1. SPXSERV.DLL - The Gateway to Document Processor Data Areas

Types of Functions

The SPXServ functions can be divided into two basic types, GET and PUT, which copy data to or from data areas (such as program storage and auxiliary storage) of the 3890/XP Control Program:

GET The GET type of function copies or returns data from auxiliary storage or program storage to your data area.

PUT The PUT type of function copies or stores data from your data area to auxiliary storage or program storage.

Categories of Functions

Each function of SPXServ is classified into one of the following categories:

- Immediate action
- Fixed-length transfer
- Fixed-length element transfer
- Variable-length transfer.

Immediate action

Functions in this category require no parameters. To call functions in this category, you specify only the name of the function.

Fixed-length transfer

Functions in this category require one parameter. To call functions in this category, you must specify the name of the function and an address to indicate the location in your data area (either the source of the data in PUT functions or the destination of the data in GET functions).

Fixed-length element transfer

Functions in this category require two parameters. This category copies a single element (record) to or from a 3890/XP data area. To call functions in this category, you must specify the name of the function and an address to indicate the location in your data area (either the source of the data in PUT functions or the destination of the data in GET functions). You must also specify a word that indicates which element (a record) to transfer.

Variable-length transfer

Functions in this category require three parameters. The variable-length-transfer category is similar to the fixed-length element transfer category, but it specifies (either directly or indirectly) the length of the transfer. To call functions in this category, you must specify the number of bytes to transfer and all the parameters that the fixed-length element transfer category requires.

SPXServ Call-Return Interface

You use the CALL function to request SPXServ support. This function *calls* a specific SPXServ function entry point within the SPXSERV.DLL library. Because the functions in this library are dynamically-linked subroutines, you must code them as:

```
EXTRN entrypoint:FAR.
```

This instructs the compiler to create a standard external far reference.

To reference this function, in the OS/2 .DEF file, the following needs to be included in the IMPORTS section:

```
SPXSERV.entrypoint
```

The function names are printed in this manual in mixed case for readability. They are known to the system as uppercase character strings. If you are using a compiler that generates mixed case external names, you must code the function calls in uppercase.

Data Passing

All SPXServ functions must be called using a PASCAL protocol.

You can describe the data item directly, by giving its value (BYTE, WORD, or DWORD), or indirectly, by giving its reference (address). The following pseudocode indicates whether the description that you push onto the stack is a value or a reference:

PUSH This pushes data items of various sizes onto the stack (pass by value).

PUSH@ For SPXServ this pushes the address of a data item onto the stack. For SPXServ all addresses consist of a 32-bit value, a 16-bit selector, and a 16-bit offset (pass by reference).

The pseudocode contains the following types of data items:

BYTE This is 8 bits, pushed onto the stack by value or by reference.

WORD This is 2 bytes, pushed onto the stack by value or by reference.

DWORD This is 4 bytes, pushed onto the stack by value or by reference.

ASCIIZ This is a character string that ends in a null character (X'00') and that is pushed onto the stack by reference.

OTHER This is a structure that is pushed onto the stack by reference.

Warning: If you specify an incorrect address, field, length, or pointer, a *protection violation* might occur. The operating system detects the protection violation; the Sorter Control Program does not detect this violation and cannot assist in the recovery from such an error.

Warning: If you define a data area that is smaller than the character string that a function returns, you will overlay data in your program. This could result in a *protection violation* and the loss of data or other unpredictable results.

Return Codes

All functions return a code in the accumulator (AX) register. A return code of zero indicates that the function processed without error. A return code of 100 indicates that this function does not work with this type of document processor. You would get a return code of 100, for example, if your Sort program were running on the 3890/XP and called a function that is meaningful only to the 3891/XP.

SPX32 Call-Return Interface

You use the CALL function to request SPXserv support. This function *calls* a specific SPX32 function entry point within the SPXSERV.DLL library. Because the functions in this library are dynamically-linked subroutines, you must code them as:

Note: These descriptions use the IBM CSet/2 or CSet++ System calling conventions.

```
unsigned short _System entrypoint(parms...);
```

This instructs the compiler to create a standard external reference.

To reference this function, in the OS/2 .DEF file, the following needs to be included in the IMPORTS section:

```
SPXSERV.entrypoint
```

The function names are printed in this manual in mixed case for readability. They are known to the system as uppercase character strings. If you are using a compiler that generates mixed case external names, you must code the function calls in uppercase.

Data Passing

All SPXS32 functions must be called using a OS/2 System protocol. By prototyping the function correctly in your code, the compiler will check that you are calling the function with the correct parameters.

The parameters that can be passed to SPX32 functions are:

unsigned char
: This causes the value of the byte referenced to be pushed on the stack.

unsigned short
: This causes the value of the 16-bit word referenced to be pushed on the stack.

unsigned long
: This causes the value of the 32-bit word referenced to be pushed on the stack.

void *
: This causes a 32-bit address to be pushed on the stack. A void * does not do any type checking.

unsigned char *
: This causes a 32-bit address of a byte variable to be pushed on the stack.

unsigned short *
: This causes a 32-bit address of a 16-bit variable to be pushed on the stack.

unsigned long *
: This causes a 32-bit address of a 32-bit variable to be pushed on the stack.

Warning: If you specify an incorrect address, field, length, or pointer, a *protection violation* might occur. The 3890/XP Enhanced Sorter Control Program detects this violation and will record the information to assist in the recovery from such an error.

Warning: If you define a data area that is smaller than the character string that a function returns, you will overlay data in your program. This could result in a *protection violation* and the loss of data or other unpredictable results.

Return Codes

All functions return a code in the accumulator (AX) register. A return code of zero indicates that the function processed without error. A return code of 100 indicates that this function does not work with this type of document processor. You would get a return code of 100, for example, if your Sort program were running on the 3890/XP and called a function that is meaningful only to the 3891/XP.

HLLServ Calls

The High-Level-Language Services licensed program (HLLServ) supplies additional functions that are tailored to the 3890/XP Series machines.

For more information, see the *IBM 3890/XP High-Level Language Services Program Reference Manual*.

This manual is *recommended reading* for the concepts involved in writing XP user Sort programs in OS/2 supported languages as dynamic link library modules (DLLs).

Chapter 2. Alphabetic Reference

This chapter provides an alphabetic reference of all the SPXServ functions. For each function, it describes:

- Purpose
- Calling sequence
- Return codes.

SpxAbSrtProg

Purpose: The SpxAbSrtProg function prematurely ends the Sort Program Sequence (SPS). No further modules for the current SPS event are called and additional SPS events for the current document are ignored.

The process buffer reflects the level of processing that was completed when the abort flag was set. Byte 7, bit 1 in the header is set on to indicate the Cancel operation; the document is sent to module-pocket 1/1. The feature control is handled as though the document had been autoselected.

Calling Sequence:

```
EXTRN  SpxAbSrtProg:FAR

CALL   SpxAbSrtProg
```

Return Codes:

AX	*Description*
0	Successful.

SpxDSG

Purpose: The SpxDSG function requests the disengage function.

This function does the same thing as the Stacker Control Instruction (SCI) language disengage (DSG) macro. It sets a flag that is acted on during post-sort processing of the current document. Document feeding from the main hopper stops immediately. Merge documents feed until all merge requests are acted on. All documents in the document path are processed normally before the transport stops.

Note: There can be up to 12 documents in the document path.

Calling Sequence:

```
EXTRN  SpxDSG:FAR

CALL   SpxDSG
```

Return Codes:

AX	*Description*
0	Successful.

SpxDSGA

Purpose: The SpxDSGA function requests the disengage-and-set-operator-attention function.

This function does the same thing as the SCI language disengage-and-set-operator-attention (DSGA) macro. SpxDSGA sets a flag that is acted on during post-processing of the current document. Document feeding from the main hopper stops immediately. Merge documents feed until all merge requests are acted on. All documents in the document path are processed normally before the transport stops.

Note: There can be up to 12 documents in the document path.

Calling Sequence:

```
EXTRN  SpxDSGA:FAR

CALL   SpxDSGA
```

Return Codes:

AX	*Description*
0	Successful.

SpxEIMAT

Purpose: The SpxEIMAT function writes a 16-bit data word from the calling program into the control field for extended code line data match. This overrides the definition given in the initialization data.

Calling Sequence:

```
EXTRN  SpxEIMAT:FAR

PUSH@  WORD    buffer            ;Data buffer
CALL   SpxEIMAT
```

buffer
This is a data area that specifies the fields to be matched.

Return Codes:

AX	*Description*
0	Successful. All requested fields are valid.
1	A specified field was not previously defined for code line data match by the code line definition table or the initialization data. It will now be used.

Bit definitions:

0	Field 1	**8**	Field 9
1	Field 2	**9**	Field 10
2	Field 3	**10**	Field 11
3	Field 4	**11**	Field 12
4	Field 5	**12**	Field 13
5	Field 6	**13**	Field 14
6	Field 7	**14**	Field 15
7	Field 8	**15**	Reserved

Note: To use extended code line data matching, first call SpxEIMAT; then call SCII (supplied by IBM in SCIEM) from your program. You must make this call during the SPSDOC event.

To call SCII, code the following in the source file:

```
EXTRN  SCII:FAR

CALL   SCII
```

then code the following in the link edit:

```
IMPORTS    SCIEM.SCII
```

SpxFixM

Purpose: The SpxFixM function is called at SPSDOC-time to specify fixed message endorsement(s) for the current document. See SpxLoadFixM.
This function is supported on the 3891/XP, 3892/XP and Universal Transport (UT)/XP**

Calling Sequence:

```
EXTRN SpxFixM:FAR

PUSH@ OTHER       buffer      ;Message numbers
CALL  SpxFixM
```

buffer

A 6-byte data area that specifies six fixed message numbers, front side lines 1, 2, and 3 and back side lines 1, 2, and 3. A zero specifies no message for that side/line.

If the font size is defined as large, only one line can be printed on that side. It must be specified as line 1.

Return Code:

AX	*Description*
0	Successful
1	Invalid message number(s)
100	This function is not supported on this document processor.

Notes:

1. The same fixed message can be given for more than one side/line.
2. If you direct a fixed message to the same side/line that has a variable text message (via CONVERGE), the error will cause the message
   6030 Transport Option State Error.
3. If you specify a large font size and put a message to other than line 1 for that side, the error will cause the message 6030 Transport Option State Error.
4. If you specify a fixed message that was not loaded, the return code reflects the error. The machine does not stop.

** The UT/XP is trademarked by Recognition International Inc as *Recognition UT 600/1000 XP*.

SpxFM

Purpose: The SpxFM function requests a document from the merge feeder. This function does the same thing as the SCI feed merge (FM) macro.

Calling Sequence:

```
EXTRN  SpxFM:FAR

CALL   SpxFM
```

Return Codes:

AX	*Description*
0	Successful
1	Request ignored; can only be requested once during document-time.

SpxGet92Features

Purpose: The SpxGet92Features function returns the features installed on the 3891/XP and 3892/XP.

Calling Sequence:

```
EXTRN  SpxGetFeatures:FAR

PUSH@  WORD    feature          ;Features installed
CALL   SpxGetFeatures
```

feature

This is a 2-byte data area in which this function returns the features installed status. The following names apply:

Bit	*Feature*
0	MICR-1 Reader
1	Hi-Line Reader/MICR2
2	OCR-1 Reader (OCR RPQ)
3	OCR-2 Reader (OCR RPQ)
4	IBM OCR-3 Reader
5	Front Ink Jet Printer
6	Back Ink Jet Printer
7	Power Encoder
8	Image Scanner
9	Microfilmer
10	Roll On Endorser
11	Merge Hopper
12	Reserved
13	Reserved
14	Reserved
15	Reserved.

Return Codes:

AX	*Description*
0	Successful.
100	This Spx is not available on a 3890/XP

Note: This Spx returns the 3891/XP or 3892/XP feature installed list and is not applicable on the 3890/XP. For a complete feature installed list on the 3890/XP, see “SpxGetFeatures” on page 2-21.

SpxGet92InitList

Purpose: The SpxGet92InitList function returns the list of initialized features for the 3891/XP and 3892/XP.

Calling Sequence:

```
EXTRN  SpxGetInitList:FAR

PUSH@  OTHER   featlist          ;Feature list
CALL   SpxGet92InitList
```

featlist

This is a 2-byte data area in which this function returns the initialized feature data. The following values apply:

Bit	*Feature*
0	MICR-1 Reader
1	Hi-Line Reader/MICR2
2	OCR-1 Reader (OCR RPQ)
3	OCR-2 Reader (OCR RPQ)
4	IBM OCR-3 Reader
5	Front Ink Jet Printer
6	Back Ink Jet Printer
7	Power Encoder
8	Image Scanner
9	Microfilmer
10	Roll On Endorser
11	Merge Hopper
12	Reserved
13	Reserved
14	Reserved
15	Reserved.

Return Codes:

AX	*Description*
0	Successful.
100	This Spx is not available on a 3890/XP

Note: This Spx returns the 3891/XP or 3892/XP feature list and is not applicable on the 3890/XP. For a complete feature list on the 3890/XP see, “SpxGetInitList” on page 2-26.

SpxGetAbortSrt

Purpose: The SpxGetAbortSrt function returns the status of the abort-sort-program flag. See SpxAbSrtProg.

Calling Sequence:

```
EXTRN  SpxGetAbortSrt:FAR

PUSH@  BYTE        flag     ;Returned flag
CALL   SpxGetAbortSrt
```

flag

This is a 1-byte data area to receive the abort-sort-program flag. The following values apply:

hex 00 The flag is not on.
hex 01 The flag is on.

Return Codes:

AX	*Description*
0	Successful.

SpxGetActEndDate

Purpose: The SpxGetActEndDate function returns the active endorse date.

Calling Sequence:

```
EXTRN  SpxGetActEndDate:FAR

PUSH@  OTHER   date        ;Date returned  8 bytes
CALL   SpxGetActEndDate
```

date

This is an 8-byte data area in which this function returns the active endorse date. Format is ASCII.

Return Codes:

AX	*Description*
0	Successful.

Notes:

1. This date is set at run initialization time.
2. The date information can come from the last-used date, if no date is specified in the date field (bytes 88-95) in the initialization data.
3. If the PEDProtect keyword of the Run Profile is not active, the date can be changed by the operator from the Endorse Setup/Verify screen.
4. If characters in the endorse date field are not valid, question marks will print.

SpxGetAdditnlMsg

Purpose: The SpxGetAdditnlMsg function returns message text from the additional sort message area. See SpxPutAdditnlMsg.

Calling Sequence:

```
EXTRN  SpxGetAdditnlMsg:FAR

PUSH@  OTHER   buffer            ;Text of message
CALL   SpxGetAdditnlMsg
```

buffer

This is a 40-byte data area in which this function returns the additional sort message. This function always stores 40 bytes. If you specify a data area that is smaller than 40 bytes, you might cause a *protection violation* or lose other data.

Return Codes:

AX	*Description*
0	Successful.

SpxGetAddMsglong

Purpose: The SpxGetAddMsglong function returns the message data from the additional sort message area.

Calling Sequence:

```
EXTRN  SpxGetAddMsglong:FAR

PUSH@  OTHER   buffer            ;Text of message
CALL   SpxGetAddMsglong
```

buffer

This is an 80-byte data area in which the function stores the data copied from the additional sort message area of the document processor.

Return Codes:

AX	*Description*
0	Successful.

SpxGetAltINFArea

Purpose: The SpxGetAltINFArea function returns the alternate INF flag and the alternate item-number-feature (INF) data.

The alternate INF number is the user's responsibility. It must be in an unsigned, packed-decimal format, using all 10 digits. Blanks are specified as X'A'. A question mark (?) indicates that characters in the output are not valid. The invalid-INF-data bit is set on in the process buffer header to indicate that the
INF data is not valid and that it is available to the next document for action.

Calling Sequence:

```
EXTRN  SpxGetAltINFArea:FAR

PUSH@  OTHER   buffer           ;6-byte buffer
CALL   SpxGetAltINFArea
```

buffer

This is a 6-byte data area in which this function returns the alternate INF data and the alternate INF flag. The first byte is the flag; the remaining 5 bytes (10 digits) are the data.

The following flag values apply:

00 Alternate data is not used.
01 Alternate INF data is used.

Return Codes:

AX	*Description*
0	Successful.

Note: Initialization data (byte 82, bit 2) can inhibit this function.

SpxGetAutoSel

Purpose: The SpxGetAutoSel function returns the autoselect bits from the current document in the document processor or from the previous document during a motors stop exit.

Calling Sequence:

```
EXTRN  SpxGetAutoSel:FAR

PUSH@  BYTE    autosel          ;Autoselect bits
CALL   SpxGetAutoSel
```

autosel

This is a 1-byte data area in which this function returns the autoselect bits. The following values apply:

BIT	*Reason*
0	Multiple documents
1	Short gap
2	Long document
3	Short document
4	Short lead-edge to lead-edge (LE-LE)
5	Special symbol sequence error
6	Reserved
7	Reserved.

Return Codes:

AX	*Description*
0	Successful.

SpxGetCDT

Purpose: The SpxGetCDT function returns the code line definition table from mapped auxiliary storage.

Calling Sequence:

```
EXTRN  SpxGetCDT:FAR

PUSH@  OTHER   cdtbuff           ;CDT buffer
CALL   SpxGetCDT
```

cdtbuff

This is a 149-byte data area in which this function stores the code line definition table. The format is defined in Appendix C of the *3890/XP Enhanced Document Processor Programming Guide*.

Return Codes:

AX	*Description*
0	Successful.

SpxGetCtrlNum

Purpose: The SpxGetCtrlNum function returns the document processor Control Program release version. This appears on the “logo” screen and on the Main Menu.

Calling Sequence:

```
EXTRN  SpxGetCtrlNum:FAR

PUSH@  OTHER   vernum      ;Release version
CALL   SpxGetCtrlNum
```

vernum

This is a 6-character data area in which this function stores the Control Program version number.

Return Codes:

AX	*Description*
0	Successful.

SpxGetDisengage

Purpose: The SpxGetDisengage function returns the status of the disengage flag. See SpxDSG.

Calling Sequence:

```
EXTRN  SpxGetDisengage:FAR

PUSH@  BYTE    flag          ;Returned flag
CALL   SpxGetDisengage
```

flag

This is a 1-byte data area in which this function returns the status of the disengage flag. The following values apply:

00 Disengage is not requested.

01 Disengage was requested but is not yet acted on. It is acted on, where requested, after sort processing of the document. When the request is acted on, the flag is reset to X'00'.

Return Codes:

AX	*Description*
0	Successful.

SpxGetEncStatus

Purpose: The SpxGetEncStatus function returns the value of the power encoder status bytes. This function is supported on the 3892/XP.

Calling Sequence:

```
EXTRN  SpxGetEncStatus:FAR

PUSH@  OTHER    buffer          ;Address of returned value
CALL   SpxGetEncStatus
```

buffer

This is a 3-byte data area that will contain the power encoder status bytes.

Byte 1, general encoder status
- bit 7 = Verify disabled
- bit 6 = Encoder selected
- bit 5 = Reserved
- bit 4 = Jam detected
- bit 3 = Interlock open
- bit 2 = Reserved
- bit 1 = Verification threshold reached
- bit 0 = Data format error

Byte 2, path A status
- bit 7 = Ribbon low
- bit 6 = Ribbon not ready
- bit 5 = Ribbon not present
- bit 4 = Ribbon break or jam, capstan fault
- bit 3 = Hammer sync error
- bit 2 = Hammer hardware fault
- bit 1 = Hammer voltage fault
- bit 0 = Path disabled

Byte 3, path B status

Same as byte 2, but for path B

Return Codes:

AX	*Description*
0	Successful.
4	The function was issued while actively processing documents. No action was taken.
17	The feature is not enabled.
100	This function is not supported on this document processor.

SpxGetEncVStat

Purpose: The SpxGetEncVStat function returns the value of the power encoder verify status bytes. This function is supported on the 3892/XP.

Calling Sequence:

```
EXTRN  SpxGetEncVStat:FAR

PUSH@  OTHER    buffer          ;Address of returned value
CALL   SpxGetEncVStat
```

buffer

This is a 4-byte data area that will contain the power encoder verify status bytes.

Byte 1, verification reader status
- bit 7 = Reserved
- bit 6 = Reserved
- bit 5 = Reserved
- bit 4 = Reserved
- bit 3 = Verification disabled
- bit 2 = Reserved
- bit 1 = Reserved
- bit 0 = Reader sync error

Byte 2, path A status
- bit 7 = Reserved
- bit 6 = Cumulative substitution error
- bit 5 = Cumulative reject error
- bit 4 = Consecutive substitution error
- bit 3 = Consecutive reject error
- bit 2 = Substitution document dispositioned
- bit 1 = Reject document dispositioned
- bit 0 = Path disabled

Byte 3, path B status
- bit 7 = Reserved
- bit 6 = Cumulative substitution error
- bit 5 = Cumulative reject error
- bit 4 = Consecutive substitution error
- bit 3 = Consecutive reject error
- bit 2 = Substitution document dispositioned
- bit 1 = Reject document dispositioned
- bit 0 = Path disabled

Byte 4, count of saved verification error documents, 0 to 16

Return Codes:

AX	*Description*
0	Successful.
4	The function was issued while actively processing documents. No action was taken.
17	The feature is not enabled.
100	This function is not supported on this document processor.

SpxGetErd

Purpose:: The SpxGetErd function returns data from the External Reader Data buffer. It is supported on the 3891/XP and 3892/XP.

Calling Sequence:

```
EXTRN  SpxGetErd:FAR

PUSH@  OTHER   buffer            ;Data buffer
CALL   SpxGetErd
```

buffer

This is a 256-byte user data area in which this function returns the ERD data. The format is:

Byte	*Data Description*
0	Length of data
1	Data block ID, Hex B0
2-7	Reserved
8-255	Reader data for selected reader(s), in the form,

rdrlngth 1-byte length of the rdrdata that follows,

rdrdata Reader ID byte and document data from that reader

Reader ID bytes (in hex) are document data from that reader

E0 MICR-1
F0 OCR1 (RPQ)
F8 OCR2 (RPQ)
D8 Hi-Line/MICR2

Return Codes:

AX	Description
0	Successful
100	Function not supported by this document processor

Note: Data is valid only during one of the document time SPS events. There will be reader data for one or more readers.

SpxGetFeatures

Purpose: The SpxGetFeatures function returns the features-installed byte.

Calling Sequence:

```
EXTRN  SpxGetFeatures:FAR

PUSH@  BYTE    feature          ;Features installed
CALL   SpxGetFeatures
```

feature

This is a 1-byte data area in which this function returns the features-installed status. The following names apply:

BIT	*Feature*
0	INF
1	Endorse
2	Image Scanner
3	Microfilm
4	IBM OCR (OCR-3)
5-7	Reserved.

Return Codes:

AX	*Description*
0	Successful.

Note: This Spx returns the 3890/XP feature list and is applicable on all XP series machines. For a complete feature list on the 3891/XP and 3892/XP, see “SpxGet92Features” on page 2-7.

SpxGetHOZNum

Purpose: The SpxGetHOZNum function returns the high-order-zero (HOZ) correction control data (see byte 60 of the IREC).

Calling Sequence:

```
EXTRN  SpxGetHOZNum:FAR

PUSH@  BYTE    hozbyte          ;Correction control
CALL   SpxGetHOZNum
```

hozbyte

This is a 1-byte data area in which this function returns the HOZ correction control number.

Although one byte of data is returned, only the low-order 4 bits of data have significance. The high-order 4 bits are zeros. This is a binary integer that ranges from 0 to 15.

Return Codes:

AX	*Description*
0	Successful.

SpxGetImgBufPntr

Purpose: The SpxGetImgBufPntr function returns the 12 code line data match buffer pointers.

Calling Sequence:

```
EXTRN  SpxGetImgBufPntr:FAR

PUSH@  OTHER   pointer           ;Buffer for pointers
CALL   SpxGetImgBufPntr
```

pointer

This is a 48-byte data area in which this function returns the code line data match buffer pointers.

The following are the first six code line data match buffer pointers:

- Low limit
- High limit
- Search center
- Starter
- Wrap pointer
- Wrap reset.

When the extended code line data match is active, the next six pointers are copies of these first six pointers. SCII saves the first six pointers here and increments the first four pointers for a *not found* condition when the extended code line data match is active. Restoration is made if an extended code line data match call is made.

Return Codes:

AX	***Description***
0	Successful.

SpxGetINFNum

Purpose: The SpxGetINFNum function returns the INF number.

Calling Sequence:

```
EXTRN  SpxGetINFNum:FAR

PUSH@  OTHER   buffer           ;5-byte buffer
CALL   SpxGetINFNum
```

buffer

This is a 5-byte data area in which this function returns the INF number.

This function returns the INF number as an unsigned, packed-decimal number. For 3890 emulation, this 10-digit field is left-aligned; the eight leftmost (high-order) digits are the INF number; the two rightmost (low-order) digits are blanks (X'AA'). In the XP mode, all 10 digits can be the INF number, as the initialization data (IREC) specifies.

Return Codes:

AX	*Description*
0	Successful.

SpxGetInitFtr

Purpose: The SpxGetInitFtr function returns the initialize-feature-data (IFD) flag. See SpxIFD.

Calling Sequence:

```
EXTRN  SpxGetInitFtr:FAR

PUSH@  BYTE    flag          ;Address where the value is returned
CALL   SpxGetInitFtr
```

flag

This is a 1-byte data area in which this function returns the IFD flag. The following values apply:

hex 00 The flag is not set; IFD is not requested.
hex 01 The flag is set; IFD is requested.

Return Codes:

AX	*Description*
0	Successful.

SpxGetInitList

Purpose: The SpxGetInitList function returns the list of initialized features.

Calling Sequence:

```
EXTRN  SpxGetInitList:FAR

PUSH@  OTHER   featlist          ;Feature list
CALL   SpxGetInitList
```

featlist

This is a 1-byte data area in which this function returns the initialized feature data. The following values apply:

Bit	*Feature*
0	INF
1	Endorse
2	Image Scanner
3	Microfilm
4	IBM OCR (OCR-3)
5-7	Reserved.

Return Codes:

AX	*Description*
0	Successful.

Note: This Spx returns the 3890/XP feature list and is applicable on all XP series machines. For a complete feature list on the 3891/XP and 3892/XP, see “SpxGet92InitList” on page 2-8.

SpxGetInptBufDat

Purpose: The SpxGetInptBufDat function returns data from the input buffer.

Calling Sequence:

```
EXTRN  SpxGetInptBufDat:FAR

PUSH@  OTHER   buffer            ;Data buffer
CALL   SpxGetInptBufDat
```

buffer

This is a 296-byte data area in which this function returns the input buffer data. During sort processing, this is the data for the current document. When the motors are stopped, this is the data for the last document processed. The format is defined in Appendix C of the *3890/XP Enhanced Document Processor Programming Guide*.

Return Codes:

AX	*Description*
0	Successful.

SpxGetISFFeatCtl

Purpose: The SpxGetISFFeatCtl function returns the Image Scanner feature control byte for the Image Capture System.

Calling Sequence:

```
EXTRN SpxGetISFFeatCtl:FAR

PUSH@  OTHER      buffer     ;ISF control byte
CALL   SpxGetISFFeatCtl
```

buffer

A 1-byte buffer to receive the ISF control byte. The byte is defined by initialization data (IREC), byte 110.

Return Codes:

AX	*Description*
0	Successful.

SpxGetJamData

Purpose: The SpxGetJamData function returns jam data.

Calling Sequence:

```
EXTRN  SpxGetJamData:FAR

PUSH@  OTHER   jam         ;12-byte buffer
CALL   SpxGetJamData
```

jam

This is a 12-byte data area in which this function returns the exception record header that has the jam data. The format is described in the *3890/XP Enhanced Document Processor Programming Guide*.

Return Codes:

AX	*Description*
0	Successful.

