EXTERNAL REFERENCE SPECIFICATION

for

Input Output Control (IOC) Interface

Submitted:	W. R. Bayer
Approved:	J. J. Krautbauer
	P. W. Haynes
	J. R. Ruble

REVISION DEFINITION SHEET

REV	DATE	DESCRIPTION
1	02/05/79	Original Release.
2	03/05/79	Updates including definition of iot$line, clarification of procedures and types.
3	03/12/79	Updates to insert *CALL of common decks to allow dynamic update of document.
4	10/31/79	Updates to reflect code implementation.
5	02/18/80	SES Release 13 update.
6	07/16/80	Document format changes. Obsoletes all previous versions.

1.0 INTRODUCTION

The programming language used in this implementation is CDC CYBIL Extended. The details of the interface are defined in terms of CYBIL structures.

1.1 SCOPE OF DOCUMENT

This document is one of a set of documents describing portions of the interfaces which are part of the SES Utility Library. A seperation has been made to simplify documentation efforts and to exemplify the natural modularity.

This document contains information necessary for the understanding and use of the Input Output Control (IOC) available through Software Engineering Services Utility Library (SESUL).

1.2 ASSOCIATED DOCUMENTS

The following documents may be referenced in part to obtain a more complete understanding of the origin, uses and nomenclature associated with Input Output Control.

NOS/VE ERS

Language Specification for CDC CYBIL (ARH2298)

ERS for CYBILIO (ARH2739)

Cyber 180 System Interface Standard (S2196)

SES Procedure Writers Guide (ARH2894)

SES User's Handbook (ARH1833)

Message Generator ERS (SES Internal)

Miscellaneous Routines ERS (SES Internal)

Command Processor (CP) ERS (SES Internal)

System Command Language (SCL) ERS (SES Internal)

SES Processor ERS (SES Internal)

1.3 NAMING CONVENTIONS

The following naming conventions have been imposed upon the IOC and reflect upon the SES Utility Library interface routines in general.
Decknames are of the form Zpcyxxx where:

Z	universal SES identifier
pc	two character interface identifier IO Input Output Control
y	type of deck I Compass module (Ident) P CYBIL procedure reference C CYBIL constant declaration T CYBIL TYPE declaration V CYBIL variable declaration M CYBIL module F CYBIL function
xxx	three characters representing the abbreviated descriptive name of the deck (suggestion is first characters of the words composing the descriptive name).

1.4 IOC_USAGE

All procedures described in this document are available for use. Common decks are made available by specifying the "CYBCCMN" keyword on the SES.GENCOMP procedure. The binaries are available for linking by specifying the "CYBCLIB" keyword on the SES.LINK170 procedure.

Note that more dcurrent copes of IOC routines are generally made available in the "SSS" catalog as a function of pre-release while a release version is found on the "SES" catalog. A complete summary of decks updated during pre-release may be obtained from your local SES representative.

2.0 IOC DESCRIPTION

IOC provides the user an input/output interface to logically concatenate input from multiple sources and distribute output to multiple destinations. The purpose of the IOC interface is to centralize the control of input and output streams to and from user command programs, so that processing is invariant with the mode of access; e.g., local batch, remote batch or interactive. IOC handles only text type data and works with CYBILIO legible files.

2.1 OBJECTIVES_OF_IOC

The objectives of the IOC interface are:

(1) centralize control of input and output for SES Utility interfaces and

(2) make the I/O processing invariant with the mode of access.

2.2 PHILOSOPHY OF IOC

IOC is a high level interface to control I/O to CYBILIO legible files. There are two sides to the IOC interface.

Input is controlled through the **iov$input_stack**. Entry one is associated with standard input. Entries 2 .. n are associated with command file (alternate) input sources opened during program operation.

Output is controlled through the **iov$ioc_stream_table**. It is composed of a list of files and a matrix of booleans indicating 8 possible stream connections to the 8 files. The file list contains control block information for the files. A table is kept for stream names and ordinal association. Variables **iov$input_stack**, **iov$ioc_stream_table** and **iov$stream_map** are externally declared and initialized by **iop$ioc_initialize**.

Basic procedure interfaces are provided to connect/disconnect streams and files, open/close input files, get/put data, and position input files.

Note that automatic buffer flushing is done for all CYBILIO files of terminal origin. It is expected that **iop$ioc_terminate** is used to terminate IOC operations by closing IOC files opened by initialization.

3.0 IOC INTERFACES

3.1 DESCRIPTION

The IOC interfaces are composed of the declarations of the Input Stack, Stream Map and Stream Table along with procedure interfaces to control I/O using these structures.

3.2 DATA_FLOW_DIAGRAM_OF_INPUT_OUTPUT_CONTROL

3.3 DATA_STRUCTURES

3.3.1 GENERAL STRUCTURES

The common decks presented contain information which forms the basis for the higher level structures in the next sections.

```
{ ZIOTLNG    Definition of command file line limitations. }

  CONST
    ioc$max_line_number = 1000;

  TYPE
    iot$line_range = 0 .. ioc$max_line_number;



{ ZIOCSOS    Definition of the  input stack size. }

  CONST
    ioc$size_of_input_stack = 64;



{ ZIOCMSR    Definition of input stack limitations. }

  CONST
    ioc$max_stack_entry = 64;



{ ZIOCIOM    Definition of stream table dimensions. }

  CONST
    ioc$max_stream = 8,
    ioc$max_file = 8;
```

```
 { ZIOCSTM    Name constants for the IOC Environment. }

   CONST
     out_file = 'output',
     log_file = 'seslog',
     cond_stream = 'conditional_output',
     di_stream = 'di_output',
     log_stream = 'diagnostic',
     std_stream = 'standard',
     alt_stream = 'alternate';


 { ZIOCSTO    Definition of input stack ordinals. }

   CONST
     ioc$standard = 1,
     ioc$alternate = 2;
```

3.3.2 IOC LINE

A basic structure used throughout IOC is the line.

```
*callc zoststr

{ ZIOTIOL    Definition of the iot$line structure. }

  TYPE
    iot$line = record
      index: ost$string_index,
      length: ost$string_length,
      text: string (osc$max_string_length),
    recend;
```

index	current position pointer within the text string
length	length of text string
text	buffer containing the string of characters which compose a line

3.3.3 INPUT STACK

The Input Stack provides a structure to handle input sources to a program. The first entry of the stack is reserved for standard input. Entries 2 through ioc$size_of_stack are used for alternative input sources. On exhaustion of input or exit from that source the former frame is restored.

```
*callc ziocsto
*callc pxiotyp
*callc ziotlng
*callc zostnam
*callc ziotiol

{ ZIOTSKF    Definition of the input stack frame. }

  TYPE
    iot$stack_frame = record
      pointer_to_file: file,
      file_name: ost$nos170_name,
      line_number: iot$line_range,
      line_contents: iot$line,
    recend;
```

pointer_to_file	pointer to cell which will contain address of file control block
file_name	name of file
line_number	the line number of file that is currently being processed
line_contents	the line that is currently being processed

The variable input stack common deck is found on deck ZIOVSTK.

```
 *callc ziocmsr
 *callc ziocsos

 { ZIOVSTK    Declaration of the input stack structure. }

   VAR
     iov$input_stack: [XREF] record
       size_of_stack: 0 .. ioc$max_stack_entry,
       entry_in_use: 0 .. ioc$max_stack_entry,
       frame: array[1 .. ioc$size_of_input_stack] of iot$stack_frame,
     recend;
```

size_of_stack	top of the input stack (equivalent to the number of entries)
entry_in_use	the current entry in use in the stack
frame	file description information

3.3.4 STREAM TABLE

The output stream table provides a means of outputting to more than one file with a single procedure call. A stream may be associated with up to 8 files. This type is available on deck ZIOTISM.

```
*callc ziociom
*callc zostnam

{ ZIOTISM    Definition of the output stream table structure. }

  TYPE
    iot$stream_table = record
      connects: packed array[1 .. ioc$max_stream] of packed array[1 ..
        ioc$max_file] of boolean,
      list: array[1 .. ioc$max_file] of record
        pointer_to_file: file,
        file_name: ost$nos170_name,
        connect_count: 0 .. ioc$max_stream,
      recend,
    recend;
```

connects	this field contains a boolean map representing the connections currently established. The rows represent possible streams and the columns represent possible connections for a stream. TRUE indicates an established connection FALSE indicates the absence of a connection
list	this field contains the files currently connected to streams. The file control block and file name are maintained for each file connected together with a count indicating the number of streams to which the file is currently connected.