SpxGetJamDetail

Purpose: The SpxGetJamDetail function returns document data for the last jam. It is supported on the 3891/XP and the 3892/XP.

Calling Sequence:

```
EXTRN     SpxGetJamDetail:FAR

PUSH@     OTHER   buffer                ;Address of 80-byte buffer
CALL      SpxGetJamDetail
```

buffer

This is an 80-byte data area that receives the document data. The format is as follows:

10 bytes:	Current document serial number. This consists of ten ASCII characters and is the last serial number sent to the Transport for INF, Alt INF, or microfilm.
10 bytes:	Reserved
10 bytes:	Reserved
10 bytes:	Reserved
10 bytes:	Serial number of the last document at the microfilm camera. If the device is not selected, or if no documents have reached the device since the last feed start, the response will be binary zeros.
4 bytes:	Reserved
6 bytes:	Reserved
10 bytes:	Reserved
10 bytes:	Reserved

Return Codes:

AX	***Description***
0	Successful.
4	Document-time SPS event, no action taken.
100	Function not supported by this document processor.

Note: This function can be called during the Motor Stop SPS event. Because the data returned pertains to the last jam, the user sort program should first call SpxGetJamInd to determine whether a jam has occurred.

SpxGetJamInd

Purpose: The SpxGetJamInd function returns the state of the jam indicator flag.

This flag is set to indicate that this is the first document after a jam. The flag is cleared as soon as this document is processed. If a jam caused the motors to stop, the flag is also on during the processing of the "motors stopping" SPS function. You can use the SpxGetJamData function to retrieve the exception record header. On the 3891/XP and 3892/XP, you can also use the SpxGetJamDetail to retrieve document data.

Calling Sequence:

```
EXTRN  SpxGetJamInd:FAR

PUSH@  BYTE    flag          ;Returned flag
CALL   SpxGetJamInd
```

flag

This is a 1-byte data area in which this function returns the status of the jam indicator flag. The following values apply:

hex 00 The flag is not set; jam has not occurred.
hex 01 The flag is set; jam has occurred.

Return Codes:

AX	*Description*
0	Successful.

SpxGetLCLTable

Purpose: The SpxGetLCLTable function returns an element from the level control label (LCL) table.

Calling Sequence:

```
EXTRN  SpxGetLCLTable:FAR

PUSH@  OTHER   lcl         ;LCL buffer
PUSH   WORD    element     ;Which element
CALL   SpxGetLCLTable
```

lcl

This is an 80-byte data area in which this function stores an element of the LCL table.

element

This is a 16-bit word value that specifies which LCL record to return. For example, a value of 1 returns the first 80-byte record.

To receive the complete LCL table, use a loop and increment the value of the element until the return code is 1. The following pseudo code is an example of this type of loop:

```
Do element = element +1 Until Ax = 1;
   Push@ Other;    /* buffer for return */
   Push  Element;  /* which LCL is req. */
   Call SPXGETLCLTABLE;
End;
```

The format is:

16 bytes: user label
8 bytes: file name
4 bytes: file extension (includes period)
52 bytes: file path.

See the LCL Run Profile keyword in the *3890/XP Enhanced Document Processor Programming Guide*.

Return Codes:

AX	*Description*
0	Successful
1	Not a valid element value.

SpxGetMicr2Data

Purpose: The SpxGetMicr2Data function returns the High Read feature (MICR2) input data. This function is supported on the 3892/XP.

Calling Sequence:

```
EXTRN  SpxGetMicr2Data:FAR

PUSH@  OTHER   buffer   ;Address of result buffer
CALL   SpxGetMicr2Data
```

buffer

This is a 266-byte data area within the caller's memory space where the MICR2 input data is returned.

The format is:

2 bytes:	hex 20 00 (valid data status)
2 bytes:	length of data
260 bytes:	document data, right justified
2 bytes:	hex 20 20 (ending pads).

Return Codes:

AX	*Description*
0	Successful.
12	Non-document SPS event, no action taken.
13	The requested feature is not installed.
16	Data not valid.
17	Feature not enabled.
100	This function is not supported on this document processor.

SpxGetModel

Purpose: The SpxGetModel function returns the XP Series document processor model type.

Calling Sequence:

```
EXTRN  SpxGetModel:FAR

PUSH@  BYTE    model    ;1-byte model type
CALL   SpxGetModel
```

model

This is a 1-byte data area in which this function returns the XP Series document processor model type. The following values apply:

Byte	*Document Processor*
1	3890/XP
2	3892/XP
3	3891/XP.
4	UT/XP.
5	3890/XP Enhanced

Return Codes:

AX	*Description*
0	Successful.
4	Document-time SPS event, no action taken.

SpxGetMP

Purpose: The SpxGetMP function returns the module destination and the pocket destination for the current document.

Calling Sequence:

```
EXTRN  SpxGetMP:FAR

PUSH@  WORD    buffer   ;Module-pocket destination
CALL   SpxGetMP
```

buffer

This is a 2-byte data area in which this function returns the module destination and the pocket destination. The following values apply:

Byte	*Destination*
1	Module, 1 through 6
2	Pocket, 1 through 6.

Return Codes:

AX	*Description*
0	Successful.

SpxGetNatModRC

Purpose: The SpxGetNatModRC function returns the OS/2 language (“native”) module return code.

Calling Sequence:

```
EXTRN  SpxGetNatModRC:FAR

PUSH@  WORD    retcode     ;Return code
CALL   SpxGetNatModRC
```

retcode

This is a 2-byte data area in which this function returns the current OS/2 module return code. Subsequent SPS event entry point modules can use this return code to determine how the previous module completed. The return code in AX is saved by the Control Program each time an SPS event module returns.

Return Codes:

AX	*Description*
0	Successful.

SpxGetNewArea

Purpose: The SpxGetNewArea function returns up to 4K bytes of data from the New Save Area in additional AUX storage.

Calling Sequence:

```
EXTRN  SpxGetNewArea:FAR

PUSH@  OTHER   buffer            ;Buffer
PUSH   WORD    displacement      ;Displacement
PUSH   WORD    length            ;Number of bytes
CALL   SpxGetNewArea
```

buffer
Is a data area to which data is copied.

displacement
This is the displacement within the extended save area of the first byte that is to be copied. For the displacement value to begin at the first byte, you should specify zero.

length
This is the number of bytes to be copied.

Return Codes:

AX	*Description*
0	Successful.
1	Displacement out-of-bounds (greater than 4095).
2	Displacement plus length out-of-range (greater than 4096).
3	Length is zero.

SpxGetNoDateIns

Purpose: The SpxGetNoDateIns function returns the status of the date-substitution-in-endorse-data flag.

Calling Sequence:

```
EXTRN  SpxGetNoDateIns:FAR

PUSH@  BYTE    flag          ;1-byte flag
CALL   SpxGetNoDateIns
```

flag

This is a 1-byte area in which this function returns the status of the date-substitution-in-endorse-data flag. The following values apply:

hex 00 No suppression; date substitution allowed.
hex 01 Suppressed; date substitution not allowed.

Return Codes:

AX	*Description*
0	Successful.

Note: The flag is reset at the next initialization.

SpxGetNoDWConv

Purpose: The SpxGetNoDWConv function returns the status of the double-wide-endorse-data-conversion flag.

Calling Sequence:

```
EXTRN  SpxGetNoDWConv:FAR

PUSH@  BYTE    flag          ;1-byte flag
CALL   SpxGetNoDWConv
```

flag

This is a 1-byte data area in which this function returns the status of the double-wide-endorse-data-conversion flag. The following values apply:

hex 00 No suppression; normal conversion allowed.
hex 01 Suppressed; conversion not allowed.

Return Codes:

AX	*Description*
0	Successful.

Note: The flag is reset at the next initialization.

SpxGetOCR1Data

Purpose: The SpxGetOCR1Data function returns the OCR1 input data. This function is supported on the 3891/XP and the 3892/XP.

Calling Sequence:

```
EXTRN  SpxGetOCR1Data:FAR

PUSH@  OTHER   buffer        ;Address of the data returned
CALL   SpxGetOCR1Data
```

buffer

This is a 266-byte data area within the caller's memory space in which this function returns the OCR1 input buffer data. The format is:

2 bytes:	hex 20 00 (valid data status)
2 bytes:	length of data
260 bytes:	document data, right justified, in ASCII
2 bytes:	hex 20 20 (ending pads).

Return Codes:

AX	*Description*
0	Successful.
12	Non-document SPS event; no action taken.
16	Data not valid.
17	Feature not enabled.
100	This function is not supported on this document processor.

SpxGetOCR2Data

Purpose: The SpxGetOCR2Data function returns the OCR2 input data. This function is supported on the 3891/XP and 3892/XP.

Calling Sequence:

```
EXTRN  SpxGetOCR2Data:FAR

PUSH@  OTHER   buffer        ;Address of the data returned
CALL   SpxGetOCR2Data
```

buffer

This is a 266-byte data area within the caller's memory space in which this function returns the OCR2 input buffer data. The format is:

2 bytes:	hex 20 00 (valid data status)
2 bytes:	length of data
260 bytes:	document data, right justified, in ASCII
2 bytes:	hex 20 20 (ending pads).

Return Codes:

AX	*Description*
0	Successful.
12	Non-document SPS event; no action taken.
16	Data not valid.
17	Feature not enabled.
100	This function is not supported on this document processor.

SpxGetOCR3Data

Purpose: The SpxGetOCR3Data function returns the OCR Feature input data. This function is supported on the 3890/XP.

Calling Sequence:

```
EXTRN  SpxGetOCR3Data:FAR

PUSH@  OTHER   buffer        ;Address of the data returned
CALL   SpxGetOCR3Data
```

buffer

This is a 296-byte data area within the caller's space in which this function returns the OCR3 input buffer data. The format is:

1 byte:	01 = valid data
1 byte:	00 = good read record
	01 = partial read record
	02 = no OCR data for this document
2 bytes:	data length
2 bytes:	Document length
28 bytes:	reserved
260 bytes:	document data, right justified, based on .OTT table
2 bytes:	reserved

Return Codes:

AX	*Description*
0	Successful.
12	Non-document SPS event; no action taken
16	Data not valid
17	Feature not enabled.

SpxGetOCR3Setup

Purpose: The SpxGetOCR3Setup function returns the IBM OCR-3 initialization data.

Calling Sequence:

```
EXTRN  SpxGetOCR3Setup:FAR

PUSH@  OTHER   OCR3_Setup        ;OCR-3 Setup parameters
CALL   SpxGetOCR3Setup
```

OCR3_Setup

This is a 5-byte data area in which this function returns the following setup data.

Byte	*Data Description*
0-1	Hexadecimal value indicating OCR3 Scan Area (Intel Format)
2	Font-1
3	Font-2
4	Vertical Bars and Spaces enabled

Bit	*Bit Description*
0	Spaces Enabled
1	Vertical Bars Enabled
2-7	Reserved

Bytes 2 and 3 are font identifier bytes. The following list represents the value returned for a given font.

Returned Value	*Font*
'00'X	No Font Set
'01'X	OCR A Numeric
'03'X	OCR B Numeric
'04'X	OCR B Alphanumeric (RPQ)
'05'X	E13B (RPQ)

Return Codes:

AX	*Description*
0	Successful.
17	Feature Not Enabled.

SpxGetOnOfflag

Purpose: The SpxGetOnOfflag function returns the status of the host online-or-offline flag.

Calling Sequence:

```
EXTRN  SpxGetOnOfflag:FAR

PUSH@  BYTE    flag          ;Address of the returned flag
CALL   SpxGetOnOfflag
```

flag

This is a 1-byte data area in which this function returns the host online-or-offline flag. The following values apply:

hex 00 The document processor is offline.
hex 01 The document processor is online.

Return Codes:

AX	*Description*
0	Successful.

SpxGetOpCmdLine

Purpose: The SpxGetOpCmdLine function returns the operator command line input.

Calling Sequence:

```
EXTRN  SpxGetOpCmdLine:FAR

PUSH@  OTHER   buffer            ;Address of 65-byte buffer
CALL   SpxGetOpCmdLine
```

buffer

This is a 65-byte data area in which this function returns the operator command line data. All 65 bytes are always transferred.

This function returns the command that this SPS module or a previous SPS module placed in the command field to be displayed on the Run screen.

Return Codes:

AX	*Description*
0	Successful.

SpxGetOpData

Purpose: The SpxGetOpData function returns operator-entered data.

Calling Sequence:

```
EXTRN  SpxGetOpData:FAR

PUSH@  OTHER   buffer      ;Address of 50-byte data area
CALL   SpxGetOpData
```

buffer

When the operator causes the operator SPS event to occur, the data from the command line following the EXec command will be available via this function. Entry is made to the Sort module specified by the Run Profile SPSOPER keyword.

Return Codes:

AX	*Description*
0	Successful.

SpxGetOpMsgNum

Purpose: The SpxGetOpMsgNum function returns the sort operator message (SOM) number.

Calling Sequence:

```
EXTRN  SpxGetOpMsgNum:FAR

PUSH@  BYTE    msgnum    ;Address of message number
CALL   SpxGetOpMsgNum
```

msgnum

This is a 1-byte field in which this function returns the current SOM number. Message number 0 indicates that the message has been cleared.

Return Codes:

AX	*Description*
0	Successful.

SpxGetOpMsgTxt

Purpose: The SpxGetOpMsgTxt function returns the text of the sort operator message.

Calling Sequence:

```
EXTRN  SpxGetOpMsgTxt:FAR

PUSH@  OTHER   text        ;Address of text
PUSH   BYTE    somnum      ;SOM number
CALL   SpxGetOpMsgTxt
```

text

This is an 81-byte data area in which this function returns the sort operator message (SOM) text. The first byte of the message text is the severity code:

I	Information
W	Warning
E	Error.

Format is ASCII.

somnum

This is the message number (SOM), in the range 1-255, that specifies which message text to return.

Return Codes:

AX	*Description*
0	Successful.
1	SOM number cannot be zero.

SpxGetPcktLim

Purpose: The SpxGetPcktLim function returns the pocket limit table.

Calling Sequence:

```
EXTRN  SpxGetPcktLim:FAR

PUSH@  OTHER   buffer            ;Buffer
CALL   SpxGetPcktLim
```

buffer

This is a 72-byte data area in which this function returns the pocket limit table. Upon completion, this data area contains 36 entries of 16 bits each from the pocket limit table. (For more information about the pocket limit table, see the section about additional auxiliary storage in the *3890/XP Enhanced Document Processor Programming Guide*.)

Return Codes:

AX	*Description*
0	Successful.

SpxGetPktCount

Purpose: The SpxGetPktCount function returns the pocket counters.

Calling Sequence:

```
EXTRN  SpxGetPktCount:FAR

PUSH@  OTHER   pktcntr           ;Pocket counters
CALL   SpxGetPktCount
```

pktcntr

This is a 72-byte data area in which this function returns 36 two-byte pocket counters in SCI format.

Although the pocket counter area is 96 bytes, the actual pocket counter data is 36 two-byte words. This function strips out unused bytes and compresses the data. (For more information about pocket counters, see the section about the accessible storage map for load/store SCIs in the *3890/XP Enhanced Document Processor Programming Guide*.)

Return Codes:

AX	*Description*
0	Successful.

SpxGetPktTblUsed

Purpose: The SpxGetPktTblUsed function returns the pocket limit flag.

Calling Sequence:

```
EXTRN  SpxGetPktTblUsed:FAR

PUSH@  BYTE    flag          ;Returned flag
CALL   SpxGetPktTblUsed
```

flag

This is a 1-byte data area in which this function returns the value of the pocket limit flag. The following values apply:

hex 00 The initialization data determines pocket limits for merge feed.

hex 01 The pocket table determines pocket limits for merge feed. See the Run Profile PKTDEFtbl keyword.

Return Codes:

AX	*Description*
0	Successful.

SpxGetPrgEndDat

Purpose: The SpxGetPrgEndDat function returns the document processor Active Endorsement Area (AEA) data.

Calling Sequence:

```
EXTRN  SpxGetPrgEndDat:FAR

PUSH@  OTHER   endorsedata       ;72-byte data
CALL   SpxGetPrgEndDat
```

endorsedata

This is a 72-byte data area in which this function returns the AEA data.

The returned data depends on the status of the document processor. If the document processor is not yet initialized, this function returns the last data loaded with the endorse command. After initialization, this function returns either the last data loaded with the endorse command or the data specified in the current run profile. This function returns valid data even if the endorser is not currently being used.

Note: Date substitution and double-wide conversion might have been performed. That means the data can be ASCII or double-wide transport codes. For the 3891/XP and the 3892/XP, double-wide characters are represented by “character-underscore” in ASCII.

Return Codes:

AX	*Description*
0	Successful.

SpxGetPrgStSelTb

Purpose: The SpxGetPrgStSelTb function returns the number of segments and selectors for program storage.

Calling Sequence:

```
EXTRN  SpxGetPrgStSelTb:FAR

PUSH@  OTHER              buffer            ;Buffer
CALL   SpxGetPrgStSelTb
```

buffer

This is a 322-byte data area in which this function returns the program storage table of selectors and the count of valid selectors. The count field and each selector occupy one word (2 bytes). Only the number of selectors specified in the count field are copied. Other selector fields in the receiving buffer are not changed and are not valid as selectors.

Return Codes:

AX	*Description*
0	Successful.

Comments: The table definition is:

Word 0	Count of valid selectors
Word 1	Selector for first segment of program storage
Word 2	Selector for second segment of program storage
Words 3 - 160	Other valid selectors in sequence.

To access program storage successfully, you must understand how to use the table. Program storage is allocated in segments of 64K bytes. At the time of allocation, a table of selectors is created to be used for addressability into the area. The table of selectors is preceded by a count of the number of 64K-byte segments allocated. To access storage within the first 64K-byte segment, you must use the first selector.

You could treat program storage as a continuous area that is addressed as a 32-bit integer offset from the first byte. When you do this, the number of the desired selector corresponds to the high-order word of the address (or the 32-bit offset divided by 64K) plus one. The offset into the segment is then the low-order word of the address (or the remainder of the 32-bit offset divided by 64K).

Be sure that you do not attempt to cross a 64K boundary. If you wish to access a block of storage that extends across a segment boundary, you must first access the portion addressed by the lower-numbered selector. Then the next-higher selector must be retrieved and used to access the remainder of the data. The function SpxGetProgStore allows you to obtain data that spans program storage segments as a single operation.

SpxGetPrgStSize

Purpose: The SpxGetPrgStSize function returns the size of program storage.

Calling Sequence:

```
EXTRN  SpxGetPrgStSize:FAR

PUSH@  DWORD   buffer           ;Buffer (32 bit Intel-format integer)
CALL   SpxGetPrgStSize
```

buffer

This is a 32-bit data area (32-bit Intel** format integer) in which this function returns the size of program storage as a fixed, 32-bit binary value.

Return Codes:

AX	*Description*
0	Successful.

** Trademark of Intel

SpxGetPrgSwtch

Purpose: The SpxGetPrgSwtch function returns the program switches data.

Calling Sequence:

```
EXTRN  SpxGetPrgSwtch:FAR

PUSH@  BYTE    buffer           ;Result buffer
CALL   SpxGetPrgSwtch
```

buffer

This is a 1-byte data area in which this function returns the program switches.

Note: These switches can be changed from the Run screen.

Return Codes:

AX	*Description*
0	Successful.

SpxGetProcBufDat

Purpose: The SpxGetProcBufDat function returns data from the process buffer.

Calling Sequence:

```
EXTRN  SpxGetProcBufDat:FAR

PUSH@  OTHER   proc        ;Buffer
CALL   SpxGetProcBufDat
```

proc
This is a 244-byte data area in which this function returns the process buffer data in a right-aligned format.

Note: Use SpxGetProcBufHdr to retrieve the process buffer header.

Return Codes:

AX	*Description*
0	Successful.

SpxGetProcBufDL

Purpose: The SpxGetProcBufDL function returns the length (in Intel format) of the process buffer.

Calling Sequence:

```
EXTRN  SpxGetProcBufDL:FAR

PUSH@  WORD    length      ;Buffer
CALL   SpxGetProcBufDL
```

length

This is a 2-byte area in which this function returns the length of the process buffer data as a 16-bit, binary integer.

Return Codes:

AX	*Description*
0	Successful.

Note: The length does NOT include the length of the process buffer header (length 12).

SpxGetProcBufHdL

Purpose: The SpxGetProcBufHdL function returns the length (in Intel format) of the process buffer header.

Calling Sequence:

```
EXTRN  SpxGetProcBufHdL:FAR

PUSH@  WORD    length      ;Buffer
CALL   SpxGetProcBufHdL
```

length

This is a 2-byte data area in which this function returns the length of the (12-byte) process buffer header as a 16-bit, binary integer.

Return Codes:

AX	*Description*
0	Successful.

Note: Use SpxGetProcBufHdr to retrieve the process buffer header.

SpxGetProcBufHdr

Purpose: The SpxGetProcBufHdr function returns the process buffer header.

Calling Sequence:

```
EXTRN  SpxGetProcBufHdr:FAR

PUSH@  OTHER   buffer      ;Buffer
CALL   SpxGetProcBufHdr
```

buffer

This is a 12-byte data area in which this function stores the process buffer header. See "Document Data Record Header" in the *3890/XP Enhanced Document Processor Programming Guide*.

Return Codes:

AX	*Description*
0	Successful.

SpxGetProgStore

Purpose: The SpxGetProgStore function returns data from program storage.

Calling Sequence:

```
EXTRN  SpxGetProgStore:FAR

PUSH@  OTHER   buffer          ;Result buffer
PUSH   DWORD   disp            ;Displacement
PUSH   WORD    length          ;Length of move
CALL   SpxGetProgStore
```

buffer
This is a data area in which this function returns the data from program storage.

disp
This is a 32-bit displacement into program storage.

length
This is the length in bytes of the data to be copied. It must be less than 65 536 bytes. To copy more than 65 535 bytes of data, use a series of function calls.

Return Codes:

AX	*Description*
0	Successful
1	Displacement value greater than program store
2	Displacement plus length out-of-bounds
3	Length not-greater-than zero.

Note: This function does not check for 3890-emulation mode but always returns the actual program storage data from the specified displacement.

SpxGetReinitLoadPath

Purpose: SpxGetReinitLoadPath returns the current value of the re-initialization load path.

Calling Sequence:

```
EXTRN  SpxGetReinitLoadPath:FAR

PUSH@  ASCIIZ  ReinitLoadPath
CALL   SpxGetReinitLoadPath
```

ReinitLoadPath
is the directory path to be used to load the run profile and associated files during re-initialization. ReinitLoadPath must be 66 bytes long to contain the maximum path value.

Return Codes:

AX	Description
0	Successful.

Notes:

1. Use **SpxPutReinitLoadPath** to change the path that will be used to load the run profile and associated files during re-initialization.
2. Use **SpxPutReinitReq** to actually request re-initialization during the SPSINIT event.

SpxGetReinitReq

Purpose: SpxGetReinitReq gets the current setting of the re-initialization request flag.

Calling Sequence:

```
EXTRN  SpxGetReinitReq:FAR

PUSH@  BYTE    ReinitReq
CALL   SpxGetReinitReq
```

ReinitReq
is a one-byte flag that indicates if re-initialization has been requested. Possible values are:

0 Re-initialization has not been requested

1 Re-initialization has been requested

Return Codes:

AX	Description
0	Successful.

Notes:

1. Use **SpxPutReinitReq** to request re-initialization during the SPSINIT event.

SpxGetRPERec

Purpose: The SpxGetRPERec function returns one Run Profile Element Record from the Program Storage Data (PSD) table in a 32-byte data area that the calling program specifies.

Calling Sequence:

```
EXTRN  SpxGetRPERec:FAR

PUSH@  OTHER   buffer            ;Buffer
PUSH   WORD    element           ;PSD record
CALL   SpxGetRPERec
```

buffer

This is a 32-byte data area in which this function returns the data. The format is:

8 bytes:	filename
8 bytes:	user tag
4 bytes:	offset in Program Storage - IBM format
4 bytes:	length - IBM format
4 bytes:	offset in Program Storage - Intel format
4 bytes:	length - Intel format.

element

This identifies the record element within the PSD table to be copied. A value of 1 copies the first record.

To get the complete table, use a loop and increment the value of the element until the return code is 1. The following pseudo code is an example of this type of loop:

```
Do element = element +1 Until Ax = 1;
   Push@ Other;    /* buffer for return */
   Push  Element;
   Call SPXGETRPEREC;
End;
```

Return Codes:

AX	*Description*
0	Successful
1	Element value out of range.

SpxGetRqNotInit

Purpose: The SpxGetRqNotInit function returns the value of the request-not-initialized flag.

This flag is reset by run initialization or by the SPXPutRQNotInit function.
(For more information about request-not-initialized, see the section about additional auxiliary storage in the *3890/XP Enhanced Document Processor Programming Guide*.)

Calling Sequence:

```
EXTRN  SpxGetRqNotInit:FAR

PUSH@  BYTE    flag          ;1-byte flag
CALL   SpxGetRqNotInit
```

flag

This is a 1-byte data area in which this function returns the value of the request-not-initialized flag. The following values apply:

hex 00 The flag is not set.
hex 01 The flag is set.

Return Codes:

AX	*Description*
0	Successful.

SpxGetSavArea

Purpose: The SpxGetSavArea function returns up to 256 bytes of data from the Save Area.

Calling Sequence:

```
EXTRN  SpxGetSavArea:FAR

PUSH@  OTHER   buffer            ;Result buffer
PUSH   WORD    displacement      ;Displacement
PUSH   WORD    length            ;Number of bytes
CALL   SpxGetSavArea
```

buffer
This is a data area to which the data is copied.

displacement
This is the displacement within the save area of the first byte to be copied (specify 0 to begin at the first byte). The data is copied, beginning at the low-order address byte and proceeding to the high-order address bytes, that is, left to right.

length
This is the number of bytes to be copied.

Return Codes:

AX	*Description*
0	Successful.
1	The displacement was out-of-bounds (greater than 255).
2	The displacement plus the length was out-of-range (greater than 256).
3	The length was not greater than zero.

SpxGetSCICount

Purpose: The SpxGetSCICount function returns the values of the SCI counters.

Calling Sequence:

```
EXTRN  SpxGetSCICount:FAR

PUSH@  OTHER   counter           ;SCI counters
CALL   SpxGetSCICount
```

counter

This is a 32-byte data area in which this function returns the SCI counters in SCI (IBM) format. Each counter is 2-bytes long.

Return Codes:

AX	*Description*
0	Successful.

SpxGetSCIErrData

Purpose: The SpxGetSCIErrData function returns the SCI error data.

Calling Sequence:

```
EXTRN  SpxGetSCIErrData:FAR

PUSH@  OTHER   buffer            ;12-byte buffer
CALL   SpxGetSCIErrData
```

buffer

This is a 12-byte data area in which this function returns the SCI error data. See "SCI-Error Record Header" in the *3890/XP Enhanced Document Processor Programming Guide*.

Return Codes:

AX	*Description*
0	Successful.

SpxGetSEC

Purpose: The SpxGetSEC function returns the symbol-error-correction-value byte.

Calling Sequence:

```
EXTRN  SpxGetSEC:FAR

PUSH@  BYTE    secbyte          ;SEC byte
CALL   SpxGetSEC
```

secbyte
This is a 1-byte data area in which this function returns the symbol-error-correction value. This is the same as initialization data byte 55.

Return Codes:

AX	*Description*
0	Successful.

SpxGetSerial

Purpose: The SpxGetSerial function returns the machine serial number.

Calling Sequence:

```
EXTRN  SpxGetSerial:FAR

PUSH@  WORD    serial           ;Serial number
CALL   SpxGetSerial
```

serial

This is a 3-byte data area in which this function returns the machine serial number in unsigned, packed-decimal format. The 3890/XP uses a 5-digit serial number with a leading 0. The 3891/XP and the 3892/XP use a 6-digit serial number. The machine serial number is displayed on the Main Menu.