3.0 IOC INTERFACES
3.3.4 STREAM TABLE

The Stream Table is available as a variable in deck ZIOVISM.

```
 { ZIOVISM    Declaration of the stream table. variable }

   VAR
     {output control}
     iov$ioc_stream_table: [XREF] iot$stream_table;
```

3.3.5 STREAM MAP

The **iov$stream_map** is used to relate the stream names to the relative ordinal positions in the Stream Table.

```
{ ZIOVSTM    Declaration of the stream map variable. }

   VAR
     iov$stream_map: [XREF] array[1 .. ioc$max_stream] of ost$name;
```

The array of stream names is established by the connect procedure. When an empty entry or the name is found in the array, its position becomes the stream ordinal. The **connects** field in the **iov$ioc_stream_table** is set TRUE for that file. A disconnect sets the **connects** field false for that file.

3.4 PROCEDURES

3.4.1 INITIALIZE IOC ENVIRONMENT

The purpose of this procedure is to initialize the IOC environment. This is executed once per program and must be executed before any of the following procedures may be used. The **iov$input_stack**, **iov$ioc_stream_table**, and **iov$stream_map** are initialized and some basic stream connections are made. For general purpose use, the initialization handles any stream-file connections necessary for IOC operation.

```
{ ZIOPINI    Initialize the IOC environment. }

  PROCEDURE [XREF] iop$ioc_initialize ALIAS 'ziopini' (VAR status:
    ost$status);


    status          variable into which status is returned
```

3.4.2 CONNECT FILE/STREAM

The purpose of this procedure is to establish a connection between a file and a stream for output use. When this procedure is called the map field of the stream connection is altered to indicate the connection. The file is physically opened for output. Note that the default character set on open is ascii612#.

```
{ ZIOPCFS    Connect the file to a stream. }

  PROCEDURE [XREF] iop$connect_file_to_stream ALIAS 'ziopcfs' (file_name:
    string ( * );
    stream: string ( * );
    VAR status: ost$status);
```

file_name	specifies the name of the file to be connected. If the file does not exist an error condition will result.
stream	the name of the stream to be connected
status	variable into which status is to be returned

3.4.3 DISCONNECT FILE/STREAM

The purpose of this procedure is to sever the output connection between a file and a stream. The stream connection table is updated and the file is physically closed if no other streams are connected to it.

```
{ ZIOPDFS    Disconnect the file from the stream. }

  PROCEDURE [XREF] iop$disconnect_file_from_stream ALIAS 'ziopdfs'
    (file_name: string ( * );
    stream: string ( * );
    VAR status: ost$status);
```

file_name	name of file to be disconnected
stream	name of stream to be disconnected
status	variable into which status is to be returned

3.4.4 OPEN INPUT FILE

The purpose of this procedure is to open a file for input. The file specified becomes the current input file. Its description is added to the input control stack and the file is physically opened for input. Note that default character set on open is ascii612#. The mode may be altered by iop$codeset.

This procedure saves the line_contents of the current input file in the current stack frame before opening the new file. The current stack frame is updated for recall before creating a new stack frame. When a source is exhausted the stack is collapsed, the saved text and position restored. This operation occurs during the iop$close_current_input_file.

```
*callc zostnam
*callc ziotlng
*callc ziotiol

{ ZIOPOIF    Open an input file. }


  PROCEDURE [XREF] iop$open_input_file ALIAS 'ziopoif' (line_contents:
    iot$line;
    file_name: ost$nos170_name;
    VAR status: ost$status);
```

line_contents	the current source line to be saved from the stack file. This enables the text processing to continue when the stack is collapsed back to this point. Note that passing a null line preserves the stack frame buffer contents.
file_name	the name of file to be opened. If the file does not exist, an error condition results.
status	variable into which status is returned

3.4.5 CLOSE CURRENT INPUT FILE

The purpose of this procedure is to close the current input file. When a file is closed by this procedure its description is removed from the input control stack and the file is physically closed (unlesss it is used at a lower level in the stack).

```
{ ZIOPCIF    Close the current input file. }

  PROCEDURE [XREF] iop$close_current_input_file ALIAS 'ziopcif' (status:
    ost$status);
```

```
    status          the variable into which status is to be returned
```