The UT/XP does not support a manufacturing serial number. For this function the UT/XP returns to the control program a descriptive name that may contain non-numeric data.

Return Codes:

AX	*Description*
0	Successful.
100	This function is not supported on this document processor.

SpxGetSpxEvent

Purpose: The SpxGetSpxEvent function returns the Sort Program Sequence event ID.

Calling Sequence:

```
EXTRN  SpxGetSpxEvent:FAR

PUSH@  OTHER   eventID             ;SPS event ID
CALL   SpxGetSpxEvent
```

EventID

This is a 16-byte data area in which this function returns the Sort program event ID.

Return Codes:

AX	*Description*
0	Successful.

Each event is represented by a byte. A value of 1 indicates the current event. The events, in order, are:

0 SPSVERIFY
1 SPSCORR
2 SPSFORMAT
3 SPSIMATCH
4 SPSDOC
5 SPSPOST
6-11 Reserved events
12 SPSPAUSE
13 SPSINIT
14 SPSMS
15 SPSOPER.

SpxGetStackers

Purpose: The SpxGetStackers function returns the number of stackers installed.

Calling Sequence:

```
EXTRN  SpxGetStackers:FAR

PUSH@  BYTE    stacker          ;Stackers installed
CALL   SpxGetStackers
```

stacker
This is a 1-byte data area in which this function returns the number of stackers installed.

Return Codes:

AX	*Description*
0	Successful.

SpxGetUsrMPDef

Purpose: The SpxGetUsrMPDef function returns the 36 three-byte module-pocket names defined in the Machine Profile.

Calling Sequence:

```
EXTRN  SpxGetUsrMPDef:FAR

PUSH@  OTHER   buffer            ;Module-pocket names
CALL   SpxGetUsrMPDef
```

buffer

This is a 108-byte data area in which this function returns the module-pocket data table (in ASCII format).

Return Codes:

AX	*Description*
0	Successful.

SpxGetUsrStat

Purpose: The SpxGetUsrStat function returns user statistics data.

Calling Sequence:

```
EXTRN  SpxGetUsrStat:FAR

PUSH@  BYTE   buffer            ;Result buffer
CALL   SpxGetUsrStat
```

buffer

This is a 1-byte data area in which this function returns the user statistics data ("user stats").

Return Codes:

AX	*Description*
0	Successful.

SpxGetVerNum

Purpose: The SpxGetVerNum function returns the SPXServ version number.

Calling Sequence:

```
EXTRN  SpxGetVerNum:FAR

PUSH@  OTHER   vernum     ;Version number
CALL   SpxGetVerNum
```

vernum

This is a 5-character data area in which this function returns the SPXServ version number. This number is an ASCII string of 5 characters in the format NN.NN. The first two characters contain the portion of the version number that is greater than or equal to 1 and the last two characters contain the portion of the version number that is less than 1, but greater than or equal to zero (for example, Version 01.05).

Return Codes:

AX	*Description*
0	Successful.

SpxGetWrkReg

Purpose: The SpxGetWrkReg function returns the contents of the work register. SCI programs use the work register as a temporary storage area to manipulate data.

Calling Sequence:

```
EXTRN  SpxGetWrkReg:FAR

PUSH@  OTHER   buffer            ;16-byte buffer
CALL   SpxGetWrkReg
```

buffer

This is a 16-byte data area in which this function returns the contents of the work register.

Return Codes:

AX	*Description*
0	Successful.

SpxIFD

Purpose: The SpxIFD function requests the Initialize Feature Data function.

Calling Sequence:

```
EXTRN  SpxIFD:FAR

CALL   SpxIFD
```

Return Codes:

AX	*Description*
0	Successful.
1	Machine was online.

Notes:

1. This function runs only when the document processor is offline.
2. This function is the same as the SCI “IFD 0” macro, except that it does not store the mark register low-order byte in byte 0 of initialization data (as SCI does). You should use the SpxPutProgStore function to completely emulate the SCI “IFD 0” macro.

SpxLoadFixM

Purpose: The SpxLoadFixM function loads a fixed message to be used for endorsement. This function is called at SPSINIT-time. Use of the message is specified at SPSDOC-time. See SpxFixM. This function is only supported on the 3891/XP, 3892/XP, and UT/XP.

Calling Sequence:

```
EXTRN  SpxLoadFixM:FAR

PUSH@  OTHER   buffer            ;Message
CALL   SpxLoadFixM
```

buffer

This is a 50-byte data area that has the message. The first byte is the message number (1-15), followed by the text (0 to 48 ASCII), ended by hex 7F. A null message is defined without text; however, the 7F is required. If a null message is "printed" (see SpxFixM), it occupies no space on the document. On the UT/XP this is a 68 byte data area that has the message. The first byte is the message number (1-15), followed by the text (0 to 66 ascii bytes), ended by hex 7F.

Return Codes:

AX	*Description*
0	Successful.
1	Invalid message number.
18	Command failed.
100	This function is not supported on this document processor.

SpxLogMrg

Purpose: The SpxLogMrg logical merge function requests document operations without a merge feeder: merge before main, merge on pocket count, and feed merge. This function is only supported on the 3891/XP and the 3892/XP.

Calling Sequence:

```
EXTRN  SpxLogMrg:FAR

PUSH@  BYTE    buffer            ;Flag
CALL   SpxLogMrg
```

buffer

A 1-byte data area in which this function returns the Logical Merge Requested flag.

hex 00 Not requested.
hex 01 Requested. User program should start a tolerance countdown.

Return Codes:

AX	*Description*
0	Successful.
100	This function is not supported on this document processor.

Notes:

1. Logical merge mode is entered by calling this function, regardless of whether the machine has a merge feeder or not. The Control Program will not request merge feeder documents from the hardware. Instead, the operator inserts merge documents in the main feeder work stream. This mode is reset at the next initialization.
2. For merge before main operation, this function must be called at SPSINIT time, so logical merge mode is set before the control routine starts feed requests.
3. It is recommended that the user's tolerance count be applied on a per-pocket basis. When the count is reached and a merge document has not been seen (for that pocket), the user program must decide whether to stop the machine and what to tell the operator (SpxDSG and SpxPutAdditnlMsg).
4. The user program should look at the current document (SpxGetProcBufDat) and determine if it is a merge document. If so, the merge flag should be set in the header (byte 8 bit 4, via SpxGetProcBufHdr and SpxPutProcBufHdr). This must be done as the last module at SPSFORMAT-time prior to execution of the remaining Sort program logic.
5. If the mode is active, the logical-merge-requested flag is set whenever there is a request for a merge document (SpxFM), including merge before main. This function clears the flag after returning it to the caller.

SpxNoOP

Purpose: The SpxNoOP function allows the Control Program to check for time out or late TSCI (terminate stacker control instructions) termination.

The Control Program makes a time-out check at the start of every Spx function. A Sort program that has a long-running loop without other Spx calls should include SpxNoOp as part of the loop.

Calling Sequence:

```
EXTRN  SpxNoOP:FAR

CALL   SpxNoOP
```

Return Codes:

AX	*Description*
0	Successful.

SpxNoRollOn

Purpose: The SpxNoRollOn function will disable the Rollon Endorser on a per document basis. This function is supported only on the 3891/XP and 3892/XP.

This function will set a bit in EGA that will allow the control program to NOT send the ACTIVE command to the traansport. This bit will be reset every document after it is tested.

Calling Sequence:

```
EXTRN  SpxNoRollOn:FAR

CALL   SpxNoRollOn
```

Return Codes:

AX	*Description*
0	Successful.
12	Non-document SPS event; no action taken.
100	This function is not supported on this document processor.

SpxPause

Purpose: SpxPause sets the SPS pause request flag.

This function sets a pause request, which stops the feeding of documents but allows documents that are in process to continue processing normally.

Once the in-process documents have processed and the sorter has paused, the control routine calls the user Sort program at the SPSPAUSE entry points, as specified in the Run Profile. This is a non-document event, and the Run Profile SPXNDPTIME parameter applies.

When control has returned from the user routines, document feeding resumes. However, if a transport motor time-out has occurred, a message is displayed on the screen and the operator has to press the START key to restart the flow of documents.

Calling Sequence:

```
EXTRN  SpxPause:FAR

CALL   SpxPause
```

Return Codes:

AX	*Description*
0	Successful.
12	Non-document SPS event; no action taken.

SpxPutAdditnlMsg

Purpose: The SpxPutAdditnlMsg function stores a message in the additional sort message area.

Calling Sequence:

```
EXTRN  SpxPutAdditnlMsg:FAR

PUSH@  OTHER   buffer            ;Text of message
CALL   SpxPutAdditnlMsg
```

buffer

This is a 40-byte data area from which the message text is copied to the additional sort message area of the document processor.

If you specify less than 40 bytes, you might receive a protection violation message and lose data.

The message appears on the operator Run screen alternate message line.

Return Codes:

AX	*Description*
0	Successful.

SpxPutAddMsglong

Purpose: The SpxPutAddMsglong function stores a message in the additional sort message area.

Calling Sequence:

```
EXTRN  SpxPutAddMsglong:FAR

PUSH@  OTHER   buffer            ;Text of message
CALL   SpxPutAddMsglong
```

buffer

This is an 80-byte data area from which the message text is copied to the additional sort message area of the document processor.

The message will appear on the operator Run screen alternate message line.

Return Codes:

AX	*Description*
0	Successful.

SpxPutAltINFArea

Purpose: The SpxPutAltINFArea function stores data in the alternate INF flag and data area.

The alternate INF number must be in unsigned, packed-decimal format. Specify blanks as X'A' and use all 10 digits. A question mark (?) indicates that characters in the output are not valid. The invalid-INF-data bit is set on in the process buffer header to indicate that INF data is not valid.

Calling Sequence:

```
EXTRN  SpxPutAltINFArea:FAR

PUSH@  OTHER   buffer           ;6-byte buffer
CALL   SpxPutAltINFArea
```

buffer

This is a 6-byte data area from which this function copies the alternate INF flag and data. The first byte is the flag; the remaining 5 bytes are the 10 digits, in BCD.

The following flag values apply:

00 Alternate data is not used.
01 Alternate INF data is used.

Note: The initialization data (byte 82, bit 2) can inhibit this function. This function clears the flag after usage. The flag must be set on for each document that is to be printed with the alternate INF.

Return Codes:

AX	*Description*
0	Successful.

SpxPutCDT

Purpose: The SpxPutCDT function copies code-line definition data to the table in auxiliary storage.

Calling Sequence:

```
EXTRN  SpxPutCDT:FAR

PUSH@  OTHER   buffer            ;CDT buffer
CALL   SpxPutCDT
```

buffer

This is a 149-byte data area from which this function copies the code-line definition data. The format is defined in Appendix C of the *3890/XP Enhanced Document Processor Programming Guide*.

Return Codes:

AX	*Description*
0	Successful.

Note: This function allows the user Sort program to support multi-format code lines. See SpxGetCDT.

SpxPutEncData

Purpose: The SpxPutEncData function copies data to the 3892/XP power encoder buffer data.

Calling Sequence:

```
EXTRN  SpxPutEncData:FAR

PUSH@   OTHER    buffer          ;Address of the 85-byte data buffer
CALL   SpxPutEncData
```

buffer

This is an 85-byte data area set up by the calling program.

The data is in ASCII, left-justified in the buffer, and ended by a hex 7F. Unused bytes after the 7F are ignored. Encoding will be right-justified on the document. Multiple spaces can be represented by a tab sequence of 09 *count*, where *count* is the number of spaces required.

Return Codes:

AX	*Description*
0	Successful.
8	An invalid value was specified.
17	The feature is not enabled.
19	The hex 7F data terminator is missing.
100	This function is not supported on this document processor.

Notes:

1. You should ensure that the data does not exceed the permissible maximum for the given document size.
2. Valid characters are:

Hex	Character
09	Tab
20	Space
23	
24	
2D	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
3C	

3. If you execute this function on the UT/XP the SpxPutHSPEData function is called to execute the actual request. The return code for that function should be used to determine the results of the request.

SpxPutEncErrDisp

Purpose: The SpxPutEncErrDisp function sets the disposition for Power Encoder verification error documents. This function is supported on the 3892/XP and the UT/XP.

Calling Sequence:

```
EXTRN  SpxPutEncErrDisp:FAR

PUSH@   OTHER    buffer          ;Address of disposition data
CALL   SpxPutEncErrDisp
```

buffer

This is 3 bytes of data used to set up the document disposition for reject, character substitutions, and device errors.

Byte 1 Disposition mode for reject documents
0 = Pocket normally, ignore error
1 = Place in reject pocket

Byte 2 Disposition mode for character substitution documents
0 = Pocket normally, ignore error
1 = Place in reject pocket

Byte 3 Disposition mode for encoder device error documents
0 = Pocket normally, ignore error
1 = Place in reject pocket

Return Codes:

AX	*Description*
0	Successful.
4	The function was issued while actively processing documents. No action was taken.
8	An invalid value was specified.
17	The feature is not enabled.
100	This function is not supported on this document processor.

SpxPutEncSetup

Purpose: The SpxPutEncSetup function specifies the Power Encoder verification failure thresholds. This function is supported on the 3892/XP and the UT/XP.

Calling Sequence:

```
EXTRN  SpxPutEncSetup:FAR

PUSH@   OTHER    buffer          ;Address of the data
CALL   SpxPutEncSetup
```

buffer

This is a 4-byte area set up by the calling program that contains the power encoder setup parameters.

Byte 1 Threshold value for consecutive document reject errors, zero to 255 documents.

Byte 2 Threshold value for consecutive document substitution errors, zero to 255 documents.

Byte 3 Threshold value for cumulative document reject errors, zero to 255 documents.

Byte 4 Threshold value for cumulative document substitution errors, zero to 255 documents.

Return Codes:

AX	*Description*
0	Successful.
4	The function was issued while actively processing documents. No action was taken.
17	The feature is not enabled.
100	This function is not supported on this document processor.

Note: If you execute this function on a UT/XP the SpxPutHSPESetup function is called to execute the request. The return codes for that function should be used to determine the results of the request.

SpxPutHOZNum

Purpose: The SpxPutHOZNum function stores a 1-byte HOZ correction control number (see byte 60 of IREC).

Calling Sequence:

```
EXTRN  SpxPutHOZNum:FAR

PUSH   BYTE    hozbyte          ;Correction control
CALL   SpxPutHOZNum
```

hozbyte

This is an actual value that is written to the HOZ correction byte in the document processor. Any change to this value takes effect on the next document.

Although one byte of data is written, only the low-order 4 bits of data have any significance. The high-order 4 bits are zeros. This is a binary integer that ranges from 0 to 15.

Return Codes:

AX	*Description*
0	Successful.

SpxPutHSPEData

Purpose: The SpxPutEncData function copies data to the UT/XP power encoder buffer data.

Calling Sequence:

```
EXTRN  SpxPutHSPEData:FAR

PUSH@   OTHER    buffer          ;Address of the 66-byte data buffer
CALL   SpxPutEncData
```

buffer

This is an 66-byte data area set up by the calling program.

The data is in ASCII, left-justified in the buffer, and ended by a hex 7F. Unused bytes after the 7F are ignored. Encoding will be right-justified on the document. Multiple spaces can be represented by a tab sequence of 09 *count*, where *count* is the number of spaces required.

Return Codes:

AX	*Description*
0	Successful.
8	An invalid value was specified.
17	The feature is not enabled.
19	The hex 7F data terminator is missing.
20	Number of characters greater than 17 with only one encoder configured or with 2 encoders configured and running at 1000 docuemnts per minute.
21	Number of printable characters specified less than 20 with 2 encoders configured.
22	Number of printable characters specified less than 10 with a minimum of 10 required for either encoder.
23	Number of characters specified greater than 34 with 2 encoders configured.
100	This function is not supported on this document processor.

Notes:

1. You should ensure that the data does not exceed the permissible maximum for the given document size.
2. Valid characters are:

Hex	Character
09	Tab
20	Space
23	
24	
2D	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
3C	

3. If you execute this function on the 3892/XP the SpxPutEncData function is called to execute the actual request. The return code for that function should be used to determine the results of the request.

SpxPutHSPESetup

Purpose: The SpxPutHSPESetup function specifies the Power Encoder verification failure thresholds. This function is supported on the 3892/XP and the UT/XP.

Calling Sequence:

```
EXTRN  SpxPutHSPESetup:FAR

PUSH@   OTHER    buffer          ;Address of the data
CALL   SpxPutHSPESetup
```

buffer

This is a 4-byte area set up by the calling program that contains the power encoder setup parameters.

Byte 1 Threshold value for consecutive document reject errors, zero to 255 documents.

Byte 2 Threshold value for consecutive document substitution errors, zero to 255 documents.

Byte 3 Threshold value for cumulative document reject errors, zero to 255 documents.

Byte 4 Threshold value for cumulative document substitution errors, zero to 255 documents.

Byte 5 Value for feed rate for encoding; 1=600 DPM, 2=1000 DPM. When encoding more than 17 characters on the UT/XP the feed rate must be set to 600 DPM.

Return Codes:

AX	*Description*
0	Successful.
4	The function was issued while actively processing documents. No action was taken.
17	The feature is not enabled.
18	Request to set feed rate failed.
100	This function is not supported on this document processor.

Note: If you execute this function on a 3892/XP the SpxPutEncSetup function is called to execute the request. The return codes for that function should be used to determine the results of the request. Byte 5 will not be passed to SpxPutEncSetup.

SpxPutImgBufPntr

Purpose: The SpxPutImgBufPntr function copies data to the 12 code line data match buffer pointers in the document processor.

Calling Sequence:

```
EXTRN  SpxPutImgBufPntr:FAR

PUSH@  OTHER   pointer           ;Address
CALL   SpxPutImgBufPntr
```

pointer

This is a data area of twelve 32-bit pointers, in Intel format, that is used to control the code line data match and the extended code line data match.

Only pointers 1-4 and 7-10 are copied.

The following are the code line data match buffer pointers:

- Low limit
- High limit
- Search center
- Starter
- Wrap pointer (protected)
- Wrap reset (protected).

When the extended code line data match is active, the next six pointers are copies of these first six pointers. SCII saves the first six pointers and increments the first four pointers for a not-found condition when extended code line data match is active. Restoration is made if an extended code line data match call is made. This function protects the wrap pointer and the wrap reset from any alteration.

Warning: You must change these pointers by an amount that is equal to the record length (process-buffer-data length plus process-buffer-header length) or a multiple of the record length. You should check these pointers after you change them to see whether they are equal to or greater than the wrap pointer. If they are, the pointer that is nearest the wrap value must be reset to the wrap reset value. If you do not manage the pointers correctly, a host interface failure or *protection violation* can occur. Failure could require a system reload to recover from the error.

Return Codes:

AX	*Description*
0	Successful.

SpxPutInkJetCtrl

Purpose: The SpxPutInkJetCtrl function allows the user program to control the front and rear ink jet endorsement printers on the 3891/XP, 3892/XP and the UT/XP.

Calling Sequence:

```
EXTRN  SpxPutInkJetCtrl:FAR

PUSH@  OTHER     buffer             ;Address of the 51-byte data buffer for
                                     3891/XP and 3892/XP.  Address of the
                                     69-byte data buffer for the UT/XP.
CALL   SpxPutInkJetCtrl
```

buffer

A 69-byte data area is described below.

Byte	Bit	Description
1		Print line and side request X'00' = Front side line 1 X'01' = Back side line 1 X'02' = Front side line 2 X'03' = Back side line 2 X'04' = Front side line 3 X'05' = Back side line 3
2	0	Print the line specified in byte 1 inverted and rotated.
	1	Print the following variable text as specified in bytes 3 through 51.
	2	Print fixed message number, per bits 4 through 7.
	3	Print the document serial number (INF) in the sequence defined by SpxPutPrtSeq.
	4-7	Fixed message number (binary, 1 through F).
3 - 51		Variable Text for the 3891/XP and 3892/XP, in ASCII The variable text string must be terminated with a byte equal to X'7F'. This area is interrogated only when byte 2, bit 1 is on. The maximum allowed is 48 variable text characters on a single line. The maximum characters per document is 64.
3 - 69		Variable Text for the UT/XP The maximum allowed is 66 variable text characters on a single line.

Return Codes:

AX	*Description*
0	Successful.
1	The print line and side value in byte 1 are incorrect.
3	Variable text was specified, but no terminator was found.
4	A fixed message was requested but byte 2, bits 4 through 7 are equal to 0.
5	Line 2 or 3 was specified while requesting a large font.
17	The front side ink jet was requested but is not enabled.

18 The back side ink jet was requested but is not enabled.
100 This function is not supported on this document processor.

Notes:

1. You should ensure that the variable data does not exceed the maximum of 64 characters per document (including both front and back side printing). If you request more than 64 variable characters, the first 64 sent to the transport are used on the document; the remainder are ignored. On the UT/XP the maximum amount of text is 158 variable characters. The sequence of the endorsement lines sent to the transport is:
   - Front side line 1
   - Back side line 1
   - Front side line 2
   - Back side line 2
   - Front side line 3
   - Back side line 3.
2. The Control Program does not clear the endorsement lines between documents. The data is cleared only after Run initializations. To clear an endorsement line between documents, call the SpxPutInkJetCtrl function with a buffer in which the first byte identifies the endorsement side and line being cleared, and the second byte contains hex 00.
3. If you specify the large ink jet font (IREC byte 64, bits 4 and 5), you can only specify one line per document side and that line must be line 1. You cannot specify large ink jet font on the UT/XP.
4. If you request fixed messages, you must first use SpxLoadFixM to load the fixed message being requested.
5. The serial number (INF) can be on multiple lines.
6. The SpxPutInkJetCtrl function has the following additional characteristics:
   - Ignores endorsement text defined by the IBM-supplied converge DLL.
   - Right justifies all printing to the currently defined position.
   - Truncates (without error indication) any fixed message or variable text that is too long to fit on the specified document.
   - Prints as spaces any hexadecimal ASCII codes that are not recognized.
   - Can be used at SPSINIT event time, allowing a sort program to establish an endorsement at a noncritical time and only make changes during critical processing time.
   - Allows any combination of variable text, fixed message, and serial number printing. Use SpxPutPrtSeq to establish the left-to-right print sequence of this data.
   - Specifies only one line during each call.

Related Spx Functions

SpxPutAltINFArea: This function stores data in the alternate INF flag and data area. This value is the same as the transport serial number and is used both in the endorsement and on the microfilm.

SpxLoadFixM: This function allows you to specify fixed messages during the initialization process.

SpxFixM: This function can be used for all fixed message endorsements as well as fixed message support for converged endorsements. If SpxFixM and SpxPutInkJetCtrl are used to specify the same line of endorsement, the transport returns an option state error and nothing is printed.

SpxPutPrtSeq: This function allows you to change the sequence of the endorsement data elements:

- Fixed message
- Variable text
- Serial number or INF.

Incompatible Spx Functions

SpxGetPrgEndDat: This function gets a copy of the programmable endorse data from the 3890 data area. On a 3891/XP or 3892/XP machine, this function only gets usable data when used with the converge DLL.

SpxPutPrgEndDat: This function only handles valid data when used in conjunction with the converge DLL.

SpxPutInptBufDat

Purpose: The SpxPutInptBufDat function copies data to the input buffer.

Calling Sequence:

```
EXTRN  SpxPutInptBufDat:FAR

PUSH@  OTHER   buffer            ;Data buffer
CALL   SpxPutInptBufDat
```

buffer

This is a 296-byte data area that contains the data to be copied.

Return Codes:

AX	*Description*
0	Successful.

SpxPutISFFeatCtl

Purpose: The SpxPutISFFeatCtl function copies the Image Scanner feature (ISF) control byte for the Image Capture System.

Calling Sequence:

```
EXTRN  SpxPutISFFeatCtl:FAR

PUSH   BYTE    byte              ;ISF Control byte
CALL   SpxPutISFFeatCtl
```

byte
The ISF control byte. It is defined by initialization data byte 110.

Return Codes:

AX	*Description*
0	Successful.

SpxPutNewArea

Purpose: The SpxPutNewArea function copies up to 4K bytes of data to the New Save Area in additional AUX storage.

Calling Sequence:

```
EXTRN  SpxPutNewArea:FAR

PUSH@  OTHER   buffer            ;Result buffer
PUSH   WORD    displacement      ;Displacement
PUSH   WORD    length            ;Number of bytes
CALL   SpxPutNewArea
```

buffer
This is a data area from which the data is copied.

displacement
This is the displacement within the extended save area of the first byte that is to be copied. Specify 0 to begin the copy at the first byte.

length
This is the number of bytes that are copied.

Return Codes:

AX	*Description*
0	Successful.
1	Displacement out-of-bounds — (greater than 4095).
2	Displacement plus length is out-of-range — (greater than 4096).
3	Length is zero.

SpxPutNoDateIns

Purpose: The SpxPutNoDateIns function sets or clears the suppress-date-substitution-in-endorse-data flag.

If no further endorse changes are required after the first document is processed, you can set this flag to prevent unneeded processing of date substitution on subsequent documents.

Calling Sequence:

```
EXTRN  SpxPutNoDateIns:FAR

PUSH   BYTE    flag          ;1-byte value
CALL   SpxPutNoDateIns
```

flag

This is a 1-byte data area. The following values apply:

Value	*Description*
0	No suppression; the date is substituted.
1	Suppression; the date is not substituted.

Return Codes:

AX	*Description*
0	Successful.

SpxPutNoDWConv

Purpose: The SpxPutNoDWConv function sets or clears the double-wide-endorse-data-conversion flag.

If no further endorse changes are required after the first document is processed, you can set this flag to prevent unneeded processing of double-wide conversion on subsequent documents.

Calling Sequence:

```
EXTRN  SpxPutNoDWConv:FAR

PUSH   BYTE    flag          ;1-byte value
CALL   SpxPutNoDWConv
```

flag

This is a 1-byte data area. The following values apply:

Value	*Description*
0	No suppression; normal conversion.
1	Suppression; no conversion.

Return Codes:

AX	*Description*
0	Successful.

SpxPutOCRSetup

Purpose: The SpxPutOCRSetup function sets the OCR1 and OCR2 setup parameters. This function is supported on the 3891/XP and 3892/XP. The function defines up to two zones on a document for OCR reading and selects a font for each zone.

Calling Sequence:

```
EXTRN  SpxPutOCRSetup:FAR

PUSH@  OTHER   buffer            ;Address of the buffer
CALL   SpxPutOCRSetup
```

buffer

This is an 11-byte data area that has the setup parameters.

ocrspec	1	OCR reader specifier 0 = OCR1 1 = OCR2
zone1def		zone 1 definition
start-tenths	2	zone start position (tenths component)
start-intg	3	zone start position (integer component)
fontspec	4	font specifier (See Note 6.)
stop-tenths	5	zone stop position (tenths component)
stop-intg	6	zone stop position (integer component)
zone2def		zone 2 definition
start-tenths	7	zone start position (tenths component)
start-intg	8	zone start position (integer component)
fontspec	9	font specifier (See Note 6.)
stop-tenths	10	zone stop position (tenths component)
stop-intg	11	zone stop position (integer component)

Return Codes:

AX	***Description***
0	Successful.
6	Invalid reader specified; not 0 or 1.
8	An invalid zone definition value was specified.
14	The requested font is not installed.
15	The zone specification is incorrect.
17	Feature not enabled.
100	This function is not supported on this document processor.

Notes:

1. The rightmost position (start) and the leftmost position (stop) of each zone are specified in inches and tenths of inches relative to document lead edge. Two bytes are used to specify a start or stop point with each byte being a Binary Coded Decimal (BCD) value from 00 to 09.
2. The function allows two zones to be defined. If only one zone is to be defined, it must be zone 1; and the zone 2 definition bytes, *zone2def*, must be all zeros.
3. The minimum zone 1 start position is 0.1 inches.

4. The font specifier, *fontspec*, must be a valid code as defined by the two tables below and must be installed on this document processor.
5. Once defined, the definition for a given OCR reader remains in effect until another Sort run is initialized.
6. A given OCR reader might have up to three numeric fonts installed but only one alphanumeric font installed. If an alphanumeric font is installed, no numeric fonts can be installed for that reader.

OCR NUMERIC FONTS

fontspec	**FONT**
hex 01	OCR-A Size 1, numeric
hex 02	E-13B
hex 03	407-1 or 407-E, whichever is installed
hex 04	7B
hex 05	OCR-B Size 1, numeric
hex 06	1428
hex 08	12F
hex 0D	1403

OCR ALPHANUMERIC FONTS

fontspec	**FONT**
hex 09	Numerics and symbols subset
hex 0A	Alphas and symbols subset (alpha only with OCR-B)
hex 0B	Full character set

Example

```
00  05  00  05  08  01  06  03  01  03  05
```

This buffer specifies that for OCR1:

Zone 1 starts 0.5 inches and stops 1.8 inches from the lead edge.
Zone 1 is to use OCR-B size 1, numeric font.

Zone 2 starts 3.6 inches and stops 5.3 inches from the lead edge.
Zone 2 is to use OCR-A size 1, numeric font.

SpxPutOpCmdLine

Purpose: The SpxPutOpCmdLine function copies data to the operator command line.

Calling Sequence:

```
EXTRN  SpxPutOpCmdLine:FAR

PUSH@  OTHER   buffer            ;65-byte buffer
CALL   SpxPutOpCmdLine
```

buffer

This is a 65-byte data area that contains the data to be copied to the operator command line input area of the document processor. All 65 bytes of data are copied.

The data appears on the command line when control returns to the Run screen. The data is in ASCII format.

Return Codes:

AX	*Description*
0	Successful.

SpxPutOpMsgNum

Purpose: The SpxPutOpMsgNum function stores the Sort operator message number. This does the same function as the SCI "SOM" macro.

Calling Sequence:

```
EXTRN  SpxPutOpMsgNum:FAR

PUSH   BYTE    msgnum          ;Message number
CALL   SpxPutOpMsgNum
```

msgnum

This is a 1-byte message number that is stored in the operator message number area of the document processor. To clear the message (that is, to indicate no message), set the message number to 0.

Return Codes:

AX	*Description*
0	Successful.

Note: The message does not appear until the document processor stops and the Run screen appears.

SpxPutOpMsgTxt

Purpose: The SpxPutOpMsgTxt function copies data to the Sort operator message (SOM) text area.

Calling Sequence:

```
EXTRN  SpxPutOpMsgTxt:FAR

PUSH@  OTHER   text           ;Address of text
PUSH   BYTE    somnum         ;SOM number
CALL   SpxPutOpMsgTxt
```

text

This is an 81-byte data area that contains the message that is copied to the operator message text area of the document processor. The first byte is the severity code and contains one of the following ASCII characters:

I	Information
W	Warning
E	Error.

somnum

This is the SOM number that specifies which message text to replace. The SOM number is in the range 1-255. The data is in ASCII format.

Return Codes:

AX	*Description*
0	Successful.
1	SOM number cannot be zero.
2	Severity level is not I, E, or W.

Notes:

1. This function changes the message text. To make the message appear, you should use the function SpxPutOpMsgNum.
2. Initialization resets the SOM message text number.

SpxPutPcktLim

Purpose: The SpxPutPcktLim function copies data from the pocket definition table (PDT).

If the value that the SpxGetPktTblUsed returns is not zero, then this table is being used to determine when a pocket reaches the merge limit.

Calling Sequence:

```
EXTRN  SpxPutPcktLim:FAR

PUSH@  OTHER   buffer            ;Buffer
CALL   SpxPutPcktLim
```

buffer
This a 72-byte data area that contains 36 entries in 16-bit Intel format.

Return Codes:

AX	*Description*
0	Successful.

Note: If you store a value greater than 4095 for any pocket, no merge documents are routed to that pocket.

SpxPutPktCount

Purpose: The SpxPutPktCount function copies data to the pocket counter area.

You should change bits 1, 2, or 3 only if you want to change the pocket status that these bits indicate. For more information about pocket counters, see the *3890/XP Enhanced Document Processor Programming Guide*.

Calling Sequence:

```
EXTRN  SpxPutPktCount:FAR

PUSH@  OTHER   pktcntr           ;Pocket counters
CALL   SpxPutPktCount
```

pktcntr

This is a 72-byte data area that contains 36 two-byte pocket counters in the SCI format.

Return Codes:

AX	*Description*
0	Successful.

SpxPutPktTblUsed

Purpose: The SpxPutPktTblUsed function sets the pocket limit flag.

Calling Sequence:

```
EXTRN  SpxPutPktTblUsed:FAR

PUSH   BYTE    flag          ;Source byte
CALL   SpxPutPktTblUsed
```

flag

This is a 1-byte data area that contains the value for the pocket limit flag. The following values apply:

hex 00 The initialization data is to determine the pocket limits.
hex 01 The pocket table is to determine the pocket limits.

Return Codes:

AX	*Description*
0	Successful.

SpxPutPrgEndDat

Purpose: The SpxPutPrgEndDat function copies data to the Active Endorsement Area (AEA).

Calling Sequence:

```
EXTRN  SpxPutPrgEndDat:FAR

PUSH@  OTHER   endorsedata       ;72-byte data buffer
CALL   SpxPutPrgEndDat
```

endorsedata
This is a 72-byte data area (3-byte by 24-byte array) that contains the endorsement data.

Return Codes:

AX	*Description*
0	Successful.

Notes:

1. Invalid characters will print as a "?" character. See the *3890/XP Enhanced Document Processor Programming Guide*.
2. If you use the "%" substitution parameter for the endorse date or the "+" for double wide characters, ensure that you did not disable substitution with the SpxPutNoDateIns function or the SpxPutNoDWConv function. After the first document is processed, the "+" ASCII characters are converted to double-wide characters (3890/XP) or to underscore characters (3891/XP and 3892/XP).

SpxPutProcBufDat

Purpose: The SpxPutProcBufDat function copies data to the process buffer.

Calling Sequence:

```
EXTRN  SpxPutProcBufDat:FAR

PUSH@  OTHER   proc            ;Buffer
CALL   SpxPutProcBufDat
```

proc
This is a 244-byte data area that contains data to be copied to the process buffer.

Return Codes:

AX	*Description*
0	Successful.

Notes:

1. This function does not replace the process buffer header.
2. This function sets byte 7, bit 4 of the process buffer header.

SpxPutProcBufHdr

Purpose: The SpxPutProcBufHdr function copies data to the process buffer header.

Calling Sequence:

```
EXTRN  SpxPutProcBufHdr:FAR

PUSH@  OTHER   buffer           ;Buffer
CALL   SpxPutProcBufHdr
```

buffer

This is a 12-byte data area that contains the data to be copied to the process buffer header.

Return Codes:

AX	*Description*
0	Successful.

Note: Although 12 bytes of data are supplied, only a few of those bytes are used to update the process buffer header: header bytes 2, 3, 5, 6, and 8 (if logical merge mode is set), and byte 9.

SpxPutProgStore

Purpose: The SpxPutProgStore function copies data to program storage.

Note: Unlike the SCI instruction that performs the same function, SpxPutProgStore ignores the 3890 emulation parameters that are set in the Run Profile.

Calling Sequence:

```
EXTRN  SpxPutProgStore:FAR

PUSH@  OTHER    buffer          ;Result buffer
PUSH   DWORD    disp            ;Displacement
PUSH   WORD     length          ;Length of move
CALL   SpxPutProgStore
```

buffer
This is the data area from which the data is written.

disp
This is a 32-bit displacement into program storage (Intel format).

length
This is the length, in bytes, to be written.

Return Codes:

AX	*Description*
0	Successful.
1	The displacement value is greater than program storage.
2	The displacement plus length is out-of-bounds.
3	The length is not greater than zero.

Notes:

1. If more than 65 535 bytes of data are to be written, use a series of calls.
2. In most cases, the values in the first 128 bytes of program storage cannot take effect until a new initialization occurs. The only data changes that are effective on succeeding documents with no reinitialization are merge feed control, merge feed pocket limits, and code line data match.

   For additional information, see Note 4 of the notes that follow the “Accessible Storage Map for SCI Load and Store” in Appendix C of the *3890/XP Enhanced Document Processor Programming Guide.*

SpxPutPrtSeq

Purpose: The SpxPutPrtSeq function allows the program to establish the sequence for printing the three endorsement data elements. The default order, from left to right, is:

- Fixed message
- Variable text
- Serial number or INF.

This function is supported on the 3891/XP, 3892/XP, and the UT/XP.

Calling Sequence:

```
EXTRN  SpxPutPrtSeq:FAR

PUSH@  OTHER     buffer       ;Address of the 6-byte data buffer
CALL   SpxPutPrtSeq
```

buffer

This is a 6-byte data area in which each byte defines the print sequence for a given line on the document.

Byte number and corresponding endorsement line:

Byte 1 = Front side line 1
Byte 2 = Front side line 2
Byte 3 = Front side line 3
Byte 4 = Back side line 1
Byte 5 = Back side line 2
Byte 6 = Back side line 3

Hex value definition for each byte:

00 = Variable text, fixed message, serial number
01 = Variable text, serial number, fixed message
02 = Fixed message, variable text, serial number (default value)
03 = Fixed message, serial number, variable text
04 = Serial number, fixed message, variable text
05 = Serial number, variable text, fixed message
20 = Leave the current value unchanged or feature not configured.

Return Codes:

AX	*Description*
0	Successful.
1	The definition value specified is not hex 00 through 05 or hex 20.
4	The Spx was called during a document event.
17	The front side ink jet is not enabled.
18	The back side ink jet is not enabled.
100	This function is not supported on this document processor.

Notes:

1. You should ensure that the variable text does not exceed the permissible maximum of 64 characters per document (including both front and back side printing) on a 3891/XP or 3892/XP or a maximum of 158 characters per document on a UT/XP.
2. This Spx is supported only during non-document events.
3. If an endorsement data element has no data for a given line, a null character is assumed for that element for that line.

SpxPutReinitLoadPath

Purpose:: SpxPutReinitLoadPath sets the re-initialization load path.

Calling Sequence:

```
EXTRN  SpxPutReinitLoadPath:FAR

PUSH@  ASCIIZ  ReinitLoadPath
CALL   SpxPutReinitLoadPath
```

ReinitLoadPath

is the directory path to be used to load the run profile and associated files during re-initialization. The path can be up to 65 characters plus the null terminator (for a maximum length of 66 bytes). ReinitLoadPath must be a valid OS/2 drive-path specification.

Return Codes:

AX	Description
0	Successful.
1	Null terminator missing. No action taken.

Notes:

1. Use **SpxPutReinitLoadPath** to change the path that will be used to load the run profile and associated files during re-initialization.
2. Use **SpxPutReinitReq** to actually request re-initialization during the SPSINIT event.

SpxPutReinitReq

Purpose: SpxPutReinitReq sets the re-initialization request flag.

Calling Sequence:

```
EXTRN  SpxPutReinitReq:FAR

PUSH   BYTE    ReinitReq
CALL   SpxPutReinitReq
```

ReinitReq

is a one-byte flag that requests re-initialization. Possible values are:

0 Re-initialization is not requested

1 Re-initialization is requested

Return Codes:

AX	Description
0	Successful.
1	Re-initialization in progress. No action taken.
2	Not initialization (SPSINIT) event. No action taken.

Notes:

1. Use **SpxPutReinitReq** to request re-initialization during the SPSINIT event.
2. Re-initialization includes processing of the run profile and the run profile associated files. Re-initialization does not include processing of the initialization data (IREC), except for the name of the run profile.
3. Use **SpxPutProgStore** to change the run profile (name is in the IREC) that will be used during re-initialization.
4. Use **SpxPutReinitLoadPath** to change the path that will be used to load the run profile and associated files during re-initialization.

SpxPutRqNotInit

Purpose: The SpxPutRqNotInit function sets or clears the request-not-initialized flag.

This flag requests the Control Program to fail initialization for this run. The failure code will be hex 1F. Message 5031 will be displayed on the Run screen.

Calling Sequence:

```
EXTRN  SpxPutRqNotInit:FAR

PUSH   BYTE     flag          ;Value
CALL   SpxPutRqNotInit
```

flag

This is a 1-byte data area that contains the value of the flag. The following values apply:

hex 00 The flag is not set.
hex 01 The flag is set.

Return Codes:

AX	*Description*
0	Successful.

SpxPutSavArea

Purpose: The SpxPutSavArea function copies up to 256 bytes of data to the Save Area.

Calling Sequence:

```
EXTRN  SpxPutSavArea:FAR

PUSH@  OTHER   area           ;Save area
PUSH   WORD    displacement   ;Displacement
PUSH   WORD    length         ;Transfer length
CALL   SpxPutSavArea
```

area
This is the data area from which the data is written.

disp
This is the displacement within the save area of the document processor where the data is stored.

length
This is the number of bytes to be written.

Return Codes:

AX	*Description*
0	Successful.
1	The displacement was out-of-bounds (greater than 255).
2	The displacement plus length was out-of-range (greater than 256).
3	The length was zero.

SpxPutSCICount

Purpose: The SpxPutSCICount function copies data to the SCI counters area.

Calling Sequence:

```
EXTRN  SpxPutSCICount:FAR

PUSH@  OTHER  counter            ;Address of counters
CALL   SpxPutSCICount
```

counter

This is a 32-byte data area that the calling program supplies. These are the 16 counters, which are 2-bytes each.

Return Codes:

AX	*Description*
0	Successful.

SpxPutSEC

Purpose: The SpxPutSEC function copies data to the symbol error correction value in emulated auxiliary storage.

Calling Sequence:

```
EXTRN  SpxPutSEC:FAR

PUSH   BYTE   secbyte           ;SEC byte
CALL   SpxPutSEC
```

secbyte

This is the 1-byte data area that contains the symbol error correction value that this function copies to the emulated auxiliary storage. Any change to this value takes effect on the next document cycle. This is the same as initialization data byte 55.

Return Codes:

AX	*Description*
0	Successful.

SpxPutUsrStat

Purpose: The SpxPutUsrStat function copies a value to the user statistics area.

Calling Sequence:

```
EXTRN  SpxPutUsrStat:FAR

PUSH   BYTE   buffer            ;Source buffer
CALL   SpxPutUsrStat
```

buffer

This is a 1-byte data area that contains the value that this function copies to the user statistics area (user stats).

Return Codes:

AX	*Description*
0	Successful.

SpxPutWrkReg

Purpose: The SpxPutWrkReg function copies data to the work register. SCI programs use the work register as temporary storage to manipulate data.

Calling Sequence:

```
EXTRN  SpxPutWrkReg:FAR

PUSH@  OTHER   buffer            ;16-byte buffer
CALL   SpxPutWrkReg
```

buffer
This is a 16-byte data area that the calling program supplies.

Return Codes:

AX	*Description*
0	Successful.

SpxSinglePick

Purpose: The SpxSinglePick function requests the transport to feed just one document. This function is only supported on the 3891/XP and the 3892/XP.

SpxSinglePick sets a flag so the Control Program requests just one document from the transport. The call should be made at SPSINIT-time. Merge before main (MBM) is ignored, so that the single pick document comes from the main feeder.

If running offline from the host, the motors are stopped and control returns to the Run screen. If this function is called again at SPSDOC-time, the single pick flag is set for the next feed, so each press of the Feed key feeds just one document. Therefore, the user can single step through a set of documents and look at Sort program messages for each one, perhaps looking for a document that causes an error.

If running online to the host, the Control Program sets the Sort Pause Requested flag (header byte 9, bit 7). When the machine pauses, documents in the process buffer are flushed to the host and the Control Program then waits for "Host OK to Start" before requesting the next document. The host might download different initialization data at this point and therefore run a different job defined by special "beginning of job" documents.

The user Sort module can resume single pick, such as when another special control document is seen, making the call to SpxSinglePick at SPSDOC-time. To allow for the fact that the machine does not pause immediately, there should be a number of these control documents. Up to 12 more documents are seen before the transport pauses or stops because of "documents in flight." These "end of job" documents can be followed by another "beginning of job" special document. The effect is that many different sort jobs can be run as a single operation at the machine.

Calling Sequence:

```
EXTRN  SpxSinglePick:FAR

CALL   SpxSinglePick
```

Return Codes:

AX	*Description*
0	Successful.
100	This function is not supported on this document processor.

SpxSPC

Purpose: The SpxSPC function sets the module-pocket in the process buffer header (byte 6) to the module-pocket (MP) code of the first (lowest-numbered) pocket that has reached or exceeded a predetermined value (set during initialization). This is the same as the SCI "SPC" macro.

The operation scans from 1/1 to 6/6. If no counter has reached the predetermined value, the module-pocket code is set to 1/1 (reject).

Calling Sequence:

```
EXTRN  SpxSPC:FAR

CALL   SpxSPC
```

Return Codes:

AX	*Description*
0	MP set for full counter
1	MP set for 1/1 (all counters low).

Note: If you use the SpxPutProcBufHdr function after using this function, you can overwrite and cancel this function.

Chapter 3. Alphabetic Reference (Spx32 Functions)

This chapter provides an alphabetic reference of all the Spx32 functions. These functions are available **ONLY** with the 3890/XP Enhanced control program. For each function, it describes:

- Purpose
- Calling sequence
- Return codes.

Spx32AbSrtProg

Purpose: The Spx32AbSrtProg function prematurely ends the Sort Program Sequence (SPS). No further modules for the current SPS event are called and additional SPS events for the current document are ignored.

The process buffer reflects the level of processing that was completed when the abort flag was set. Byte 7, bit 1 in the header is set on to indicate the Cancel operation; the document is sent to module-pocket 1/1. The feature control is handled as though the document had been autoselected.

Calling Sequence:

```
short _System Spx32AbSrtProg(void);

rc = Spx32AbSrtProg();
```

Return Codes:

rc	*Description*
0	Successful.

Spx32DSG

Purpose: The Spx32DSG function requests the disengage function.

This function does the same thing as the Stacker Control Instruction (SCI) language disengage (DSG) macro. It sets a flag that is acted on during post-sort processing of the current document. Document feeding from the main hopper stops immediately. Merge documents feed until all merge requests are acted on. All documents in the document path are processed normally before the transport stops.

Note: There can be up to 12 documents in the document path.

Calling Sequence:

```
short _System Spx32DSG(void);

rc = Spx32DSG();
```

Return Codes:

rc	*Description*
0	Successful.

Spx32DSGA

Purpose: The Spx32DSGA function requests the disengage-and-set-operator-attention function.

This function does the same thing as the SCI language disengage-and-set-operator-attention (DSGA) macro. Spx32DSGA sets a flag that is acted on during post-processing of the current document. Document feeding from the main hopper stops immediately. Merge documents feed until all merge requests are acted on. All documents in the document path are processed normally before the transport stops.

Note: There can be up to 12 documents in the document path.

Calling Sequence:

```
short _System Spx32DSGA(void);

rc = Spx32DSGA();
```

Return Codes:

rc	*Description*
0	Successful.

Spx32EIMAT

Purpose: The Spx32EIMAT function writes a 16-bit data word from the calling program into the control field for extended code line data match. This overrides the definition given in the initialization data.

Calling Sequence:

```
short _System Spx32EIMAT(unsigned short *);
unsigned short Buffer;

rc = Spx32EIMAT(&Buffer);
```

buffer
This is a data area that specifies the fields to be matched.

Return Codes:

rc	*Description*
0	Successful. All requested fields are valid.
1	A specified field was not previously defined for code line data match by the code line definition table or the initialization data. It will now be used.

Bit definitions:

0	Field 1	**8**	Field 9
1	Field 2	**9**	Field 10
2	Field 3	**10**	Field 11
3	Field 4	**11**	Field 12
4	Field 5	**12**	Field 13
5	Field 6	**13**	Field 14
6	Field 7	**14**	Field 15
7	Field 8	**15**	Reserved

Note: To use extended code line data matching, first call Spx32EIMAT; then call SCII32 (supplied by IBM in SCIEM) from your program. You must make this call during the SPSDOC event.

To call SCII32, code the following in the source file:

```
void _System SCII32(void);

SCII32();
```

then code the following in the link edit:

```
IMPORTS    SCIEM.SCII32
```

Spx32EndPM

Purpose: The Spx32EndPM function tells the control program that the PM call is exiting. This is used by the control program to track usage of user interface calls.

Calling Sequence:

```
short _System Spx32EndPM(unsigned short);
short PMCounter;

rc = Spx32EndPM(PMCounter);
```

PMCounter
This is a data area that contains the number of user dialogs active under PM. A maximum of 5 user dialogs may be active at any one time.

Return Codes:

rc	*Description*
0	Successful
1	Invalid PMCounter
2	Nested PM Procedure(s) Active

Spx32FixM

Purpose: The Spx32FixM function is called at SPSDOC-time to specify fixed message endorsement(s) for the current document. See Spx32LoadFixM.
This function is supported on the 3891/XP, 3892/XP and Universal Transport (UT)/XP**

Calling Sequence:

```
short _System Spx32FixM(void *);
unsigned char DataArea[6];

rc = Spx32FixM(DataArea);
```

buffer

A 6-byte data area that specifies six fixed message numbers, front side lines 1, 2, and 3 and back side lines 1, 2, and 3. A zero specifies no message for that side/line.

If the font size is defined as large, only one line can be printed on that side. It must be specified as line 1.

Return Code:

rc	*Description*
0	Successful
1	Invalid message number(s)
100	This function is not supported on this document processor.

Notes:

1. The same fixed message can be given for more than one side/line.
2. If you direct a fixed message to the same side/line that has a variable text message (via CONVERGE), the error will cause the message 6030 Transport Option State Error.
3. If you specify a large font size and put a message to other than line 1 for that side, the error will cause the message 6030 Transport Option State Error.
4. If you specify a fixed message that was not loaded, the return code reflects the error. The machine does not stop.

** The UT/XP is trademarked by Recognition International Inc as *Recognition UT 600/1000 XP*.

Spx32FM

Purpose: The Spx32FM function requests a document from the merge feeder. This function does the same thing as the SCI feed merge (FM) macro.

Calling Sequence:

```
short _System Spx32FM(void);

rc = Spx32FM();
```

Return Codes:

rc	*Description*
0	Successful
1	Request ignored; can only be requested once during document-time.

Spx32Get92Features

Purpose: The Spx32Get92Features function returns the features installed on the 3891/XP and 3892/XP.

Calling Sequence:

```
short _System Spx32Get92Features(unsigned short *);
unsigned short Features;

rc = Spx32Get92Features(&Features);
```

feature

This is a 2-byte data area in which this function returns the features installed status. The following names apply:

Bit	*Feature*
0	MICR-1 Reader
1	Hi-Line Reader/MICR2
2	OCR-1 Reader (OCR RPQ)
3	OCR-2 Reader (OCR RPQ)
4	IBM OCR-3 Reader
5	Front Ink Jet Printer
6	Back Ink Jet Printer
7	Power Encoder
8	Image Scanner
9	Microfilmer
10	Roll On Endorser
11	Merge Hopper
12	Reserved.
13	Reserved.
14	Reserved.
15	Reserved.

Return Codes:

rc	*Description*
0	Successful.
100	This Spx32 is not available on a 3890/XP

Note: This Spx32 returns the 3891/XP or 3892/XP feature installed list and is not applicable on the 3890/XP. For a complete feature installed list on the 3890/XP, see “Spx32GetFeatures” on page 3-23.

Spx32Get92InitList

Purpose: The Spx32Get92InitList function returns the list of initialized features for the 3891/XP and 3892/XP.

Calling Sequence:

```
short _System Spx32Get92InitList(unsigned short *);
unsigned short InitList;

rc = Spx32Get92InitList(&InitList);
```

featlist

This is a 2-byte data area in which this function returns the initialized feature data. The following values apply:

Bit	*Feature*
0	MICR-1 Reader
1	Hi-Line Reader/MICR2
2	OCR-1 Reader (OCR RPQ)
3	OCR-2 Reader (OCR RPQ)
4	IBM OCR-3 Reader
5	Front Ink Jet Printer
6	Back Ink Jet Printer
7	Power Encoder
8	Image Scanner
9	Microfilmer
10	Roll On Endorser
11	Merge Hopper
12	Reserved.
13	Reserved.
14	Reserved.
15	Reserved.

Return Codes:

rc	*Description*
0	Successful.
100	This Spx32 is not available on a 3890/XP

Note: This Spx32 returns the 3891/XP or 3892/XP feature list and is not applicable on the 3890/XP. For a complete feature list on the 3890/XP see, “Spx32GetInitList” on page 3-31.

Spx32GetAbortSrt

Purpose: The Spx32GetAbortSrt function returns the status of the abort-sort-program flag. See Spx32AbSrtProg.

Calling Sequence:

```
short _System Spx32GetAbortSrt(unsigned char *);
unsigned char AbSortFlag;

rc = Spx32GetAbortSrt(&AbSortFlag);
```

flag

This is a 1-byte data area to receive the abort-sort-program flag. The following values apply:

hex 00 The flag is not on.
hex 01 The flag is on.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetActEndDate

Purpose: The Spx32GetActEndDate function returns the active endorse date.

Calling Sequence:

```
short _System Spx32GetActEndDate(char *);
char EndDate[8];

rc = Spx32GetActEndDate(EndDate);
```

date

This is an 8-byte data area in which this function returns the active endorse date. Format is ASCII.

Return Codes:

rc	*Description*
0	Successful.

Notes:

1. This date is set at run initialization time.
2. The date information can come from the last-used date, if no date is specified in the date field (bytes 88-95) in the initialization data.
3. If the PEDProtect keyword of the Run Profile is not active, the date can be changed by the operator from the Endorse Setup/Verify screen.
4. If characters in the endorse date field are not valid, question marks will print.

Spx32GetAdditnlMsg

Purpose: The Spx32GetAdditnlMsg function returns message text from the additional sort message area. See Spx32PutAdditnlMsg.

Calling Sequence:

```
short _System Spx32GetAdditnlMsg(char *);
char AdditnlMsg[40];

rc = Spx32GetAdditnlMsg(AdditnlMsg);
```

buffer

This is a 40-byte data area in which this function returns the additional sort message. This function always stores 40 bytes. If you specify a data area that is smaller than 40 bytes, you might cause a *protection violation* or lose other data.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetAddMsglong

Purpose: The Spx32GetAddMsglong function returns the message data from the additional sort message area.

Calling Sequence:

```
short _System Spx32GetAddMsglong(char *);
char MsgLong[80];

rc = Spx32GetAddMsglong(MsgLong);
```

buffer

This is an 80-byte data area in which the function stores the data copied from the additional sort message area of the document processor.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetAltINFArea

Purpose: The Spx32GetAltINFArea function returns the alternate INF flag and the alternate item-number-feature (INF) data.

The alternate INF number is the user's responsibility. It must be in an unsigned, packed-decimal format, using all 10 digits. Blanks are specified as X'A'. A question mark (?) indicates that characters in the output are not valid. The invalid-INF-data bit is set on in the process buffer header to indicate that the
INF data is not valid and that it is available to the next document for action.

Calling Sequence:

```
short _System Spx32GetAltINFArea(unsigned char *);
unsigned char AltINF[6];

rc = Spx32GetAltINFArea(AltINF);
```

buffer

This is a 6-byte data area in which this function returns the alternate INF data and the alternate INF flag. The first byte is the flag; the remaining 5 bytes (10 digits) are the data.

The following flag values apply:

00 Alternate data is not used.
01 Alternate INF data is used.

Return Codes:

rc	*Description*
0	Successful.

Note: Initialization data (byte 82, bit 2) can inhibit this function.

Spx32GetAutoSel

Purpose: The Spx32GetAutoSel function returns the autoselect bits from the current document in the document processor or from the previous document during a motors stop exit.