3.4.6 SET FILE CHARACTER SET

The purpose of this procedure is to set the character set of a legible file which has been opened by an iop$open_input_file or a iop$connect_file_to_stream. Note that for a connect this should be executed only once, at the time of the first connection of the file and a stream. Care should be taken not to attempt to change character sets during the course of operations. The default used on all CYBILIO opens is ascii612#.

```
{ ZIOPCDS    Change the character code set of a file. }

  PROCEDURE [XREF] iop$codeset ALIAS 'ziopcds' (file_name: string ( * );
    codeset: file_encoding;
    VAR status: ost$status);
```

file_name	specifies name of file to be set. If the file doesn't exist an error condition results.
codeset	corresponds to CYBILIO codeset: ascii64# 6 bit display code character set ascii612# NOS 6/12 character set ascii# 8 of 12 ASCII character set
status	variable into which status is returned

3.4.7 GET COMMAND LINE

The purpose of this procedure is to get a command line from the current input source. Current source may be Standard Input or the alternate input source. Each input line is examined to eliminate trailing blanks. If an ellipsis is found as the last two characters of the line continuation is processed. The text line is built until 256 characters are processed or no continuation is found. In the event the 256 character limit is exceeded abnormal status is set and input is flushed until no continuation is found. When a command occupies only a portion of the current text line a boolean is set true. This indicates a user should repeat the get operation to retrieve remaining command text in the current line being processed. This may be done after the current command is processed and prior to taking any action which would alter stack contents. Multiple commands per line assume a semicolon separator. It is the responsibility of the user to process the error status as well as determine the action upon the input source.

```
{ ZIOPGCL    Get the next command line. }

  PROCEDURE [XREF] iop$get_command_line ALIAS 'ziopgcl' (VAR text:
    iot$line;
    VAR text_remaining_in_line: boolean;
    VAR status: ost$status);
```

text	variable into which next command line is returned
text_remaining_in_line	boolean value is true if current text line contains more commands else false if current text line exhausted
status	variable into which status is returned

3.4.8 GET COMMAND TEXT

The purpose of this procedure is to get command text from the current input source. This interface gives the user the ability to process a command greater than 256 characters. Current source may be Standard Input or the alternate input source. Each input line is examined to eliminate trailing blanks. If an ellipsis is found as the last two characters of the line continuation is flagged via **end_of_command** set to FALSE and **end_of_input_line** set FALSE. Multiple commands per line assume a semicolon separator. It is the responsibility of the user to process the error status as well as determine the action upon the input source. The following table explains actions to be taken on the text line.

e_o_i_l	e_o_c	user_action
0	0	process current partial command, repeat call
0	1	process current command, repeat call
1	0	process current command
1	1	process current command

```
*callc ziotiol
*callc osdstat

{ ZIOPGCT    Get command text }

  PROCEDURE [XREF] iop$get_command_text ALIAS 'ziopgct' (VAR text:
    iot$line;
    VAR end_of_command: boolean;
    VAR end_of_input_line: boolean;
    VAR status: ost$status);
```

text	variable into which command text is returned
end_of_command	boolean, true when semi-colon or end of input line indicates end of command , false when continuation of being processed.
end_of_input_line	the current input line is completely processed (line text exhausted)
status	variable into which status of this operation is returned

3.4.9 GET FROM STANDARD INPUT

The purpose of this procedure is to get the next line from the Standard Input File.

```
{ ZIOPGSI    Get the next line of input from the standard input file. }

  PROCEDURE [XREF] iop$get_standard_input ALIAS 'ziopgsi' (VAR text:
    iot$line;
    VAR status: ost$status);


    text           variable into which the next line is returned (note the
                   line may contain several commands)

    status         variable into which status is returned
```

3.4.10 GET FROM CURRENT INPUT

The purpose of this procedure is to get the next record from the current (alternate or standard source) input file.

```
{ ZIOPGCI    Get the next line of input from the current input file. }

   PROCEDURE [XREF] iop$get_current_input ALIAS 'ziopgci' (VAR text:
     iot$line;
     VAR status: ost$status);
```

text	variable into which the next line is returned (note the line may contain several commands)
status	variable into which status is returned

3.4.11 PUT TO STREAM

The purpose of this procedure is to output text. When text is output by this procedure it is written on all files connected to the stream specified.

```
*callc osdstat

{ ZIOPPTS    Send an output string to the IOC stream. }

  PROCEDURE [XREF] iop$put_to_stream ALIAS 'zioppts' (stream: string ( * );
    output_text: string ( * );
    VAR status: ost$status);
```

stream	the name of the stream to which the text is to be output
output_text	the buffer from which the text is written
status	variable into which the status is to be returned