Calling Sequence:

```
short _System Spx32GetAutoSel(unsigned char *);
unsigned char AutoSel;

rc = Spx32GetAutoSel(&AutoSel);
```

autosel

This is a 1-byte data area in which this function returns the autoselect bits. The following values apply:

BIT	*Reason*
0	Multiple documents
1	Short gap
2	Long document
3	Short document
4	Short lead-edge to lead-edge (LE-LE)
5	Special symbol sequence error
6	Reserved
7	Reserved.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetCDT

Purpose: The Spx32GetCDT function returns the code line definition table from mapped auxiliary storage.

Calling Sequence:

```
short _System Spx32GetCDT(void *);
unsigned char CDTBuffer[149];

rc = Spx32GetCDT(CDTBuffer);
```

cdtbuff

This is a 149-byte data area in which this function stores the code line definition table. The format is defined in Appendix C of the *3890/XP Enhanced Document Processor Programming Guide*.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetCtrlNum

Purpose: The Spx32GetCtrlNum function returns the document processor Control Program release version. This appears on the “logo” screen and on the Main Menu.

Calling Sequence:

```
short _System Spx32GetCtrlNum(char *);
char CtrlNum[6];

rc = Spx32GetCtrlNum(CtrlNum);
```

vernum

This is a 6-character data area in which this function stores the Control Program version number.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetDisengage

Purpose: The Spx32GetDisengage function returns the status of the disengage flag. See Spx32DSG.

Calling Sequence:

```
short _System Spx32GetDisengage(unsigned char *);
unsigned char Disengage;

rc = Spx32GetDisengage(&Disengage);
```

flag

This is a 1-byte data area in which this function returns the status of the disengage flag. The following values apply:

00 Disengage is not requested.

01 Disengage was requested but is not yet acted on. It is acted on, where requested, after sort processing of the document. When the request is acted on, the flag is reset to X'00'.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetDocsToPause

Purpose: The Spx32GetDocsToPause function gets a value in number of documents to process before a pause will be requested to test syncronization between the sorter and the Image Capture Processor.

This Spx is to be used in conjunction with the PAUSEXP.DLL supplied on the control program diskette.

Calling Sequence:

```
short _System Spx32GetDocsToPause(void *  );
unsigned long buffer;

rc = Spx32GetDocsToPause(&buffer);
```

buffer

This is an unsigned long data area where the current document to pause count will be placed.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetEncStatus

Purpose: The Spx32GetEncStatus function returns the value of the power encoder status bytes. This function is supported on the 3892/XP.

Calling Sequence:

```
short _System Spx32GetEncStatus(unsigned char *);
unsigned char buffer[3];

rc = Spx32GetEncStatus(buffer);
```

buffer

This is a 3-byte data area that will contain the power encoder status bytes.

Byte 1, general encoder status
- bit 7 = Verify disabled
- bit 6 = Encoder selected
- bit 5 = Reserved
- bit 4 = Jam detected
- bit 3 = Interlock open
- bit 2 = Reserved
- bit 1 = Verification threshold reached
- bit 0 = Data format error

Byte 2, path A status
- bit 7 = Ribbon low
- bit 6 = Ribbon not ready
- bit 5 = Ribbon not present
- bit 4 = Ribbon break or jam, capstan fault
- bit 3 = Hammer sync error
- bit 2 = Hammer hardware fault
- bit 1 = Hammer voltage fault
- bit 0 = Path disabled

Byte 3, path B status

Same as byte 2, but for path B

Return Codes:

rc	*Description*
0	Successful.
4	The function was issued while actively processing documents. No action was taken.
17	The feature is not enabled.
100	This function is not supported on this document processor.

Spx32GetEncVStat

Purpose: The Spx32GetEncVStat function returns the value of the power encoder verify status bytes. This function is supported on the 3892/XP.

Calling Sequence:

```
short _System Spx32GetEncVStat(unsigned char *);
unsigned char buffer[4];

rc = Spx32GetEncVStat(buffer);
```

buffer

This is a 4-byte data area that will contain the power encoder verify status bytes.

Byte 1, verification reader status
- bit 7 = Reserved
- bit 6 = Reserved
- bit 5 = Reserved
- bit 4 = Reserved
- bit 3 = Verification disabled
- bit 2 = Reserved
- bit 1 = Reserved
- bit 0 = Reader sync error

Byte 2, path A status
- bit 7 = Reserved
- bit 6 = Cumulative substitution error
- bit 5 = Cumulative reject error
- bit 4 = Consecutive substitution error
- bit 3 = Consecutive reject error
- bit 2 = Substitution document dispositioned
- bit 1 = Reject document dispositioned
- bit 0 = Path disabled

Byte 3, path B status
- bit 7 = Reserved
- bit 6 = Cumulative substitution error
- bit 5 = Cumulative reject error
- bit 4 = Consecutive substitution error
- bit 3 = Consecutive reject error
- bit 2 = Substitution document dispositioned
- bit 1 = Reject document dispositioned
- bit 0 = Path disabled

Byte 4, count of saved verification error documents, 0 to 16

Return Codes:

rc	*Description*
0	Successful.
4	The function was issued while actively processing documents. No action was taken.
17	The feature is not enabled.
100	This function is not supported on this document processor.

Spx32GetErd

Purpose:: The Spx32GetErd function returns data from the External Reader Data buffer. It is supported on the 3891/XP and 3892/XP.

Calling Sequence:

```
short _System Spx32GetErd(void *);
unsigned short buffer[256];

rc = Spx32GetErd(buffer);
```

buffer

This is a 256-byte user data area in which this function returns the ERD data. The format is:

Byte	*Data Description*
0	Length of data
1	Data block ID, Hex B0
2-7	Reserved
8-255	Reader data for selected reader(s), in the form,

rdrlngth 1-byte length of the rdrdata that follows,

rdrdata Reader ID byte and document data from that reader

Reader ID bytes (in hex) are document data from that reader

E0 MICR-1
F0 OCR1 (RPQ)
F8 OCR2 (RPQ)
D8 Hi-Line/MICR2

Return Codes:

rc	**Description**
0	Successful
100	Function not supported by this document processor

Note: Data is valid only during one of the document time SPS events. There will be reader data for one or more readers.

Spx32GetFeatures

Purpose: The Spx32GetFeatures function returns the features-installed byte.

Calling Sequence:

```
short _Spx32GetFeatures(unsigned char *);
unsigned short feature;

rc = Spx32GetFeatures(&feature);
```

feature

This is a 1-byte data area in which this function returns the features-installed status. The following names apply:

BIT	*Feature*
0	INF
1	Endorse
2	Image Scanner
3	Microfilm
4	IBM OCR (OCR-3)
5-7	Reserved.

Return Codes:

rc	*Description*
0	Successful.

Note: This Spx32 returns the 3890/XP feature list and is applicable on all XP series machines. For a complete feature list on the 3891/XP and 3892/XP, see "Spx32Get92Features" on page 3-8.

Spx32GetHAB

Purpose: The Spx32GetHAB function returns the Anchor Block Handle for the current process.

Calling Sequence:

```
short _System Spx32GetHAB(unsigned short,unsigned long *);
unsigned short PMCounter;
unsigned long PResults;

rc = Spx32GetHAB(PMCounter,&PResults);
```

PMCounter
This is a data area that contains the number of user dialogs active under PM. A maximum of 5 user dialogs may be active at any one time.

PResults
This is a data area in which this function returns the Anchor Block Handle for the current process.

Return Codes:

rc	*Description*
0	Successful
1	Invalid PMCounter

Spx32GetHMQ

Purpose: The Spx32GetHMQ function returns the Message Queue Handle for the current process.

Calling Sequence:

```
short _System Spx32GetHMQ(unsigned short,unsigned long *);
unsigned short PMCounter;
unsigned long PResults;

rc = Spx32GetHMQ(PMCounter,&PResults);
```

PMCounter
This is a data area that contains the number of user dialogs active under PM. A maximum of 5 user dialogs may be active at any one time.

PResults
This is a data area in which this function returns the Message Queue Handle for the current process.

Return Codes:

rc	*Description*
0	Successful
1	Invalid PMCounter

Spx32GetHOZNum

Purpose: The Spx32GetHOZNum function returns the high-order-zero (HOZ) correction control data (see byte 60 of the IREC).

Calling Sequence:

```
short _System Spx32GetHOZNum(unsigned char *);
unsigned char hozbyte;

rc = Spx32GetHOZNum(&hozbyte);
```

hozbyte

This is a 1-byte data area in which this function returns the HOZ correction control number.

Although one byte of data is returned, only the low-order 4 bits of data have significance. The high-order 4 bits are zeros. This is a binary integer that ranges from 0 to 15.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetHwnd

Purpose: The Spx32GetHwnd function returns the window handle of the run screen.

Calling Sequence:

```
short _System Spx32GetHwnd(unsigned long *);
unsigned long buffer;

rc = Spx32GetHwnd(&buffer);
```

buffer
This is a data area in which this function returns the window handle of the run screen.

Return Codes:

rc	*Description*
0	Successful

Notes:

1. Using this function enables invocation of WinDlgBox to display a user dialog.
2. The user dialog code will execute in the thread that called it.

Spx32GetImgBufPntr

Purpose: The Spx32GetImgBufPntr function returns the 12 code line data match buffer pointers.

Calling Sequence:

```
short _System Spx32GetImgBufPntr(void *);
void pointer[12];

rc = Spx32GetImgBufPntr(pointer);
```

pointer

This is a 48-byte data area in which this function returns the code line data match buffer pointers.

The following are the first six code line data match buffer pointers:

- Low limit
- High limit
- Search center
- Starter
- Wrap pointer
- Wrap reset.

When the extended code line data match is active, the next six pointers are copies of these first six pointers. SCII saves the first six pointers here and increments the first four pointers for a *not found* condition when the extended code line data match is active. Restoration is made if an extended code line data match call is made.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetINFNum

Purpose: The Spx32GetINFNum function returns the INF number.

Calling Sequence:

```
short _System Spx32GetINFNum(unsigned char *);
unsigned char buffer[5];

rc = Spx32GetINFNum(buffer);
```

buffer

This is a 5-byte data area in which this function returns the INF number.

This function returns the INF number as an unsigned, packed-decimal number. For 3890 emulation, this 10-digit field is left-aligned; the eight leftmost (high-order) digits are the INF number; the two rightmost (low-order) digits are blanks (X'AA'). In the XP mode, all 10 digits can be the INF number, as the initialization data (IREC) specifies.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetInitFtr

Purpose: The Spx32GetInitFtr function returns the initialize-feature-data (IFD) flag. See Spx32IFD.

Calling Sequence:

```
short _System Spx32GetInitFtr(unsigned char *  );
unsigned short flag;

rc = Spx32GetInitFtr(&flag);
```

flag

This is a 1-byte data area in which this function returns the IFD flag. The following values apply:

hex 00 The flag is not set; IFD is not requested.
hex 01 The flag is set; IFD is requested.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetInitList

Purpose: The Spx32GetInitList function returns the list of initialized features.

Calling Sequence:

```
short _System Spx32GetInitList(unsigned char *  );
unsigned char featlist;

rc = Spx32GetInitList(&featlist);
```

featlist

This is a 1-byte data area in which this function returns the initialized feature data. The following values apply:

Bit	*Feature*
0	INF
1	Endorse
2	Image Scanner
3	Microfilm
4	IBM OCR (OCR-3)
5-7	Reserved.

Return Codes:

rc	*Description*
0	Successful.

Note: This Spx32 returns the 3890/XP feature list and is applicable on all XP series machines. For a complete feature list on the 3891/XP and 3892/XP, see “Spx32Get92InitList” on page 3-9.

Spx32GetInptBufDat

Purpose: The Spx32GetInptBufDat function returns data from the input buffer.

Calling Sequence:

```
short _System Spx32GetInptBufDat(void *  );
unsigned char buffer[296];

rc = Spx32GetInptBufDat(buffer);
```

buffer

This is a 296-byte data area in which this function returns the input buffer data. During sort processing, this is the data for the current document. When the motors are stopped, this is the data for the last document processed. The format is defined in Appendix C of the *3890/XP Enhanced Document Processor Programming Guide*.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetISFFeatCtl

Purpose: The Spx32GetISFFeatCtl function returns the Image Scanner feature control byte for the Image Capture System.

Calling Sequence:

```
short _System Spx32GetISFFeatCtl(unsigned char *  );
unsigned char buffer;

rc = Spx32GetISFFeatCtl(&buffer);
```

buffer

A 1-byte buffer to receive the ISF control byte. The byte is defined by initialization data (IREC), byte 110.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetJamData

Purpose: The Spx32GetJamData function returns jam data.

Calling Sequence:

```
short _System Spx32GetJamData(void *  );
unsigned char jam[12];

rc = Spx32GetJamData(jam);
```

jam

This is a 12-byte data area in which this function returns the exception record header that has the jam data. The format is described in the *3890/XP Enhanced Document Processor Programming Guide*.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetJamDetail

Purpose: The Spx32GetJamDetail function returns document data for the last jam. It is supported on the 3891/XP and the 3892/XP.

Calling Sequence:

```
short _System Spx32GetJamDetail(void *  );
unsigned char buffer[80];

rc = Spx32GetJamDetail(buffer);
```

buffer

This is an 80-byte data area that receives the document data. The format is as follows:

10 bytes:	Current document serial number. This consists of ten ASCII characters and is the last serial number sent to the Transport for INF, Alt INF, or microfilm.
10 bytes:	Reserved
10 bytes:	Reserved
10 bytes:	Reserved
10 bytes:	Serial number of the last document at the microfilm camera. If the device is not selected, or if no documents have reached the device since the last feed start, the response will be binary zeros.
4 bytes:	Reserved
6 bytes:	Reserved
10 bytes:	Reserved
10 bytes:	Reserved

Return Codes:

rc	*Description*
0	Successful.
4	Document-time SPS event, no action taken.
100	Function not supported by this document processor.

Note: This function can be called during the Motor Stop SPS event. Because the data returned pertains to the last jam, the user sort program should first call Spx32GetJamInd to determine whether a jam has occurred.

Spx32GetJamInd

Purpose: The Spx32GetJamInd function returns the state of the jam indicator flag.

This flag is set to indicate that this is the first document after a jam. The flag is cleared as soon as this document is processed. If a jam caused the motors to stop, the flag is also on during the processing of the "motors stopping" SPS function. You can use the Spx32GetJamData function to retrieve the exception record header. On the 3891/XP and 3892/XP, you can also use the Spx32GetJamDetail to retrieve document data.

Calling Sequence:

```
short _System Spx32GetJamInd(unsigned char *  );
unsigned char flag;

rc = Spx32GetJamInd(&flag);
```

flag

This is a 1-byte data area in which this function returns the status of the jam indicator flag. The following values apply:

hex 00 The flag is not set; jam has not occurred.
hex 01 The flag is set; jam has occurred.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetLCLTable

Purpose: The Spx32GetLCLTable function returns an element from the level control label (LCL) table.

Calling Sequence:

```
short _System Spx32GetLCLTable(void *, unsigned short );
unsigned char lcl[80];
unsigned short element;

rc = Spx32GetLCLTable(lcl,element);
```

lcl

This is an 80-byte data area in which this function stores an element of the LCL table.

element

This is a 16-bit word value that specifies which LCL record to return. For example, a value of 1 returns the first 80-byte record.

To receive the complete LCL table, use a loop and increment the value of the element until the return code is 1. The following pseudo code is an example of this type of loop:

```
repeat
  rc = Spx32GetLCLTable(lcl,element);
until rc = 1;
```

The format is:

16 bytes: user label
8 bytes: file name
4 bytes: file extension (includes period)
52 bytes: file path.

See the LCL Run Profile keyword in the *3890/XP Enhanced Document Processor Programming Guide*.

Return Codes:

rc	***Description***
0	Successful
1	Not a valid element value.

Spx32GetMicr2Data

Purpose: The Spx32GetMicr2Data function returns the High Read feature (MICR2) input data. This function is supported on the 3892/XP.

Calling Sequence:

```
short _System Spx32GetMicr2Data(void *  );
unsigned char buffer[266];

rc = Spx32GetMicr2Data(buffer);
```

buffer

This is a 266-byte data area within the caller's memory space where the MICR2 input data is returned.

The format is:

2 bytes:	hex 20 00 (valid data status)
2 bytes:	length of data
260 bytes:	document data, right justified
2 bytes:	hex 20 20 (ending pads).

Return Codes:

rc	*Description*
0	Successful.
12	Non-document SPS event, no action taken.
13	The requested feature is not installed.
16	Data not valid.
17	Feature not enabled.
100	This function is not supported on this document processor.

Spx32GetModel

Purpose: The Spx32GetModel function returns the XP Series document processor model type.

Calling Sequence:

```
short _System Spx32GetModel(unsigned char *  );
unsigned char model;

rc = Spx32GetModel(&model);
```

model

This is a 1-byte data area in which this function returns the XP Series document processor model type. The following values apply:

Byte	*Document Processor*
1	3890/XP
2	3892/XP
3	3891/XP.
4	UT/XP.
5	3890/XP Enhanced

Return Codes:

rc	*Description*
0	Successful.
4	Document-time SPS event, no action taken.

Spx32GetMP

Purpose: The Spx32GetMP function returns the module destination and the pocket destination for the current document.

Calling Sequence:

```
short _System Spx32GetMP(unsigned short *  );
unsigned short buffer;

rc = Spx32GetMP(&buffer);
```

buffer

This is a 2-byte data area in which this function returns the module destination and the pocket destination. The following values apply:

Byte	*Destination*
1	Module, 1 through 6
2	Pocket, 1 through 6.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetNatErrHdr

Purpose: The Spx32GetNatErrHdr function returns the Native Error header.

Calling Sequence:

```
short _System Spx32GetNatErrHdr(void *  );
unsigned char buffer[12];

rc = Spx32GetNatErrHdr(buffer);
```

buffer

This is a 12-byte data area in which this function stores the native error header. See "Document Data Record Header" in the *3890/XP Enhanced Document Processor Programming Guide*.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetNatModRC

Purpose: The Spx32GetNatModRC function returns the OS/2 language (“native”) module return code.

Calling Sequence:

```
short _System Spx32GetNatModRC(unsigned short *  );
unsigned short retcode;

rc = Spx32GetNatModRC(&retcode);
```

retcode

This is a 2-byte data area in which this function returns the current OS/2 module return code. Subsequent SPS event entry point modules can use this return code to determine how the previous module completed. The return code in AX is saved by the Control Program each time an SPS event module returns.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetNewArea

Purpose: The Spx32GetNewArea function returns up to 4K bytes of data from the New Save Area in additional AUX storage.

Calling Sequence:

```
short _System Spx32GetNewArea(void *  ,unsigned long,unsigned long);
unsigned char buffer[300];
unsigned long displacement;
unsigned long length;

rc = Spx32GetNewArea(buffer,displacement,length);
```

buffer
Is a data area to which data is copied.

displacement
This is the displacement within the extended save area of the first byte that is to be copied. For the displacement value to begin at the first byte, you should specify zero.

length
This is the number of bytes to be copied.

Return Codes:

rc	***Description***
0	Successful.
1	Displacement out-of-bounds (greater than 4095).
2	Displacement plus length out-of-range (greater than 4096).
3	Length is zero.

Spx32GetNoDateIns

Purpose: The Spx32GetNoDateIns function returns the status of the date-substitution-in-endorse-data flag.

Calling Sequence:

```
short _System Spx32GetNoDateIns(unsigned char *  );
unsigned char flag;

rc = Spx32GetNoDateIns(&flag);
```

flag

This is a 1-byte area in which this function returns the status of the date-substitution-in-endorse-data flag. The following values apply:

hex 00 No suppression; date substitution allowed.
hex 01 Suppressed; date substitution not allowed.

Return Codes:

rc	*Description*
0	Successful.

Note: The flag is reset at the next initialization.

Spx32GetNoDWConv

Purpose: The Spx32GetNoDWConv function returns the status of the double-wide-endorse-data-conversion flag.

Calling Sequence:

```
short _System Spx32GetNoDWConv(unsigned char *  );
unsigned char flag;

rc = Spx32GetNoDWConv(&flag);
```

flag

This is a 1-byte data area in which this function returns the status of the double-wide-endorse-data-conversion flag. The following values apply:

hex 00 No suppression; normal conversion allowed.
hex 01 Suppressed; conversion not allowed.

Return Codes:

rc	*Description*
0	Successful.

Note: The flag is reset at the next initialization.

Spx32GetOCR1Data

Purpose: The Spx32GetOCR1Data function returns the OCR1 input data. This function is supported on the 3891/XP and the 3892/XP.

Calling Sequence:

```
short _System Spx32GetOCR1Data(void *  );
unsigned char buffer[266];

rc = Spx32GetOCR1Data(buffer);
```

buffer

This is a 266-byte data area within the caller's memory space in which this function returns the OCR1 input buffer data. The format is:

2 bytes:	hex 20 00 (valid data status)
2 bytes:	length of data
260 bytes:	document data, right justified, in ASCII
2 bytes:	hex 20 20 (ending pads).

Return Codes:

rc	*Description*
0	Successful.
12	Non-document SPS event; no action taken.
16	Data not valid.
17	Feature not enabled.
100	This function is not supported on this document processor.

Spx32GetOCR2Data

Purpose: The Spx32GetOCR2Data function returns the OCR2 input data. This function is supported on the 3891/XP and 3892/XP.

Calling Sequence:

```
short _System Spx32GetOCR2Data(void *  );
unsigned char buffer[266];

rc = Spx32GetOCR2Data(buffer);
```

buffer

This is a 266-byte data area within the caller's memory space in which this function returns the OCR2 input buffer data. The format is:

2 bytes:	hex 20 00 (valid data status)
2 bytes:	length of data
260 bytes:	document data, right justified, in ASCII
2 bytes:	hex 20 20 (ending pads).

Return Codes:

rc	*Description*
0	Successful.
12	Non-document SPS event; no action taken.
16	Data not valid.
17	Feature not enabled.
100	This function is not supported on this document processor.

Spx32GetOCR3Data

Purpose: The Spx32GetOCR3Data function returns the OCR Feature input data. This function is supported on the 3890/XP.

Calling Sequence:

```
short _System Spx32GetOCR3Data(void *  );
unsigned char buffer[296];

rc = Spx32GetOCR3Data(buffer);
```

buffer

This is a 296-byte data area within the caller's space in which this function returns the OCR3 input buffer data. The format is:

1 byte:	01 = valid data
1 byte:	00 = good read record
	01 = partial read record
	02 = no OCR data for this document
2 bytes:	data length
2 bytes:	Document length
28 bytes:	reserved
260 bytes:	document data, right justified, based on .OTT table
2 bytes:	reserved

Return Codes:

rc	*Description*
0	Successful.
12	Non-document SPS event; no action taken
16	Data not valid
17	Feature not enabled.

Spx32GetOCR3Setup

Purpose: The Spx32GetOCR3Setup function returns the IBM OCR-3 initialization data.

Calling Sequence:

```
short _System Spx32GetOCR3Setup(void *  );
unsigned char OCR3_Setup[5];

rc = Spx32GetOCR3Setup(&OCR3_Setup);
```

OCR3_Setup

This is a 5-byte data area in which this function returns the following setup data.

Byte	*Data Description*
0-1	Hexadecimal value indicating OCR3 Scan Area (Intel Format)
2	Font-1
3	Font-2
4	Vertical Bars and Spaces enabled

Bit	*Bit Description*
0	Spaces Enabled
1	Vertical Bars Enabled
2-7	Reserved

Bytes 2 and 3 are font identifier bytes. The following list represents the value returned for a given font.

Returned Value	*Font*
'00'X	No Font Set
'01'X	OCR A Numeric
'03'X	OCR B Numeric
'04'X	OCR B Alphanumeric (RPQ)
'05'X	E13B (RPQ)

Return Codes:

rc	*Description*
0	Successful.
17	Feature Not Enabled.

Spx32GetOnOfflag

Purpose: The Spx32GetOnOfflag function returns the status of the host online-or-offline flag.

Calling Sequence:

```
short _System Spx32GetOnOfflag(unsigned char *  );
unsigned char flag;

rc = Spx32GetOnOfflag(&flag);
```

flag

This is a 1-byte data area in which this function returns the host online-or-offline flag. The following values apply:

hex 00 The document processor is offline.
hex 01 The document processor is online.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetOpCmdLine

Purpose: The Spx32GetOpCmdLine function returns the operator command line input.

Calling Sequence:

```
short _System Spx32GetOpCmdLine(void *  );
char buffer[65];

rc = Spx32GetOpCmdLine(buffer);
```

buffer

This is a 65-byte data area in which this function returns the operator command line data. All 65 bytes are always transferred.

This function returns the command that this SPS module or a previous SPS module placed in the command field to be displayed on the Run screen.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetOpData

Purpose: The Spx32GetOpData function returns operator-entered data.

Calling Sequence:

```
short _System Spx32GetOpData(void *  );
unsigned char buffer[50];

rc = Spx32GetOpData(buffer);
```

buffer

When the operator causes the operator SPS event to occur, the data from the command line following the EXec command will be available via this function. Entry is made to the Sort module specified by the Run Profile SPSOPER keyword.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetOpMsgNum

Purpose: The Spx32GetOpMsgNum function returns the sort operator message (SOM) number.

Calling Sequence:

```
short _System Spx32GetOpMsgNum(unsigned char *  );
unsigned char msgnum;

rc = Spx32GetOpMsgNum(&msgnum);
```

msgnum

This is a 1-byte field in which this function returns the current SOM number. Message number 0 indicates that the message has been cleared.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetOpMsgTxt

Purpose: The Spx32GetOpMsgTxt function returns the text of the sort operator message.

Calling Sequence:

```
short _System Spx32GetOpMsgTxt(void *  ,unsigned char);
unsigned char text[81];
unsigned char somnum;

rc = Spx32GetOpMsgTxt(text,somnum);
```

text

This is an 81-byte data area in which this function returns the sort operator message (SOM) text. The first byte of the message text is the severity code:

I	Information
W	Warning
E	Error.

Format is ASCII.

somnum

This is the message number (SOM), in the range 1-255, that specifies which message text to return.

Return Codes:

rc	*Description*
0	Successful.
1	SOM number cannot be zero.

Spx32GetPcktLim

Purpose: The Spx32GetPcktLim function returns the pocket limit table.

Calling Sequence:

```
short _System Spx32GetPcktLim(void *  );
unsigned char buffer[72];

rc = Spx32GetPcktLim(buffer);
```

buffer

This is a 72-byte data area in which this function returns the pocket limit table. Upon completion, this data area contains 36 entries of 16 bits each from the pocket limit table. (For more information about the pocket limit table, see the section about additional auxiliary storage in the *3890/XP Enhanced Document Processor Programming Guide*.)

Return Codes:

rc	*Description*
0	Successful.

Spx32GetPktCount

Purpose: The Spx32GetPktCount function returns the pocket counters.

Calling Sequence:

```
short _System Spx32GetPktCount(void *  );
unsigned char pktcntr[72];

rc = Spx32GetPktCount(pktcntr);
```

pktcntr

This is a 72-byte data area in which this function returns 36 two-byte pocket counters in SCI format.

Although the pocket counter area is 96 bytes, the actual pocket counter data is 36 two-byte words. This function strips out unused bytes and compresses the data. (For more information about pocket counters, see the section about the accessible storage map for load/store SCIs in the *3890/XP Enhanced Document Processor Programming Guide*.)

Return Codes:

rc	*Description*
0	Successful.

Spx32GetPktTblUsed

Purpose: The Spx32GetPktTblUsed function returns the pocket limit flag.

Calling Sequence:

```
short _System Spx32GetPktTblUsed(unsigned char *  );
unsigned char flag;

rc = Spx32GetPktTblUsed(&flag);
```

flag

This is a 1-byte data area in which this function returns the value of the pocket limit flag. The following values apply:

hex 00 The initialization data determines pocket limits for merge feed.

hex 01 The pocket table determines pocket limits for merge feed. See the Run Profile PKTDEFtbl keyword.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetPrgEndDat

Purpose: The Spx32GetPrgEndDat function returns the document processor Active Endorsement Area (AEA) data.

Calling Sequence:

```
short _System Spx32GetPrgEndDat(void *  );
unsigned char endorsedata[72];

rc = Spx32GetPrgEndDat(endorsedata);
```

endorsedata

This is a 72-byte data area in which this function returns the AEA data.

The returned data depends on the status of the document processor. If the document processor is not yet initialized, this function returns the last data loaded with the endorse command. After initialization, this function returns either the last data loaded with the endorse command or the data specified in the current run profile. This function returns valid data even if the endorser is not currently being used.

Note: Date substitution and double-wide conversion might have been performed. That means the data can be ASCII or double-wide transport codes. For the 3891/XP and the 3892/XP, double-wide characters are represented by “character-underscore” in ASCII.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetPrgStSelTb

Purpose: The Spx32GetPrgStSelTb function returns the number of segments and selectors for program storage.

Calling Sequence:

```
short _System Spx32GetPrgStSelTb(void *  );
unsigned char buffer[322];

rc = Spx32GetPrgStSelTb(buffer);
```

buffer

This is a 322-byte data area in which this function returns the program storage table of selectors and the count of valid selectors. The count field and each selector occupy one word (2 bytes). Only the number of selectors specified in the count field are copied. Other selector fields in the receiving buffer are not changed and are not valid as selectors.

Return Codes:

rc	***Description***
0	Successful.

Comments: The table definition is:

Word 0	Count of valid selectors
Word 1	Selector for first segment of program storage
Word 2	Selector for second segment of program storage
Words 3 - 160	Other valid selectors in sequence.

To access program storage successfully, you must understand how to use the table. Program storage is allocated in segments of 64K bytes. At the time of allocation, a table of selectors is created to be used for addressability into the area. The table of selectors is preceded by a count of the number of 64K-byte segments allocated. To access storage within the first 64K-byte segment, you must use the first selector.

You could treat program storage as a continuous area that is addressed as a 32-bit integer offset from the first byte. When you do this, the number of the desired selector corresponds to the high-order word of the address (or the 32-bit offset divided by 64K) plus one. The offset into the segment is then the low-order word of the address (or the remainder of the 32-bit offset divided by 64K).

Be sure that you do not attempt to cross a 64K boundary. If you wish to access a block of storage that extends across a segment boundary, you must first access the portion addressed by the lower-numbered selector. Then the next-higher selector must be retrieved and used to access the remainder of the data. The function Spx32GetProgStore allows you to obtain data that spans program storage segments as a single operation.

Note: This SPX is provided only for compatibility with existing sort programs. It is recommended that new sort programs use Spx32GetProgStoreAddr (see “Spx32GetProgStoreAddr” on page 3-67).

Spx32GetPrgStSize

Purpose: The Spx32GetPrgStSize function returns the size of program storage.

Calling Sequence:

```
short _System Spx32GetPrgStSize(unsigned long *  );
unsigned long buffer;

rc = Spx32GetPrgStSize(&buffer);
```

buffer

This is a 32-bit data area (32-bit Intel** format integer) in which this function returns the size of program storage as a fixed, 32-bit binary value.

Return Codes:

rc	*Description*
0	Successful.

** Trademark of Intel

Spx32GetPrgSwtch

Purpose: The Spx32GetPrgSwtch function returns the program switches data.

Calling Sequence:

```
short _System Spx32GetPrgSwtch(void *  );
unsigned char buffer;

rc = Spx32GetPrgSwtch(&buffer);
```

buffer

This is a 1-byte data area in which this function returns the program switches.

Note: These switches can be changed from the Run screen.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetProcBufDat

Purpose: The Spx32GetProcBufDat function returns data from the process buffer.

Calling Sequence:

```
short _System Spx32GetProcBufDat(void *  );
unsigned char proc[244];

rc = Spx32GetProcBufDat(proc);
```

proc

This is a 244-byte data area in which this function returns the process buffer data in a right-aligned format.

Note: Use Spx32GetProcBufHdr to retrieve the process buffer header.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetProcBufDL

Purpose: The Spx32GetProcBufDL function returns the length (in Intel format) of the process buffer.

Calling Sequence:

```
short _System Spx32GetProcBufDL(unsigned short *  );
unsigned short length;

rc = Spx32GetProcBufDL(&length);
```

length

This is a 2-byte area in which this function returns the length of the process buffer data as a 16-bit, binary integer.

Return Codes:

rc	*Description*
0	Successful.

Note: The length does NOT include the length of the process buffer header (length 12).

Spx32GetProcBufHdL

Purpose: The Spx32GetProcBufHdL function returns the length (in Intel format) of the process buffer header.

Calling Sequence:

```
short _System Spx32GetProcBufHdl(void *  );
unsigned short length;

rc = Spx32GetProcBufHdl(&length);
```

length

This is a 2-byte data area in which this function returns the length of the (12-byte) process buffer header as a 16-bit, binary integer.

Return Codes:

rc	*Description*
0	Successful.

Note: Use Spx32GetProcBufHdr to retrieve the process buffer header.

Spx32GetProcBufHdr

Purpose: The Spx32GetProcBufHdr function returns the process buffer header.

Calling Sequence:

```
short _System Spx32GetProcBufHdr(void *  );
unsigned char buffer[12];

rc = Spx32GetProcBufHdr(buffer);
```

buffer

This is a 12-byte data area in which this function stores the process buffer header. See "Document Data Record Header" in the *3890/XP Enhanced Document Processor Programming Guide*.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetProgStore

Purpose: The Spx32GetProgStore function returns data from program storage.

Calling Sequence:

```
short _System Spx32GetProgStore(void *  ,unsigned long,unsigned long);
unsigned char buffer[300];
unsigned long disp;
unsigned long length;

rc = Spx32GetProgStore(buffer,disp,length);
```

buffer
This is a data area in which this function returns the data from program storage.

disp
This is a 32-bit displacement into program storage.

length
This is the length in bytes of the data to be copied. It must be less than 65 536 bytes. To copy more than 65 535 bytes of data, use a series of function calls.

Return Codes:

rc	*Description*
0	Successful
1	Displacement value greater than program store
2	Displacement plus length out-of-bounds
3	Length not-greater-than zero.

Note: This function does not check for 3890-emulation mode but always returns the actual program storage data from the specified displacement.

Note: This SPX is supplied only for compatibility with old sort programs. It is recommended that this function not be used in new 32-bit code, instead, Spx32GetProgStoreFlat (see “Spx32GetProgStoreFlat” on page 3-68) should be used. Spx32GetProgStoreFlat does not have the 64k limit on length;

Spx32GetProgStoreAddr

Purpose: The Spx32GetProgStoreAddr function returns the address of the start of program storage.

Calling Sequence:

```
short _System Spx32GetProgStoreAddr(void *);
void *buffer;

rc = Spx32GetProgStoreAddr(&buffer)
```

buffer
This is a data area in which this function returns the start of program storage.

Return Codes:

rc	*Description*
0	Successful

Spx32GetProgStoreFlat

Purpose: The Spx32GetProgStoreFlat function returns data from program storage to a specified buffer.

Calling Sequence:

```
short _System Spx32GetProgStoreFlat( void *, unsigned long, unsigned long );
void *buffer;
unsigned long disp;
unsigned long length;

rc = Spx32GetProgStoreFlat(&buffer,disp,length);
```

buffer
This is a data area in which this function returns the data from program storage.

disp
This is a 32-bit displacement into program storage.

length
This is the length in bytes of the data to be copied.

Return Codes:

rc	*Description*
0	Successful
1	Displacement value greater than program store
2	Displacement plus length greater than end of program store.

Spx32GetReinitLoadPath

Purpose: Spx32GetReinitLoadPath returns the current value of the re-initialization load path.

Calling Sequence:

```
short _System Spx32GetReinitLoadPath(void *  );
unsigned char ReinitLoadPath[66];

rc = Spx32GetReinitLoadPath(ReinitLoadPath);
```

ReinitLoadPath
is the directory path to be used to load the run profile and associated files during re-initialization. ReinitLoadPath must be 66 bytes long to contain the maximum path value.

Return Codes:

rc	Description
0	Successful.

Notes:

1. Use **Spx32PutReinitLoadPath** to change the path that will be used to load the run profile and associated files during re-initialization.
2. Use **Spx32PutReinitReq** to actually request re-initialization during the SPSINIT event.

Spx32GetReinitReq

Purpose: Spx32GetReinitReq gets the current setting of the re-initialization request flag.

Calling Sequence:

```
short _System Spx32GetReinitReq(unsigned char *  );
unsigned char ReinitReq;

rc = Spx32GetReinitReq(&ReinitReq);
```

ReinitReq
is a one-byte flag that indicates if re-initialization has been requested. Possible values are:

0 Re-initialization has not been requested

1 Re-initialization has been requested

Return Codes:

rc	Description
0	Successful.

Notes:

1. Use **Spx32PutReinitReq** to request re-initialization during the SPSINIT event.

Spx32GetRPERec

Purpose: The Spx32GetRPERec function returns one Run Profile Element Record from the Program Storage Data (PSD) table in a 32-byte data area that the calling program specifies.

Calling Sequence:

```
short _System Spx32GetRPERec(void *  ,unsigned long);
unsigned byte buffer[32];
unsigned long element;

rc = Spx32GetRPERec(buffer,element);
```

buffer

This is a 32-byte data area in which this function returns the data. The format is:

8 bytes:	filename
8 bytes:	user tag
4 bytes:	offset in Program Storage - IBM format
4 bytes:	length - IBM format
4 bytes:	offset in Program Storage - Intel format
4 bytes:	length - Intel format.

element

This identifies the record element within the PSD table to be copied. A value of 1 copies the first record.

To get the complete table, use a loop and increment the value of the element until the return code is 1. The following pseudo code is an example of this type of loop:

```
element = 1;
repeat
  rc = Spx32GetRPERec(buffer,element);
  element++;
until rc = 1;
```

Return Codes:

rc	*Description*
0	Successful
1	Element value out of range.

Spx32GetRqNotInit

Purpose: The Spx32GetRqNotInit function returns the value of the request-not-initialized flag.

This flag is reset by run initialization or by the Spx32PutRQNotInit function. (For more information about request-not-initialized, see the section about additional auxiliary storage in the *3890/XP Enhanced Document Processor Programming Guide*.)

Calling Sequence:

```
short _System Spx32GetRqNotInit(unsigned char *  );
unsigned char flag;

rc = Spx32GetRqNotInit(&flag);
```

flag

This is a 1-byte data area in which this function returns the value of the request-not-initialized flag. The following values apply:

hex 00 The flag is not set.
hex 01 The flag is set.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetSavArea

Purpose: The Spx32GetSavArea function returns up to 256 bytes of data from the Save Area.

Calling Sequence:

```
short _System Spx32GetSavArea(void *  ,unsigned long,unsigned long);
unsigned char buffer[300];
unsigned long displacement;
unsigned long length;

rc = Spx32GetSavArea(buffer,displacement,length);
```

buffer
This is a data area to which the data is copied.

displacement
This is the displacement within the save area of the first byte to be copied (specify 0 to begin at the first byte). The data is copied, beginning at the low-order address byte and proceeding to the high-order address bytes, that is, left to right.

length
This is the number of bytes to be copied.

Return Codes:

rc	*Description*
0	Successful.
1	The displacement was out-of-bounds (greater than 255).
2	The displacement plus the length was out-of-range (greater than 256).
3	The length was not greater than zero.

Spx32GetSCICount

Purpose: The Spx32GetSCICount function returns the values of the SCI counters.

Calling Sequence:

```
short _System Spx32GetSCICount(void *  );
unsigned char counter[32];

rc = Spx32GetSCICount(counter);
```

counter

This is a 32-byte data area in which this function returns the SCI counters in SCI (IBM) format. Each counter is 2-bytes long.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetSCIErrData

Purpose: The Spx32GetSCIErrData function returns the SCI error data.

Calling Sequence:

```
short _System Spx32GetSCIErrData(void *  );
unsigned char buffer[12];

rc = Spx32GetSCIErrData(buffer);
```

buffer

This is a 12-byte data area in which this function returns the SCI error data. See "SCI-Error Record Header" in the *3890/XP Enhanced Document Processor Programming Guide*.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetSEC

Purpose: The Spx32GetSEC function returns the symbol-error-correction-value byte.

Calling Sequence:

```
short _System Spx32GetSEC(unsigned char *  );
unsigned char secbyte;

rc = Spx32GetSEC(&secbyte);
```

secbyte
This is a 1-byte data area in which this function returns the symbol-error-correction value. This is the same as initialization data byte 55.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetSerial

Purpose: The Spx32GetSerial function returns the machine serial number.

Calling Sequence:

```
short _System Spx32GetSerial(unsigned char *  );
unsigned char serial[3];

rc = Spx32GetSerial(serial);
```

serial

This is a 3-byte data area in which this function returns the machine serial number in unsigned, packed-decimal format. The 3890/XP uses a 5-digit serial number with a leading 0. The 3891/XP and the 3892/XP use a 6-digit serial number. The machine serial number is displayed on the Main Menu.

The UT/XP does not support a manufacturing serial number. For this function the UT/XP returns to the control program a descriptive name that may contain non-numeric data.

Return Codes:

rc	*Description*
0	Successful.
100	This function is not supported on this document processor.

Spx32GetSpxEvent

Purpose: The Spx32GetSPXEvent function returns the Sort Program Sequence event ID.

Calling Sequence:

```
short _System Spx32GetSpxEvent(void *  );
unsigned char EventID[16];

rc = Spx32GetSpxEvent(EventID);
```

EventID

This is a 16-byte data area in which this function returns the Sort program event ID.

Return Codes:

rc	*Description*
0	Successful.

Each event is represented by a byte. A value of 1 indicates the current event. The events, in order, are:

0	SPSVERIFY
1	SPSCORR
2	SPSFORMAT
3	SPSIMATCH
4	SPSDOC
5	SPSPOST
6-11	Reserved events
12	SPSPAUSE
13	SPSINIT
14	SPSMS
15	SPSOPER.

Spx32GetStackers

Purpose: The Spx32GetStackers function returns the number of stackers installed.

Calling Sequence:

```
short _System Spx32GetStackers(void *  );
unsigned short stacker;

rc = Spx32GetStackers(&stacker);
```

stacker
This is a 1-byte data area in which this function returns the number of stackers installed.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetTimerData

Purpose: The Spx32GetTimerData function returns a 1600 byte data record indicating the current timing data.

Calling Sequence:

```
short _System Spx32GetTimerData(void *  );

typedef struct _TIMERDATA
{
   unsigned char starttime[8];
   unsigned char endtime[8];
   unsigned char timediff[8];
} TIMERDATA

timerdata TIMERDATA[10][10];


rc = Spx32GetTimerData(timerdata);
```

timerdata

This is a 2400 byte data area in which this function returns the current timing data. It returns the start time count, end time count, and difference for the last 10 timings. All 10 events (even if not timed) are returned.

Return Codes:

rc	*Description*
0	Successful.

Figure 3-1 (Page 1 of 3). Data Definition from Spx32GetTimerData

Run Initialization	Start Time 1	End Time 1	Difference
	Start Time 2	End Time 2	Difference
	Start Time 3	End Time 3	Difference
	Start Time 4	End Time 4	Difference
	Start Time 5	End Time 5	Difference
	Start Time 6	End Time 6	Difference
	Start Time 7	End Time 7	Difference
	Start Time 8	End Time 8	Difference
	Start Time 9	End Time 9	Difference
	Start Time 10	End Time 10	Difference
Verify	Start Time 1	End Time 1	Difference
	Start Time 2	End Time 2	Difference
	Start Time 3	End Time 3	Difference
	Start Time 4	End Time 4	Difference
	Start Time 5	End Time 5	Difference
	Start Time 6	End Time 6	Difference
	Start Time 7	End Time 7	Difference
	Start Time 8	End Time 8	Difference
	Start Time 9	End Time 9	Difference
	Start Time 10	End Time 10	Difference
Correction	Start Time 1	End Time 1	Difference
	Start Time 2	End Time 2	Difference
	Start Time 3	End Time 3	Difference
	Start Time 4	End Time 4	Difference

Figure 3-1 (Page 2 of 3). Data Definition from Spx32GetTimerData

	Start Time 5	End Time 5	Difference
	Start Time 6	End Time 6	Difference
	Start Time 7	End Time 7	Difference
	Start Time 8	End Time 8	Difference
	Start Time 9	End Time 9	Difference
	Start Time 10	End Time 10	Difference
Format	Start Time 1	End Time 1	Difference
	Start Time 2	End Time 2	Difference
	Start Time 3	End Time 3	Difference
	Start Time 4	End Time 4	Difference
	Start Time 5	End Time 5	Difference
	Start Time 6	End Time 6	Difference
	Start Time 7	End Time 7	Difference
	Start Time 8	End Time 8	Difference
	Start Time 9	End Time 9	Difference
	Start Time 10	End Time 10	Difference
Code Line Data Match	Start Time 1	End Time 1	Difference
	Start Time 2	End Time 2	Difference
	Start Time 3	End Time 3	Difference
	Start Time 4	End Time 4	Difference
	Start Time 5	End Time 5	Difference
	Start Time 6	End Time 6	Difference
	Start Time 7	End Time 7	Difference
	Start Time 8	End Time 8	Difference
	Start Time 9	End Time 9	Difference
	Start Time 10	End Time 10	Difference
Document	Start Time 1	End Time 1	Difference
	Start Time 2	End Time 2	Difference
	Start Time 3	End Time 3	Difference
	Start Time 4	End Time 4	Difference
	Start Time 5	End Time 5	Difference
	Start Time 6	End Time 6	Difference
	Start Time 7	End Time 7	Difference
	Start Time 8	End Time 8	Difference
	Start Time 9	End Time 9	Difference
	Start Time 10	End Time 10	Difference
Post	Start Time 1	End Time 1	Difference
	Start Time 2	End Time 2	Difference
	Start Time 3	End Time 3	Difference
	Start Time 4	End Time 4	Difference
	Start Time 5	End Time 5	Difference
	Start Time 6	End Time 6	Difference
	Start Time 7	End Time 7	Difference
	Start Time 8	End Time 8	Difference
	Start Time 9	End Time 9	Difference
	Start Time 10	End Time 10	Difference
Pause	Start Time 1	End Time 1	Difference
	Start Time 2	End Time 2	Difference
	Start Time 3	End Time 3	Difference
	Start Time 4	End Time 4	Difference
	Start Time 5	End Time 5	Difference
	Start Time 6	End Time 6	Difference

Figure 3-1 (Page 3 of 3). Data Definition from Spx32GetTimerData

	Start Time 7	End Time 7	Difference
	Start Time 8	End Time 8	Difference
	Start Time 9	End Time 9	Difference
	Start Time 10	End Time 10	Difference
Motor Stop	Start Time 1	End Time 1	Difference
	Start Time 2	End Time 2	Difference
	Start Time 3	End Time 3	Difference
	Start Time 4	End Time 4	Difference
	Start Time 5	End Time 5	Difference
	Start Time 6	End Time 6	Difference
	Start Time 7	End Time 7	Difference
	Start Time 8	End Time 8	Difference
	Start Time 9	End Time 9	Difference
	Start Time 10	End Time 10	Difference
Operator	Start Time 1	End Time 1	Difference
	Start Time 2	End Time 2	Difference
	Start Time 3	End Time 3	Difference
	Start Time 4	End Time 4	Difference
	Start Time 5	End Time 5	Difference
	Start Time 6	End Time 6	Difference
	Start Time 7	End Time 7	Difference
	Start Time 8	End Time 8	Difference
	Start Time 9	End Time 9	Difference
	Start Time 10	End Time 10	Difference

Spx32GetTimerFreq

Purpose: The Spx32GetTimerFreq function returns the frequency of the OS/2 high resolution timer.

Calling Sequence:

```
short _System Spx32GetTimerFreq(void *  );
unsigned long timerfreq;

rc = Spx32GetTimerFreq(&timerfreq);
```

timerfreq
This is a 32-bit integer that is to receive the frequency of the OS/2 high resolution timer.

Return Codes:

rc	*Description*
0	Successful.
Non-zero	Unsuccessful.

Spx32GetTimerOverhead

Purpose: The Spx32GetTimerOverhead function returns data indicating the number of counts that the call to the timer has. Subtracting this value from the counts returned in Spx32GetTimerData gives the actual counts that occured while timing.

Calling Sequence:

```
short _System Spx32GetTimerOverhead(void *  );
unsigned char timeroverhead[8];

rc = Spx32GetTimerOverhead(timeroverhead);
```

timeroverhead

This is a 64-bit data area in which this function returns the number of counts that occur when executing the timing function.

Return Codes:

rc	*Description*
0	Successful.

Spx32GetTimeStamp

Purpose: The Spx32GetTimeStamp function returns the current time stamp from the OS/2 high resolution timer.

Calling Sequence:

```
short _System Spx32GetTimeStamp(void *  );
unsigned char timestamp[8];

rc = Spx32GetTimerStamp(timerstamp);
```

timestamp
This is a 64-bit data area in which this function returns the current count of the OS/2 high resolution timer.

Return Codes:

rc	*Description*
0	Successful.
Non-zero	Unsuccessful.

Spx32GetTimerSelect

Purpose: The Spx32GetTimerSelect function returns data indicating the currently selected events to time.

Calling Sequence:

```
short _System Spx32GetTimerSelect(void *  );
unsigned short timerselect;

rc = Spx32GetTimerSelect(&timerselect);
```

timerselect

This is a 16-bit data area in which this function returns the state indicating the currently selected events to time.

Return Codes:

rc	*Description*
0	Successful.

Bit definitions:

0	Run Initialization
1	Verify
2	Correction
3	Format
4	Code Line Data Match
5	Document
6	Post
7	Pause
8	Motor Stop
9	Operator

Spx32GetUsrMPDef

Purpose: The Spx32GetUsrMPDef function returns the 36 three-byte module-pocket names defined in the Machine Profile.

Calling Sequence:

```
short _System Spx32GetUsrMPDef(void *  );
unsigned char buffer[108];

rc = Spx32GetUsrMPDef(buffer);
```

buffer
This is a 108-byte data area in which this function returns the module-pocket data table (in ASCII format).

Return Codes:

rc	*Description*
0	Successful.

Spx32GetUsrStat

Purpose: The Spx32GetUsrStat function returns user statistics data.

Calling Sequence:

```
short _System Spx32GetUsrStat(unsigned char *  );
unsigned char buffer;

rc = Spx32GetUsrStat(&buffer);
```

buffer

This is a 1-byte data area in which this function returns the user statistics data (“user stats”).

Return Codes:

rc	*Description*
0	Successful.

Spx32GetVerNum

Purpose: The Spx32GetVerNum function returns the control program version number.

Calling Sequence:

```
short _System Spx32GetVerNum(void *  );
char vernum[5];

rc = Spx32GetVerNum(vernum);
```

vernum

This is a 5-character data area in which this function returns the control programV version number. This number is an ASCII string of 5 characters in the format NN.NN. The first two characters contain the portion of the version number that is greater than or equal to 1 and the last two characters contain the portion of the version number that is less than 1, but greater than or equal to zero (for example, Version 01.05).

Return Codes:

rc	*Description*
0	Successful.

Spx32GetWrkReg

Purpose: The Spx32GetWrkReg function returns the contents of the work register. SCI programs use the work register as a temporary storage area to manipulate data.

Calling Sequence:

```
short _System Spx32GetWrkReg(void *  );
unsigned char buffer[16];

rc = Spx32GetWrkReg(buffer);
```

buffer

This is a 16-byte data area in which this function returns the contents of the work register.

Return Codes:

rc	*Description*
0	Successful.

Spx32IFD

Purpose: The Spx32IFD function requests the Initialize Feature Data function.

Calling Sequence:

```
short _System Spx32IFD(void);

rc = Spx32IFD();
```

Return Codes:

rc	*Description*
0	Successful.
1	Machine was online.

Notes:

1. This function runs only when the document processor is offline.
2. This function is the same as the SCI “IFD 0” macro, except that it does not store the mark register low-order byte in byte 0 of initialization data (as SCI does). You should use the Spx32PutProgStore function to completely emulate the SCI “IFD 0” macro.

Spx32LoadFixM

Purpose: The Spx32LoadFixM function loads a fixed message to be used for endorsement. This function is called at SPSINIT-time. Use of the message is specified at SPSDOC-time. See Spx32FixM. This function is only supported on the 3891/XP, 3892/XP, and UT/XP.

Calling Sequence:

```
short _System Spx32LoadFixM(void *  );
unsigned char buffer[50];

rc = Spx32LoadFixM(buffer);
```

buffer

This is a 50-byte data area that has the message. The first byte is the message number (1-15), followed by the text (0 to 48 ASCII), ended by hex 7F. A null message is defined without text; however, the 7F is required. If a null message is "printed" (see Spx32FixM), it occupies no space on the document. On the UT/XP this is a 68 byte data area that has the message. The first byte is the message number (1-15), followed by the text (0 to 66 ascii bytes), ended by hex 7F.

Return Codes:

rc	*Description*
0	Successful.
1	Invalid message number.
18	Command failed.
100	This function is not supported on this document processor.

Spx32LogMrg

Purpose: The Spx32LogMrg logical merge function requests document operations without a merge feeder: merge before main, merge on pocket count, and feed merge. This function is only supported on the 3891/XP and the 3892/XP.

Calling Sequence:

```
short _System Spx32LogMrg(unsigned char *  );
unsigned char buffer;

rc = Spx32LogMrg(&buffer);
```

buffer

A 1-byte data area in which this function returns the Logical Merge Requested flag.

hex 00 Not requested.
hex 01 Requested. User program should start a tolerance countdown.

Return Codes:

rc	*Description*
0	Successful.
100	This function is not supported on this document processor.

Notes:

1. Logical merge mode is entered by calling this function, regardless of whether the machine has a merge feeder or not. The Control Program will not request merge feeder documents from the hardware. Instead, the operator inserts merge documents in the main feeder work stream. This mode is reset at the next initialization.
2. For merge before main operation, this function must be called at SPSINIT time, so logical merge mode is set before the control routine starts feed requests.
3. It is recommended that the user's tolerance count be applied on a per-pocket basis. When the count is reached and a merge document has not been seen (for that pocket), the user program must decide whether to stop the machine and what to tell the operator (Spx32DSG and Spx32PutAdditnlMsg).
4. The user program should look at the current document (Spx32GetProcBufDat) and determine if it is a merge document. If so, the merge flag should be set in the header (byte 8 bit 4, via Spx32GetProcBufHdr and Spx32PutProcBufHdr). This must be done as the last module at SPSFORMAT-time prior to execution of the remaining Sort program logic.
5. If the mode is active, the logical-merge-requested flag is set whenever there is a request for a merge document (Spx32FM), including merge before main. This function clears the flag after returning it to the caller.

Spx32NoOP

Purpose: The Spx32NoOP function allows the Control Program to check for time out or late TSCI (terminate stacker control instructions) termination.

The Control Program makes a time-out check at the start of every Spx32 function. A Sort program that has a long-running loop without other Spx32 calls should include Spx32NoOp as part of the loop.

Calling Sequence:

```
short _System Spx32NoOP(void);

rc = Spx32NoOP();
```

Return Codes:

rc	*Description*
0	Successful.

Spx32NoRollOn

Purpose: The Spx32NoRollOn function will disable the Rollon Endorser on a per document basis. This function is supported only on the 3891/XP and 3892/XP.

This function will set a bit in EGA that will allow the control program to NOT send the ACTIVE command to the traansport. This bit will be reset every document after it is tested.

Calling Sequence:

```
short _System Spx32NoRollOn(void);

rc = Spx32NoRollOn();
```

Return Codes:

rc	*Description*
0	Successful.
12	Non-document SPS event; no action taken.
100	This function is not supported on this document processor.

Spx32Pause

Purpose: Spx32Pause sets the SPS pause request flag.

This function sets a pause request, which stops the feeding of documents but allows documents that are in process to continue processing normally.

Once the in-process documents have processed and the sorter has paused, the control routine calls the user Sort program at the SPSPAUSE entry points, as specified in the Run Profile. This is a non-document event, and the Run Profile Spx32NDPTIME parameter applies.

When control has returned from the user routines, document feeding resumes. However, if a transport motor time-out has occurred, a message is displayed on the screen and the operator has to press the START key to restart the flow of documents.

Calling Sequence:

```
short _System Spx32Pause(void);

rc = Spx32Pause();
```

Return Codes:

rc	*Description*
0	Successful.
12	Non-document SPS event; no action taken.

Spx32PutAdditnlMsg

Purpose: The Spx32PutAdditnlMsg function stores a message in the additional sort message area.

Calling Sequence:

```
short _System Spx32PutAdditnlMsg(void *  );
char buffer[40];

rc = Spx32PutAdditnlMsg(buffer);
```

buffer

This is a 40-byte data area from which the message text is copied to the additional sort message area of the document processor.

If you specify less than 40 bytes, you might receive a protection violation message and lose data.

The message appears on the operator Run screen alternate message line.

Return Codes:

rc	*Description*
0	Successful.

Spx32PutAddMsglong

Purpose: The Spx32PutAddMsglong function stores a message in the additional sort message area.

Calling Sequence:

```
short _System Spx32PutAddMsgLong(void *  );
char buffer[80];

rc = Spx32PutAddMsgLong(buffer);
```

buffer

This is an 80-byte data area from which the message text is copied to the additional sort message area of the document processor.

The message will appear on the operator Run screen alternate message line.

Return Codes:

rc	*Description*
0	Successful.

Spx32PutAltINFArea

Purpose: The Spx32PutAltINFArea function stores data in the alternate INF flag and data area.

The alternate INF number must be in unsigned, packed-decimal format. Specify blanks as X'A' and use all 10 digits. A question mark (?) indicates that characters in the output are not valid. The invalid-INF-data bit is set on in the process buffer header to indicate that INF data is not valid.

Calling Sequence:

```
short _System Spx32PutAltINFArea(void *  );
unsigned char buffer[6];

rc = Spx32PutAltINFArea(buffer);
```

buffer

This is a 6-byte data area from which this function copies the alternate INF flag and data. The first byte is the flag; the remaining 5 bytes are the 10 digits, in BCD.

The following flag values apply:

00 Alternate data is not used.
01 Alternate INF data is used.

Note: The initialization data (byte 82, bit 2) can inhibit this function. This function clears the flag after usage. The flag must be set on for each document that is to be printed with the alternate INF.

Return Codes:

rc	*Description*
0	Successful.

Spx32PutCDT

Purpose: The Spx32PutCDT function copies code-line definition data to the table in auxiliary storage.

Calling Sequence:

```
short _System Spx32PutCDT(void *  );
unsigned char buffer[149];

rc = Spx32PutCDT(buffer);
```

buffer

This is a 149-byte data area from which this function copies the code-line definition data. The format is defined in Appendix C of the *3890/XP Enhanced Document Processor Programming Guide*.

Return Codes:

rc	*Description*
0	Successful.

Note: This function allows the user Sort program to support multi-format code lines. See Spx32GetCDT.

Spx32PutDocsToPause

Purpose: The Spx32PutDocsToPause function sets a value in number of documents to process before a pause is requested to test syncronization between the sorter and the Image Capture Processor.

This Spx is to be used in conjunction with the PAUSEXP.DLL supplied on the control program diskette.

Calling Sequence:

```
short _System Spx32PutDocsToPause(void *  );
unsigned long buffer;

rc = Spx32PutDocsToPause(&buffer);
```

buffer

This is an unsigned long data area with valid values between 1 and 65535. This value represents the number of items to be processed before the sorter is paused to test for syncronization in the Image capture processor. If Image is not enabled this spx will have no effect.

Return Codes:

rc	*Description*
0	Successful.
1	value requested is out of range.

Spx32PutEncData

Purpose: The Spx32PutEncData function copies data to the 3892/XP power encoder buffer data.

Calling Sequence:

```
short _System Spx32PutEncData(void *  );
unsigned char buffer[85];

rc = Spx32PutEncData(buffer);
```

buffer

This is an 85-byte data area set up by the calling program.

The data is in ASCII, left-justified in the buffer, and ended by a hex 7F. Unused bytes after the 7F are ignored. Encoding will be right-justified on the document. Multiple spaces can be represented by a tab sequence of 09 *count*, where *count* is the number of spaces required.

Return Codes:

rc	*Description*
0	Successful.
8	An invalid value was specified.
17	The feature is not enabled.
19	The hex 7F data terminator is missing.
100	This function is not supported on this document processor.

Notes:

1. You should ensure that the data does not exceed the permissible maximum for the given document size.
2. Valid characters are:

Hex	Character
09	Tab
20	Space
23	
24	
2D	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
3C	

3. If you execute this function on the UT/XP the Spx32PutHSPEData function is called to execute the actual request. The return code for that function should be used to determine the results of the request.

Spx32PutEncErrDisp

Purpose: The Spx32PutEncErrDisp function sets the disposition for Power Encoder verification error documents. This function is supported on the 3892/XP and the UT/XP.

Calling Sequence:

```
short _System Spx32PutEncErrDisp(void *  );
unsigned char buffer[3];

rc = Spx32PutEncErrDisp(buffer);
```

buffer

This is 3 bytes of data used to set up the document disposition for reject, character substitutions, and device errors.

Byte 1 Disposition mode for reject documents
0 = Pocket normally, ignore error
1 = Place in reject pocket

Byte 2 Disposition mode for character substitution documents
0 = Pocket normally, ignore error
1 = Place in reject pocket

Byte 3 Disposition mode for encoder device error documents
0 = Pocket normally, ignore error
1 = Place in reject pocket

Return Codes:

rc	*Description*
0	Successful.
4	The function was issued while actively processing documents. No action was taken.
8	An invalid value was specified.
17	The feature is not enabled.
100	This function is not supported on this document processor.

Spx32PutEncSetup

Purpose: The Spx32PutEncSetup function specifies the Power Encoder verification failure thresholds. This function is supported on the 3892/XP and the UT/XP.

Calling Sequence:

```
short _System Spx32PutEncSetup(void *  );
unsigned char buffer[4];

rc = Spx32PutEncSetup(buffer);
```

buffer

This is a 4-byte area set up by the calling program that contains the power encoder setup parameters.

Byte 1 Threshold value for consecutive document reject errors, zero to 255 documents.

Byte 2 Threshold value for consecutive document substitution errors, zero to 255 documents.

Byte 3 Threshold value for cumulative document reject errors, zero to 255 documents.

Byte 4 Threshold value for cumulative document substitution errors, zero to 255 documents.

Return Codes:

rc	*Description*
0	Successful.
4	The function was issued while actively processing documents. No action was taken.
17	The feature is not enabled.
100	This function is not supported on this document processor.

Note: If you execute this function on a UT/XP the Spx32PutHSPESetup function is called to execute the request. The return codes for that function should be used to determine the results of the request.

Spx32PutHOZNum

Purpose: The Spx32PutHOZNum function stores a 1-byte HOZ correction control number (see byte 60 of IREC).

Calling Sequence:

```
short _System Spx32PutHOZNum(unsigned char);
unsigned char hozbyte;

rc = Spx32PutHOZNum(hozbyte);
```

hozbyte

This is an actual value that is written to the HOZ correction byte in the document processor. Any change to this value takes effect on the next document.

Although one byte of data is written, only the low-order 4 bits of data have any significance. The high-order 4 bits are zeros. This is a binary integer that ranges from 0 to 15.

Return Codes:

rc	*Description*
0	Successful.

Spx32PutHSPEData

Purpose: The Spx32PutEncData function copies data to the UT/XP power encoder buffer data.

Calling Sequence:

```
short _System Spx32PutEncData(void *);
unsigned char buffer[66];

rc = Spx32PutEncData(buffer);
```

buffer

This is an 66-byte data area set up by the calling program.

The data is in ASCII, left-justified in the buffer, and ended by a hex 7F. Unused bytes after the 7F are ignored. Encoding will be right-justified on the document. Multiple spaces can be represented by a tab sequence of 09 *count*, where *count* is the number of spaces required.

Return Codes:

rc	*Description*
0	Successful.
8	An invalid value was specified.
17	The feature is not enabled.
19	The hex 7F data terminator is missing.
20	Number of characters greater than 17 with only one encoder configured or with 2 encoders configured and running at 1000 docuemnts per minute.
21	Number of printable characters specified less than 20 with 2 encoders configured.
22	Number of printable characters specified less than 10 with a minimum of 10 required for either encoder.
23	Number of characters specified greater than 34 with 2 encoders configured.
100	This function is not supported on this document processor.

Notes:

1. You should ensure that the data does not exceed the permissible maximum for the given document size.
2. Valid characters are:

Hex	Character
09	Tab
20	Space
23	
24	
2D	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
3C	

3. If you execute this function on the 3892/XP the Spx32PutEncData function is called to execute the actual request. The return code for that function should be used to determine the results of the request.

Spx32PutHSPESetup

Purpose: The Spx32PutHSPESetup function specifies the Power Encoder verification failure thresholds. This function is supported on the 3892/XP and the UT/XP.

Calling Sequence:

```
short _System Spx32PutHSPESetup(void *);
unsigned char buffer[4];

rc = Spx32PutHSPESetup(buffer);
```

buffer

This is a 4-byte area set up by the calling program that contains the power encoder setup parameters.

Byte 1 Threshold value for consecutive document reject errors, zero to 255 documents.

Byte 2 Threshold value for consecutive document substitution errors, zero to 255 documents.

Byte 3 Threshold value for cumulative document reject errors, zero to 255 documents.

Byte 4 Threshold value for cumulative document substitution errors, zero to 255 documents.

Byte 5 Value for feed rate for encoding; 1=600 DPM, 2=1000 DPM. When encoding more than 17 characters on the UT/XP the feed rate must be set to 600 DPM.

Return Codes:

rc	*Description*
0	Successful.
4	The function was issued while actively processing documents. No action was taken.
17	The feature is not enabled.
18	Request to set feed rate failed.
100	This function is not supported on this document processor.

Note: If you execute this function on a 3892/XP the Spx32PutEncSetup function is called to execute the request. The return codes for that function should be used to determine the results of the request. Byte 5 will not be passed to Spx32PutEncSetup.

Spx32PutImgBufPntr

Purpose: The Spx32PutImgBufPntr function copies data to the 12 code line data match buffer pointers in the document processor.

Calling Sequence:

```
short _System Spx32PutImgBufPntr(void *  );
void *pointer[12];

rc = Spx32PutImgBufPntr(pointer);
```

pointer

This is a data area of twelve 32-bit pointers, in Intel format, that is used to control the code line data match and the extended code line data match.

Only pointers 1-4 and 7-10 are copied.

The following are the code line data match buffer pointers:

- Low limit
- High limit
- Search center
- Starter
- Wrap pointer (protected)
- Wrap reset (protected).

When the extended code line data match is active, the next six pointers are copies of these first six pointers. SCII saves the first six pointers and increments the first four pointers for a not-found condition when extended code line data match is active. Restoration is made if an extended code line data match call is made. This function protects the wrap pointer and the wrap reset from any alteration.

Warning: You must change these pointers by an amount that is equal to the record length (process-buffer-data length plus process-buffer-header length) or a multiple of the record length. You should check these pointers after you change them to see whether they are equal to or greater than the wrap pointer. If they are, the pointer that is nearest the wrap value must be reset to the wrap reset value. If you do not manage the pointers correctly, a host interface failure or *protection violation* can occur. Failure could require a system reload to recover from the error.

Return Codes:

rc	*Description*
0	Successful.

Spx32PutInkJetCtrl

Purpose: The Spx32PutInkJetCtrl function allows the user program to control the front and rear ink jet endorsement printers on the 3891/XP, 3892/XP and the UT/XP.

Calling Sequence:

```
short _System Spx32PutInkJetCtrl(void *  );
unsigned char buffer[69];

rc = Spx32PutInkJetCtrl(buffer);
```

buffer

A 69-byte data area is described below.

Byte	Bit	Description
1		Print line and side request X'00' = Front side line 1 X'01' = Back side line 1 X'02' = Front side line 2 X'03' = Back side line 2 X'04' = Front side line 3 X'05' = Back side line 3
2	0	Print the line specified in byte 1 inverted and rotated.
	1	Print the following variable text as specified in bytes 3 through 51.
	2	Print fixed message number, per bits 4 through 7.
	3	Print the document serial number (INF) in the sequence defined by Spx32PutPrtSeq.
	4-7	Fixed message number (binary, 1 through F).
3 - 51		Variable Text for the 3891/XP and 3892/XP, in ASCII The variable text string must be terminated with a byte equal to X'7F'. This area is interrogated only when byte 2, bit 1 is on. The maximum allowed is 48 variable text characters on a single line. The maximum characters per document is 64.
3 - 69		Variable Text for the UT/XP The maximum allowed is 66 variable text characters on a single line.

Return Codes:

rc	*Description*
0	Successful.
1	The print line and side value in byte 1 are incorrect.
3	Variable text was specified, but no terminator was found.
4	A fixed message was requested but byte 2, bits 4 through 7 are equal to 0.
5	Line 2 or 3 was specified while requesting a large font.
17	The front side ink jet was requested but is not enabled.
18	The back side ink jet was requested but is not enabled.

100 This function is not supported on this document processor.

Notes:

1. You should ensure that the variable data does not exceed the maximum of 64 characters per document (including both front and back side printing). If you request more than 64 variable characters, the first 64 sent to the transport are used on the document; the remainder are ignored. On the UT/XP the maximum amount of text is 158 variable characters. The sequence of the endorsement lines sent to the transport is:
   - Front side line 1
   - Back side line 1
   - Front side line 2
   - Back side line 2
   - Front side line 3
   - Back side line 3.
2. The Control Program does not clear the endorsement lines between documents. The data is cleared only after Run initializations. To clear an endorsement line between documents, call the Spx32PutInkJetCtrl function with a buffer in which the first byte identifies the endorsement side and line being cleared, and the second byte contains hex 00.
3. If you specify the large ink jet font (IREC byte 64, bits 4 and 5), you can only specify one line per document side and that line must be line 1. You cannot specify large ink jet font on the UT/XP.
4. If you request fixed messages, you must first use Spx32LoadFixM to load the fixed message being requested.
5. The serial number (INF) can be on multiple lines.
6. The Spx32PutInkJetCtrl function has the following additional characteristics:
   - Ignores endorsement text defined by the IBM-supplied converge DLL.
   - Right justifies all printing to the currently defined position.
   - Truncates (without error indication) any fixed message or variable text that is too long to fit on the specified document.
   - Prints as spaces any hexadecimal ASCII codes that are not recognized.
   - Can be used at SPSINIT event time, allowing a sort program to establish an endorsement at a noncritical time and only make changes during critical processing time.
   - Allows any combination of variable text, fixed message, and serial number printing. Use Spx32PutPrtSeq to establish the left-to-right print sequence of this data.
   - Specifies only one line during each call.

Related Spx32 Functions

Spx32PutAltINFArea: This function stores data in the alternate INF flag and data area. This value is the same as the transport serial number and is used both in the endorsement and on the microfilm.

Spx32LoadFixM: This function allows you to specify fixed messages during the initialization process.

Spx32FixM: This function can be used for all fixed message endorsements as well as fixed message support for converged endorsements. If Spx32FixM and Spx32PutInkJetCtrl are used to specify the same line of endorsement, the transport returns an option state error and nothing is printed.

Spx32PutPrtSeq: This function allows you to change the sequence of the endorsement data elements:

- Fixed message
- Variable text
- Serial number or INF.

Incompatible Spx32 Functions

Spx32GetPrgEndDat: This function gets a copy of the programmable endorse data from the 3890 data area. On a 3891/XP or 3892/XP machine, this function only gets usable data when used with the converge DLL.

Spx32PutPrgEndDat: This function only handles valid data when used in conjunction with the converge DLL.

Spx32PutInptBufDat

Purpose: The Spx32PutInptBufDat function copies data to the input buffer.

Calling Sequence:

```
short _System Spx32PutInptBufDat(void *  );
unsigned char buffer[296];

rc = Spx32PutInptBufDat(buffer);
```

buffer
This is a 296-byte data area that contains the data to be copied.

Return Codes:

rc	*Description*
0	Successful.

Spx32PutISFFeatCtl

Purpose: The Spx32PutISFFeatCtl function copies the Image Scanner feature (ISF) control byte for the Image Capture System.

Calling Sequence:

```
short _System Spx32PutISFFeatCtl(unsigned char);
unsigned char byte;

rc = Spx32PutISFFeatCtl(byte);
```

byte
The ISF control byte. It is defined by initialization data byte 110.

Return Codes:

rc	*Description*
0	Successful.

Spx32PutNewArea

Purpose: The Spx32PutNewArea function copies up to 4K bytes of data to the New Save Area in additional AUX storage.

Calling Sequence:

```
short _System Spx32PutNewArea(void *  ,unsigned long,unsigned long);
unsigned char buffer[300];
unsigned long displacement;
unsigned long length;

rc = Spx32PutNewArea(buffer,displacement,length);
```

buffer
This is a data area from which the data is copied.

displacement
This is the displacement within the extended save area of the first byte that is to be copied. Specify 0 to begin the copy at the first byte.

length
This is the number of bytes that are copied.

Return Codes:

rc	*Description*
0	Successful.
1	Displacement out-of-bounds — (greater than 4095).
2	Displacement plus length is out-of-range — (greater than 4096).
3	Length is zero.

Spx32PutNoDateIns

Purpose: The Spx32PutNoDateIns function sets or clears the suppress-date-substitution-in-endorse-data flag.

If no further endorse changes are required after the first document is processed, you can set this flag to prevent unneeded processing of date substitution on subsequent documents.

Calling Sequence:

```
short _System Spx32PutNoDateIns(unsigned char);
unsigned char flag;

rc = Spx32PutNoDateIns(flag);
```

flag

This is a 1-byte data area. The following values apply:

Value	*Description*
0	No suppression; the date is substituted.
1	Suppression; the date is not substituted.

Return Codes:

rc	*Description*
0	Successful.

Spx32PutNoDWConv

Purpose: The Spx32PutNoDWConv function sets or clears the double-wide-endorse-data-conversion flag.

If no further endorse changes are required after the first document is processed, you can set this flag to prevent unneeded processing of double-wide conversion on subsequent documents.

Calling Sequence:

```
short _System Spx32PutNoDWConv(unsigned char);
unsigned char flag;

rc = Spx32PutNoDWConv(flag);
```

flag

This is a 1-byte data area. The following values apply:

Value	*Description*
0	No suppression; normal conversion.
1	Suppression; no conversion.

Return Codes:

rc	*Description*
0	Successful.

Spx32PutOCRSetup

Purpose: The Spx32PutOCRSetup function sets the OCR1 and OCR2 setup parameters. This function is supported on the 3891/XP and 3892/XP. The function defines up to two zones on a document for OCR reading and selects a font for each zone.

Calling Sequence:

```
short _System Spx32PutOCRSetup(void *  );
unsigned char buffer[11];

rc = Spx32PutOCRSetup(buffer);
```

buffer

This is an 11-byte data area that has the setup parameters.

ocrspec	1	OCR reader specifier 0 = OCR1 1 = OCR2
zone1def		zone 1 definition
start-tenths	2	zone start position (tenths component)
start-intg	3	zone start position (integer component)
fontspec	4	font specifier (See Note 6.)
stop-tenths	5	zone stop position (tenths component)
stop-intg	6	zone stop position (integer component)
zone2def		zone 2 definition
start-tenths	7	zone start position (tenths component)
start-intg	8	zone start position (integer component)
fontspec	9	font specifier (See Note 6.)
stop-tenths	10	zone stop position (tenths component)
stop-intg	11	zone stop position (integer component)

Return Codes:

rc	*Description*
0	Successful.
6	Invalid reader specified; not 0 or 1.
8	An invalid zone definition value was specified.
14	The requested font is not installed.
15	The zone specification is incorrect.
17	Feature not enabled.
100	This function is not supported on this document processor.

Notes:

1. The rightmost position (start) and the leftmost position (stop) of each zone are specified in inches and tenths of inches relative to document lead edge. Two bytes are used to specify a start or stop point with each byte being a Binary Coded Decimal (BCD) value from 00 to 09.
2. The function allows two zones to be defined. If only one zone is to be defined, it must be zone 1; and the zone 2 definition bytes, *zone2def*, must be all zeros.
3. The minimum zone 1 start position is 0.1 inches.

4. The font specifier, *fontspec*, must be a valid code as defined by the two tables below and must be installed on this document processor.
5. Once defined, the definition for a given OCR reader remains in effect until another Sort run is initialized.
6. A given OCR reader might have up to three numeric fonts installed but only one alphanumeric font installed. If an alphanumeric font is installed, no numeric fonts can be installed for that reader.

OCR NUMERIC FONTS

fontspec	**FONT**
hex 01	OCR-A Size 1, numeric
hex 02	E-13B
hex 03	407-1 or 407-E, whichever is installed
hex 04	7B
hex 05	OCR-B Size 1, numeric
hex 06	1428
hex 08	12F
hex 0D	1403

OCR ALPHANUMERIC FONTS

fontspec	**FONT**
hex 09	Numerics and symbols subset
hex 0A	Alphas and symbols subset (alpha only with OCR-B)
hex 0B	Full character set

Example

```
00  05  00  05  08  01  06  03  01  03  05
```

This buffer specifies that for OCR1:

Zone 1 starts 0.5 inches and stops 1.8 inches from the lead edge.
Zone 1 is to use OCR-B size 1, numeric font.

Zone 2 starts 3.6 inches and stops 5.3 inches from the lead edge.
Zone 2 is to use OCR-A size 1, numeric font.

Spx32PutOpCmdLine

Purpose: The Spx32PutOpCmdLine function copies data to the operator command line.

Calling Sequence:

```
short _System Spx32PutOpCmdLine(void *  );
char buffer[65];

rc = Spx32PutOpCmdLine(buffer);
```

buffer

This is a 65-byte data area that contains the data to be copied to the operator command line input area of the document processor. All 65 bytes of data are copied.

The data appears on the command line when control returns to the Run screen. The data is in ASCII format.

Return Codes:

rc	*Description*
0	Successful.

Spx32PutOpMsgNum

Purpose: The Spx32PutOpMsgNum function stores the Sort operator message number. This does the same function as the SCI "SOM" macro.

Calling Sequence:

```
short _System Spx32PutOpMsgNum(unsigned char);
unsigned char msgnum;

rc = Spx32PutOpMsgNum(msgnum);
```

msgnum

This is a 1-byte message number that is stored in the operator message number area of the document processor. To clear the message (that is, to indicate no message), set the message number to 0.

Return Codes:

rc	*Description*
0	Successful.

Note: The message does not appear until the document processor stops and the Run screen appears.

Spx32PutOpMsgTxt

Purpose: The Spx32PutOpMsgTxt function copies data to the Sort operator message (SOM) text area.

Calling Sequence:

```
short _System Spx32PutOpMsgTxt(void *  ,unsigned char);
char text[81];
unsigned char somnum;

rc = Spx32PutOpMsgTxt(text,somnum);
```

text

This is an 81-byte data area that contains the message that is copied to the operator message text area of the document processor. The first byte is the severity code and contains one of the following ASCII characters:

I	Information
W	Warning
E	Error.

somnum

This is the SOM number that specifies which message text to replace. The SOM number is in the range 1-255. The data is in ASCII format.

Return Codes:

rc	*Description*
0	Successful.
1	SOM number cannot be zero.
2	Severity level is not I, E, or W.

Notes:

1. This function changes the message text. To make the message appear, you should use the function Spx32PutOpMsgNum.
2. Initialization resets the SOM message text number.

Spx32PutPcktLim

Purpose: The Spx32PutPcktLim function copies data from the pocket definition table (PDT).

If the value that the Spx32GetPktTblUsed returns is not zero, then this table is being used to determine when a pocket reaches the merge limit.

Calling Sequence:

```
short _System Spx32PutPcktLim(void *  );
unsigned char buffer[72];

rc = Spx32PutPcktLim(buffer);
```

buffer
This a 72-byte data area that contains 36 entries in 16-bit Intel format.

Return Codes:

rc	*Description*
0	Successful.

Note: If you store a value greater than 4095 for any pocket, no merge documents are routed to that pocket.

Spx32PutPktCount

Purpose: The Spx32PutPktCount function copies data to the pocket counter area.

You should change bits 1, 2, or 3 only if you want to change the pocket status that these bits indicate. For more information about pocket counters, see the *3890/XP Enhanced Document Processor Programming Guide*.

Calling Sequence:

```
short _System Spx32PutPktCount(void *  );
unsigned char pktcntr[72];

rc = Spx32PutPktCount(pktcntr);
```

pktcntr
This is a 72-byte data area that contains 36 two-byte pocket counters in the SCI format.

Return Codes:

rc	*Description*
0	Successful.

Spx32PutPktTblUsed

Purpose: The Spx32PutPktTblUsed function sets the pocket limit flag.

Calling Sequence:

```
short _System Spx32PutPktTblUsed(unsigned char);
unsigned char flag;

rc = Spx32PutPktTblUsed(flag);
```

flag

This is a 1-byte data area that contains the value for the pocket limit flag. The following values apply:

hex 00 The initialization data is to determine the pocket limits.
hex 01 The pocket table is to determine the pocket limits.

Return Codes:

rc	*Description*
0	Successful.

Spx32PutPrgEndDat

Purpose: The Spx32PutPrgEndDat function copies data to the Active Endorsement Area (AEA).

Calling Sequence:

```
short _System Spx32PutPrgEndDat(void *  );
unsigned char endorsedata[72];

rc = Spx32PutPrgEndDat(endorsedata);
```

endorsedata
This is a 72-byte data area (3-byte by 24-byte array) that contains the endorsement data.

Return Codes:

rc	*Description*
0	Successful.

Notes:

1. Invalid characters will print as a "?" character. See the *3890/XP Enhanced Document Processor Programming Guide*.
2. If you use the "%" substitution parameter for the endorse date or the "+" for double wide characters, ensure that you did not disable substitution with the Spx32PutNoDateIns function or the Spx32PutNoDWConv function. After the first document is processed, the "+" ASCII characters are converted to double-wide characters (3890/XP) or to underscore characters (3891/XP and 3892/XP).

Spx32PutProcBufDat

Purpose: The Spx32PutProcBufDat function copies data to the process buffer.

Calling Sequence:

```
short _System Spx32PutProcBufDat(void *  );
unsigned char proc[244];

rc = Spx32PutProcBufDat(proc);
```

proc
This is a 244-byte data area that contains data to be copied to the process buffer.

Return Codes:

rc	*Description*
0	Successful.

Notes:

1. This function does not replace the process buffer header.
2. This function sets byte 7, bit 4 of the process buffer header.

Spx32PutProcBufHdr

Purpose: The Spx32PutProcBufHdr function copies data to the process buffer header.

Calling Sequence:

```
short _System Spx32PutProcBufHdr(void *  );
unsigned char buffer[12];

rc = Spx32PutProcBufHdr(buffer);
```

buffer

This is a 12-byte data area that contains the data to be copied to the process buffer header.

Return Codes:

rc	*Description*
0	Successful.

Note: Although 12 bytes of data are supplied, only a few of those bytes are used to update the process buffer header: header bytes 2, 3, 5, 6, and 8 (if logical merge mode is set), and byte 9.

Spx32PutProgStore

Purpose: The Spx32PutProgStore function copies data to program storage.

Note: Unlike the SCI instruction that performs the same function, Spx32PutProgStore ignores the 3890 emulation parameters that are set in the Run Profile.

Calling Sequence:

```
short _System Spx32PutProgStore(void *  ,unsigned long,unsigned long);
unsigned char buffer[300];
unsigned long disp;
unsigned long length;

rc = Spx32PutProgStore(buffer,disp,length);
```

buffer
This is the data area from which the data is written.

disp
This is a 32-bit displacement into program storage (Intel format).

length
This is the length, in bytes, to be written.

Return Codes:

rc	*Description*
0	Successful.
1	The displacement value is greater than program storage.
2	The displacement plus length is out-of-bounds.
3	The length is not greater than zero.

Notes:

1. If more than 65 535 bytes of data are to be written, use a series of calls.
2. In most cases, the values in the first 128 bytes of program storage cannot take effect until a new initialization occurs. The only data changes that are effective on succeeding documents with no reinitialization are merge feed control, merge feed pocket limits, and code line data match.

   For additional information, see Note 4 of the notes that follow the “Accessible Storage Map for SCI Load and Store” in Appendix C of the *3890/XP Enhanced Document Processor Programming Guide.*

Note: This SPX is included only for compatibility with existing sort programs. It is recommended that Spx32PutProgStoreFlat (see “Spx32PutProgStoreFlat” on page 3-130) be used in new sort programs, as Spx32PutProgStoreFlat does not have the 64K limit on length.

Spx32PutProgStoreFlat

Purpose: The Spx32PutProgStoreFlat function copies data from a specified buffer to program storage.

Calling Sequence:

```
short _System Spx32PutProgStoreFlat( void *, unsigned long, unsigned long );
unsigned char buffer[300];
unsigned long disp;
unsigned long length;

rc = Spx32PutProgStoreFlat(buffer,disp,length);
```

buffer
This is a data area from which this function copies data into program storage.

disp
This is a 32-bit displacement into program storage.

length
This is the length in bytes of the data to be copied.

Return Codes:

rc	*Description*
0	Successful
1	Displacement value greater than program store
2	Displacement plus length greater than end of program store.

Spx32PutPrtSeq

Purpose: The Spx32PutPrtSeq function allows the program to establish the sequence for printing the three endorsement data elements. The default order, from left to right, is:

- Fixed message
- Variable text
- Serial number or INF.

This function is supported on the 3891/XP, 3892/XP, and the UT/XP.

Calling Sequence:

```
short _System Spx32PutPrtSeq(void *  );
unsigned char buffer[6];

rc = Spx32PutPrtSeq(buffer);
```

buffer

This is a 6-byte data area in which each byte defines the print sequence for a given line on the document.

Byte number and corresponding endorsement line:

Byte 1 = Front side line 1
Byte 2 = Front side line 2
Byte 3 = Front side line 3
Byte 4 = Back side line 1
Byte 5 = Back side line 2
Byte 6 = Back side line 3

Hex value definition for each byte:

00 = Variable text, fixed message, serial number
01 = Variable text, serial number, fixed message
02 = Fixed message, variable text, serial number (default value)
03 = Fixed message, serial number, variable text
04 = Serial number, fixed message, variable text
05 = Serial number, variable text, fixed message
20 = Leave the current value unchanged or feature not configured.

Return Codes:

rc	*Description*
0	Successful.
1	The definition value specified is not hex 00 through 05 or hex 20.
4	The Spx32 was called during a document event.
17	The front side ink jet is not enabled.
18	The back side ink jet is not enabled.
100	This function is not supported on this document processor.

Notes:

1. You should ensure that the variable text does not exceed the permissible maximum of 64 characters per document (including both front and back side printing) on a 3891/XP or 3892/XP or a maximum of 158 characters per document on a UT/XP.
2. This Spx32 is supported only during non-document events.
3. If an endorsement data element has no data for a given line, a null character is assumed for that element for that line.

Spx32PutReinitLoadPath

Purpose:: Spx32PutReinitLoadPath sets the re-initialization load path.

Calling Sequence:

```
short _System Spx32PutReinitLoadPath(void *  );
char ReinitLoadPath[66];

rc = Spx32PutReinitLoadPath(ReinitLoadPath);
```

ReinitLoadPath
is the directory path to be used to load the run profile and associated files during re-initialization. The path can be up to 65 characters plus the null terminator (for a maximum length of 66 bytes). ReinitLoadPath must be a valid OS/2 drive-path specification.

Return Codes:

rc	Description
0	Successful.
1	Null terminator missing. No action taken.

Notes:

1. Use **Spx32PutReinitLoadPath** to change the path that will be used to load the run profile and associated files during re-initialization.
2. Use **Spx32PutReinitReq** to actually request re-initialization during the SPSINIT event.

Spx32PutReinitReq

Purpose: Spx32PutReinitReq sets the re-initialization request flag.

Calling Sequence:

```
short _System Spx32PutReinitReq(unsigned char);
unsigned char ReinitReq;

rc = Spx32PutReinitReq(ReinitReq);
```

ReinitReq
is a one-byte flag that requests re-initialization. Possible values are:

0 Re-initialization is not requested

1 Re-initialization is requested

Return Codes:

rc	Description
0	Successful.
1	Re-initialization in progress. No action taken.
2	Not initialization (SPSINIT) event. No action taken.

Notes:

1. Use **Spx32PutReinitReq** to request re-initialization during the SPSINIT event.
2. Re-initialization includes processing of the run profile and the run profile associated files. Re-initialization does not include processing of the initialization data (IREC), except for the name of the run profile.
3. Use **Spx32PutProgStore** to change the run profile (name is in the IREC) that will be used during re-initialization.
4. Use **Spx32PutReinitLoadPath** to change the path that will be used to load the run profile and associated files during re-initialization.

Spx32PutRqNotInit

Purpose: The Spx32PutRqNotInit function sets or clears the request-not-initialized flag.

This flag requests the Control Program to fail initialization for this run. The failure code will be hex 1F. Message 5031 will be displayed on the Run screen.

Calling Sequence:

```
short _System Spx32PutRqNotInit(unsigned char);
unsigned char flag;

rc = Spx32PutRqNotInit(flag);
```

flag

This is a 1-byte data area that contains the value of the flag. The following values apply:

hex 00 The flag is not set.
hex 01 The flag is set.

Return Codes:

rc	*Description*
0	Successful.

Spx32PutSavArea

Purpose: The Spx32PutSavArea function copies up to 256 bytes of data to the Save Area.

Calling Sequence:

```
short _System Spx32PutSavArea(void *  ,unsigned long,unsigned long);
unsigned char area[300];
unsigned long disp;
unsigned long length;

rc = Spx32PutSavArea(area,disp,length);
```

area
This is the data area from which the data is written.

disp
This is the displacement within the save area of the document processor where the data is stored.

length
This is the number of bytes to be written.

Return Codes:

rc	*Description*
0	Successful.
1	The displacement was out-of-bounds (greater than 255).
2	The displacement plus length was out-of-range (greater than 256).
3	The length was zero.

Spx32PutSCICount

Purpose: The Spx32PutSCICount function copies data to the SCI counters area.

Calling Sequence:

```
short _System Spx32PutSCICount(void *  );
unsigned char counter[32];

rc = Spx32PutSCICount(counter);
```

counter

This is a 32-byte data area that the calling program supplies. These are the 16 counters, which are 2-bytes each.

Return Codes:

rc	*Description*
0	Successful.

Spx32PutSEC

Purpose: The Spx32PutSEC function copies data to the symbol error correction value in emulated auxiliary storage.

Calling Sequence:

```
short _System Spx32PutSEC(unsigned char);
unsigned char secbyte;

rc = Spx32PutSEC(secbyte);
```

secbyte

This is the 1-byte data area that contains the symbol error correction value that this function copies to the emulated auxiliary storage. Any change to this value takes effect on the next document cycle. This is the same as initialization data byte 55.

Return Codes:

rc	*Description*
0	Successful.

Spx32PutSyncPause

Purpose: The Spx32PutSyncPause function sets the pause request for testing the syncronization between the Sorter and the Image Capture Processor.

This Spx is to be used in conjunction with the PAUSEXP.DLL supplied on the control program diskette.

Calling Sequence:

```
short _System Spx32PutSyncPause(void *  );
unsigned long buffer;

rc = Spx32PutSyncPause(&buffer);
```

buffer

- if value is equal to zero, then no pause is requested.
- if value is non-zero, a pause is requested immediately.

Return Codes:

rc	*Description*
0	Successful.

Spx32PutTimerSelect

Purpose: The Spx32PutTimerSelect function selects which events to time.

Calling Sequence:

```
short _System Spx32PutTimerSelect(void *  );
unsigned long timerselect;

rc = Spx32PutTimerSelect(&timerselect);
```

timerselect

This is a 16-bit data area in which this function returns the state indicating the currently selected events to time.

Return Codes:

rc	*Description*
0	Successful.

Bit definitions:

0	Run Initialization
1	Verify
2	Correction
3	Format
4	Code Line Data Match
5	Document
6	Post
7	Pause
8	Motor Stop
9	Operator

Spx32PutUsrStat

Purpose: The Spx32PutUsrStat function copies a value to the user statistics area.

Calling Sequence:

```
short _System Spx32PutUsrStat(unsigned char);
unsigned char buffer;

rc = Spx32PutUsrStat(&buffer);
```

buffer

This is a 1-byte data area that contains the value that this function copies to the user statistics area (user stats).

Return Codes:

rc	*Description*
0	Successful.

Spx32PutWrkReg

Purpose: The Spx32PutWrkReg function copies data to the work register. SCI programs use the work register as temporary storage to manipulate data.

Calling Sequence:

```
short _System Spx32PutWrkReg(void *  );
unsigned char buffer[16];

rc = Spx32PutWrkReg(buffer);
```

buffer
This is a 16-byte data area that the calling program supplies.

Return Codes:

rc	*Description*
0	Successful.

Spx32SetDocsToPause

Purpose: The Spx32SetDocsToPause function sets the value for the number of documents to process before a pause will be requested to test syncronization between the sorter and the Image Capture Processor. This value will be automatically used to reset the value after each pause request is honored. This spx only needs to be issued once per initialization. If this spx is not used, a default count value to 5000 documents will be established.

This Spx is to be used in conjunction with the PAUSEXP.DLL supplied on the control program diskette.

Calling Sequence:

```
short _System Spx32SetDocsToPause(void *  );
unsigned long buffer;

rc = Spx32SetDocsToPause(&buffer);
```

buffer

This is an unsigned long data area containing the documents to pause, count reset value. Vaild values are from 10 - 65535.

Return Codes:

rc	*Description*
0	Successful.
1	Value outside valid range.

Spx32SetHwnd

Purpose: The Spx32SetHwnd function tells the control program what the dialog's window handle is.

Calling Sequence:

```
short _System Spx32SetHWND( unsigned short, unsigned long );
unsigned short PMCounter;
unsigned long hwnd;

rc = Spx32SetHWND(PMCounter,hwnd);
```

PMCounter
This is a data area that contains the number of user dialogs active under PM. A maximum of 5 user dialogs may be active at any one time.

hwnd
This is the value of the window handle of the dialog.

Return Codes:

rc	*Description*
0	Successful
1	Invalid PMCounter

Note: It is necessary to tell the control program the window handle of the dialog. If the dialog execution time exceeds that specified in SPXNDPTIME (run profile) then the control program will send a WM_Close message to this handle. This enable the dialog to do any execution clean up required to terminate. If this routine is not called, and the SPXNDPTIME is exceeded, the control program will hang.

Spx32SinglePick

Purpose: The Spx32SinglePick function requests the transport to feed just one document. This function is only supported on the 3891/XP and the 3892/XP.

Spx32SinglePick sets a flag so the Control Program requests just one document from the transport. The call should be made at SPSINIT-time. Merge before main (MBM) is ignored, so that the single pick document comes from the main feeder.

If running offline from the host, the motors are stopped and control returns to the Run screen. If this function is called again at SPSDOC-time, the single pick flag is set for the next feed, so each press of the Feed key feeds just one document. Therefore, the user can single step through a set of documents and look at Sort program messages for each one, perhaps looking for a document that causes an error.

If running online to the host, the Control Program sets the Sort Pause Requested flag (header byte 9, bit 7). When the machine pauses, documents in the process buffer are flushed to the host and the Control Program then waits for "Host OK to Start" before requesting the next document. The host might download different initialization data at this point and therefore run a different job defined by special "beginning of job" documents.

The user Sort module can resume single pick, such as when another special control document is seen, making the call to Spx32SinglePick at SPSDOC-time. To allow for the fact that the machine does not pause immediately, there should be a number of these control documents. Up to 12 more documents are seen before the transport pauses or stops because of "documents in flight." These "end of job" documents can be followed by another "beginning of job" special document. The effect is that many different sort jobs can be run as a single operation at the machine.

Calling Sequence:

```
short _System Spx32SinglePick(void);

rc = Spx32SinglePick();
```

Return Codes:

rc	*Description*
0	Successful.
100	This function is not supported on this document processor.

Spx32SPC

Purpose: The Spx32SPC function sets the module-pocket in the process buffer header (byte 6) to the module-pocket (MP) code of the first (lowest-numbered) pocket that has reached or exceeded a predetermined value (set during initialization). This is the same as the SCI "SPC" macro.

The operation scans from 1/1 to 6/6. If no counter has reached the predetermined value, the module-pocket code is set to 1/1 (reject).

Calling Sequence:

```
short _System Spx32SPC(void);

rc = Spx32SPC();
```

Return Codes:

rc	*Description*
0	MP set for full counter
1	MP set for 1/1 (all counters low).

Note: If you use the Spx32PutProcBufHdr function after using this function, you can overwrite and cancel this function.

Spx32UsingPM

Purpose: The Spx32UsingPM function sets a variable used by the control program to determine the number of levels of user dialogs.

Calling Sequence:

```
short _System Spx32UsingPM( unsigned short * );
unsigned short PMCounter;

rc = Spx32UsingPM(&PMCounter);
```

PMCounter
This is a data area that contains the number of user dialogs active under PM. A maximum of 5 user dialogs may be active at any one time.

Return Codes:

rc	*Description*
0	Successful
1	Nesting exceeded

Using OS/2 Presentation Manager in Sort Programs

The 3890/XP Enhanced control program allows the programmer to write user interface code using the OS/2 Presentation Manager as an extension of the user interface code already present. This is a sample of code that does that:

```
#include <spxserv.h>   /* Include Header file for SPX functions  */
#define INCL_WIN       /* We want to have PM functions available */
#define INCL_DOS       /* We also want DOS functions available   */
#include <os2.h>       /* Include the requested Headers          */
#include <mydialog.h>  /* Also, include header file for Dialog   */

/* Prototype for MyDialogProc                                    */
MRESULT EXPENTRY MyDialogProc(HWND hwnd,
                              ULONG msg,
                              MPARAM mp1,
                              MPARAM mp2);

/* Prototype for MyDialogProc1                                   */
MRESULT EXPENTRY MyDialogProc1(HWND hwnd,
                               ULONG msg,
                               MPARAM mp1,
                               MPARAM mp2);

unsigned short MyPMCounter;

/* This routine is the routine specified in the Run Profile */
void My_PMScreen()
{

     /* First, load the DLL that contains the dialogs          */
     /* The dialogs are in MYDLGS.DLL                          */
     DosLoadModule(NULL,0,"MyDlgs",&DlgHandle);

     /* Now, tell the control program that we are using PM     */
     Spx32UsingPM(&MyPMCounter);

     /* Now, get the Window Handle for the Run Screen          */
     Spx32GetHwnd(&hwnd);

     /* Now, Run the Dialog                                    */
     rc = WinDlgBox(HWND_DESKTOP,
             hwnd,
             (PFNWP) MyDialogProc,
             DlgHandle,
             MY_DIALOG_ID,
             NULL);

     /* We're done, so free up the Dialogs                     */
     DosFreeModule(DlgHandle);

     /* Tell the control program that we have finished with PM */
     Spx32EndPM(MyPMCounter);
}


/* This is the Dialog Procedure controlling the dialog */
MRESULT EXPENTRY MyDialogProc(HWND hwnd,
                              ULONG msg,
                              MPARAM mp1,
                              MPARAM mp2)
static unsigned short MyPMCounter1;
```

```
{
     switch(msg)
     {
     case WM_COMMAND:
         switch (SHORT1FROMMP(mp1))
         {
         case DID_OK:
             /* The operator pressed the OK button */
             /* So any functions that need to be   */
             /* done at this time should be here   */
             WinDismissDlg(hwnd,DID_OK);
             break;

         case DID_CANCEL:
             /* The operator pressed the CANCEL    */
             /* button, so any functions that need */
             /* to be done should be here          */
             WinDismissDlg(hwnd,DID_CANCEL);
             break;

         case DID_MY_BUTTON:
             /* The operator pressed My_Button    */
             /* We now need to load another       */
             /* Dialog                            */
             Spx32UsingPM(&MyPMCounter1);
             rc = WinDlgBox(HWND_DESKTOP,
                            hwnd,
                            (PFNWP) MyDialogProc1,
                            DlgHandle,
                            MY_DIALOG_ID1,
                            NULL);
             Spx32EndPM(MyPMCounter1);
             break;

         }
     case WM_INITDLG:
         /* We need to tell the Control Program what */
         /* Window Handle we have                    */
         Spx32SetHwnd(MyPMCounter,hwnd);
         /* Do any initialization of the display here  */

         break;

     /* The following is optional.  It will only be   */
     /* used if the SPXNDPTIME is exceeded, or the    */
     /* Dialog is defined with a System Menu, and the */
     /* Exit option is selected...                    */
     case WM_CLOSE:
         /* Put any cleanup code here */
         WinDismissDlg(hwnd,DID_CANCEL);
         break;
     default:
         return(WinDefDlgProc(hwnd,msg,mp1,mp2));
     }
     return(FALSE);
}

/* This is the Secondary Dialog Procedure */
MRESULT EXPENTRY MyDialogProc1(HWND hwnd,
                               ULONG msg,
                               MPARAM mp1,
                               MPARAM mp2)
{
     switch(msg)
```

```
                    {
                    case WM_COMMAND:
                        switch (SHORT1FROMMP(mp1))
                        {
                        case DID_OK:
                            /* The operator pressed the OK button */
                            /* So any functions that need to be   */
                            /* done at this time should be here   */
                            WinDismissDlg(hwnd,DID_OK);
                            break;

                        case DID_CANCEL:
                            /* The operator pressed the CANCEL    */
                            /* button, so any functions that need */
                            /* to be done should be here          */
                            WinDismissDlg(hwnd,DID_CANCEL);
                            break;

                        }
                    case WM_INITDLG:
                        /* We need to tell the Control Program what */
                        /* Window Handle we have                    */
                        Spx32SetHwnd(MyPMCounter,hwnd);
                        /* Do any initialization of the display here  */

                        break;

                    /* The following is optional.  It will only be   */
                    /* used if the SPXNDPTIME is exceeded, or the    */
                    /* Dialog is defined with a System Menu, and the */
                    /* Exit option is selected...                    */
                    case WM_CLOSE:
                        /* Put any cleanup code here */
                        WinDismissDlg(hwnd,DID_CANCEL);
                        break;
                    default:
                        return(WinDefDlgProc(hwnd,msg,mp1,mp2));
                    }
                    return(FALSE);
                }
```

Glossary

This glossary defines terms that are used in this manual and in other books in the XP Series library. This glossary also includes tems and definitions from the *IBM Dictionary of Computing*, SC20-1699. If you do not find the term that you want, see the index.

A

AEA. Active Endorsement Area.

alternate INF data. A five-byte field in AUX storage that is printed on the back of an item or document instead of the INF number when the alternate INF indicator in AUX storage is set to X'01' (on). You must specify ten digits, two for each byte. Each half-byte has a value range of X'0-A'. X'A' prints as a blank.

autoselect. The document is sent to module/pocket 1/1 because of hardware-detected errors.

C

CCW. See *channel command word*.

channel command word (CCW). A System/370* word interpreted by the channel as an instruction in the channel program. One or more CCWs make up the channel program. That channel program directs channel operations.

code line data matching. The process of matching data in a file to the code line data on a document.

D

default. An alternative value, attribute, or option that is assumed when none has been specified.

disk subsystem. System facilities that are identified by OS/2 drive and path conventions.

display panel. A predefined display image that can contain information such as instructions, prompts, options, and data.

display screen. An illuminated display surface on which display images are presented.

DLL. See *dynamic-link library*.

dynamic-link library (DLL). A specially-linked collection of Sort modules, loaded by OS/2 facilities, that can be called during the running of Sort programs. The OS/2 loader automatically loads additional DLL files that are needed to resolve labels in the DLL file that is being loaded. The file named in the Run Profile (USERDLL keyword) is of this type.

E

ECP. See *endorse control product*.

EIM. See *extended image match*. Also see *extended code line data match*.

endorse control product (ECP). This product aids the Sort programmer in controlling the endorse feature for each document.

extended code line data match. An operation that permits code line data matching to be done under the control of the SCI program.

extended image match (EIM). See *extended code line data match*.

I

IBM format. The high-order byte of an integer value is stored in the low address of the field; therefore, the integer value in memory appears the same as its hexadecimal format. See *Intel format*.

```
Decimal format:      1234
Hexadecimal format:  04D2
IBM format:          04D2
```

image matching. A term used in the past to describe the process of matching data in a file to the codeline data on a document. See *code line data matching*.

IML. See *initial microcode load*.

INF. Item Number Feature.

INF number. An eight- or ten-digit number that is printed on the back of each item or document that the 3890/XP processes if the alternate INF indicator in AUX storage is set to X'00' (off). The Sort program can add one to either the low-order segment or the high-order segment of this number after each document is processed.

---

* Trademark of IBM

initial microcode load (IML). This is the event that loads the 3890/XP control program. This event must occur after every power transition of the 3890/XP.

initialization data. The first 128 bytes of program storage. The initialization data defines the functions of the 3890/XP. It is extended by the run profile, which can be named in the initialization data.

initialization-record macro (IREC macro). A host-system macro that can be used to format the initialization data.

Intel format. The low-order byte of an integer value is stored in the low address of the field. Therefore, the integer value in memory appears to be the reverse of its hexadecimal format. See *IBM format*.

```
Decimal format:      1234
Hexadecimal format:  04D2
Intel format:        D204
```

IREC macro. See *initialization-record macro*.

item number. See *INF number*.

K

K byte. See *kilobyte*.

kilobyte (K byte). 1 K byte = 1024 bytes.

L

LCL. See *level-control label*.

level-control label (LCL). A 16-character string carried in every run-profile element that can be used to check the change level of that element.

LU 6.2 application-program interface (LAPI). Macros and modules that supply support for the host application to communicate with other application programs using LU 6.2 protocol.

M

M byte. See *megabyte*.

machine profile. This file contains parameters defined by the user describing the specific XP machine, such as the System/370 channel address. It is loaded with the XP Control Program from the disk subsystem. You can use the customization display panels in the XP user interface to maintain this file.

megabyte (M byte). 1 M byte = 1 048 576 bytes.

message line. The area of the display screen where messages appear.

MICR. Magnetic Ink Recognition.

N

native language. Any language that works with the OS/2 system.

O

OCR. Optical Character Recognition.

Operating System/2 (OS/2). The operating system for the IBM PS/2 computers.

OS/2. See *Operating System/2*.

P

panel. See *display panel*.

program-storage-data file (PSD). A file on the disk subsystem that is identified in the run profile that is loaded without change into program storage during initialization. PSD also refers to the name and path specification of this type of file. You use a PSD to load a table into the sorter; a run profile might include a PSD entry.

PSD. See *program-storage-data file*.

PSD table. Run-profile-element table. A table that the 3890/XP control program builds for the SCI program and OS/2 language Sort programs. The table points to locations in storage of data that is loaded from files during initialization. Such files, identified in the run profile, are run-profile elements.

PSD tag. A tag that is independent from the file name and that is carried in the PSD keyword of the run profile (PGMSTOREdata) and is available to the SCI and OS/2 language Sort programs through the PSD table. Sort programs use this tag to identify and find PSD files by type.

Q

quick stop. The hardware-initiated event that stops the transport immediately when a jam or other problem is detected. The operator clears the affected module, then uses a runout to permit trapped documents that are still in the transport to be processed. See *runout*.

R

Run Profile. An ASCII keyword file that controls operation during initialization. This file is available to the XP Control Program through the disk subsystem. If no Run Profile is named in the initialization data, the default Run Profile is used. The Machine Profile names the default Run Profile, and the drive and path for all Run Profiles.

runout. The hardware-controlled operation that the operator uses during recovery from a quick stop. It permits items that were trapped by the quick stop but not removed by the operator to be processed. During runout, the SCI program runs and creates read records, even though all documents that are read during runout appear blank. A jam can also occur during runout.

S

SCI. See *Stacker Control Instruction*.

SIO. See *start I/O*.

Stacker Control Instruction (SCI). SCI is a language that is interpreted by the 3890/XP as the Sort program. 3890/XP offers major extensions, including four-byte addressing and native-language subroutines.

sort module. See *sort-program module*.

Sort program. All the user-supplied code for SPS events. In the Run Profile, the Sort program for each SPS event is specified by an ordered series of SPS keywords. Once it is called, a sort-program module can call other sort-program modules. One special sort-program module is the IBM-supplied SCID, which runs the SCI program that is stored in the program-store area. Like any sort-program module, SCID can be specified in the Run Profile for any of the SPS events, most usually for SPSDOC.

Sort-program module. A routine which is part of a Sort program. Sort modules are entry points within DLL files. You can specify Sort modules to be run as part of Sort programs in the following ways:

- Name the sort modules in an SPSxxx keyword in the run profile.
- Name the sort module in the machine profile.
- Call the sort module directly from another sort module.

Sort-program-sequence (SPS). Sort-program-sequence keyword in the run profile. This keyword specifies the components of the Sort program for a specific SPS event. This includes the native-language subroutine (if any), the SCI program (if used), and the sequence of their processing. A run-profile can contain an SPS keyword for each SPS event.

special-symbol-sequence table (SST). This table defines the code line for 3890/XP documents.

SPS. See *sort-program-sequence*.

SPS event. An event that causes a Sort program module to run. SPS events include the following:

- Non-document SPS events:
  - Initialize
  - Motors stopped
  - Operator.
- Document SPS events:
  - Verify
  - Correction
  - Format
  - Image match
  - Document
  - Post sort.

SPXSERV.DLL. See *SPX services dynamic-link library*.

SPX services dynamic-link library (SPXSERV.DLL). This library comes with the 3890/XP control program. These subroutines are the lowest-level interface for the OS/2 language programmer to 3890/XP services.

SST. See *special-symbol-sequence table*.

start I/O (SIO). The System/370 instruction that starts all I/O operations by requesting a channel program.

T

Terminate Stacker Control Instructions (TSCT). An interrupting condition that is forced by the system when the Sort program has taken too much time. The TSCI causes the late module/pocket code to occur.

TSCI. See *Terminate Stacker Control Instructions*.

Index

H

I

J

L

M

N

O

P

R

S

U

Communicating Your Comments to IBM

3890/XP Series and 3890/XP Enhanced
SPXServ Reference

Publication No. SC31-4070-00

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:

  United States & Canada: 1-800-955-5259
- If you prefer to send comments electronically, use this network ID:

  IBM Mail Exchange: USIB1362 at IBMMAIL

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies.

Readers' Comments — We'd Like to Hear from You

3890/XP Series and 3890/XP Enhanced
SPXServ Reference

Publication No. SC31-4070-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	□	□	□	□	□

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	□	□	□	□	□
Complete	□	□	□	□	□
Easy to find	□	□	□	□	□
Easy to understand	□	□	□	□	□
Well organized	□	□	□	□	□
Applicable to your tasks	□	□	□	□	□

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? □ Yes □ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SC31-4070-00

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Payment Solutions
Department 58G MG34/204
8501 IBM Drive
Charlotte NC 28262-8563

Fold and Tape | **Please do not staple** | Fold and Tape

SC31-4070-00

Cut or Fold
Along Line

File Number
S/370-4300-40

SC31-4070-00

Printed in U.S.A.