3.4.12 GET LINE POSITION OF CURRENT INPUT FILE

The purpose of this procedure is to get the line number of the current line in the current input file.

```
{ ZIOPGPN    Get the line no. of the current line in the current input
{file. }

  PROCEDURE [XREF] iop$get_position ALIAS 'ziopgpn' (VAR line_number:
    iot$line_range;
    VAR status: ost$status);
```

line_number	name of the variable into which the line number is to be returned. The position returned is the line number of the last line obtained from the file.
status	variable into which status is to be returned

3.4.13 POSITION CURRENT INPUT FILE TO LINE

The purpose of this procedure is to set the current input file to the specified line number.

```
{ ZIOPSPN    Set the current input file to the specified line number. }

  PROCEDURE [XREF] iop$set_position ALIAS 'ziopspn' (line_number:
    iot$line_range;
    VAR status: ost$status);
```

line_number	the line number to which the file is to be set
status	variable into which the status is to be returned

3.4.14 CURRENT INPUT FILE IS A TERMINAL?

The purpose of this procedure is to determine if the current input file is of terminal origin.

```
{ ZIOPFIT    Determine if the file is a terminal. }

   PROCEDURE [XREF] iop$current_input_file_terminal ALIAS 'ziopfit' (VAR
     file_is_a_terminal: boolean);
```

file_is_a_terminal	boolean set true if current file at top of stack is of terminal origin else set false

3.4.15 CURRENT INPUT FILE MARK

This procedure determines the CYBILIO file mark of the current input file.

```
{ ZIOPFLM    Determine the present mark for the file. }

  PROCEDURE [XREF] iop$current_input_file_mark ALIAS 'ziopflm' (VAR mark:
    file_mark);
```

mark	the CYBILIO file mark of current input source

3.4.16 TERMINATE IOC OPERATION

This procedure terminates operation of the IOC interfaces. The streams connected at initialization are disconnected closing output files and the input stack is collapsed closing any remaining input files. This should be one of the last steps in a program using IOC. Note that any connects made in addition to initialization should be disconnected seperately.

```
*callc osdstat

{ ZIOPTER    Procedure to terminate IOC operations. }

  PROCEDURE [XREF] iop$ioc_terminate ALIAS 'ziopter' (VAR status:
    ost$status);


    status          variable which contains status of operation
```

3.5 ERROR_CONDITIONS

The following is a list of condition codes associated with IOC.

```
{ ZIOCECO    Error codes for the IOC Procedures. }

   CONST
     ioc$invalid_open = 4000 + 1,
     ioc$invalid_close = 4000 + 2,
     ioc$non_positionable = 4000 + 3,
     ioc$end_of_file = 4000 + 4,
     ioc$not_in_stream_map = 4000 + 5,
     ioc$stream_table_full = 4000 + 6,
     ioc$file_not_found = 4000 + 7,
     ioc$acquire_problem = 4000 + 8,
     ioc$size_exceeded = 4000 + 9,
     ioc$token_unknown = 4000 + 10,
     ioc$command_too_large = 4000 + 11,
     ioc$stack_exhausted = 4000 + 12;
```

The following are common decks ZIOVMTO and ZIONMTO containing information used in the construction of the osv$template_array. These decks are added to the template array by the SES.GENMAR procedure by including a product code of 'IO'.

```
 ?? fmt ( format := off ) ??

    VAR

      osv$template_4001 : [static] string (35) :=
        '+ INVALID OPEN ATTEMPTED ON FILE +P',

      osv$template_4002 : [static] string (36) :=
        '+ INVALID CLOSE ATTEMPTED ON FILE +P',

      osv$template_4003 : [static] string (29) :=
        '+ FILE +P IS NON-POSITIONABLE',

      osv$template_4004 : [static] string (37) :=
        '+ END OF FILE ENCOUNTERED FOR FILE +P',

      osv$template_4005 : [static] string (37) :=
        '+ ENTRY FOR FILE +P NOT IN STREAM MAP',
      osv$template_4006 : [static] string (22) :=
        '+ STREAM TABLE IS FULL',

      osv$template_4007 : [static] string (19) :=
        '+ FILE +P NOT FOUND',

      osv$template_4008 : [static] string (30) :=
        '+ THERE WAS AN ACQUIRE PROBLEM',

      osv$template_4009 : [static] string (21) :=
        '+ TABLE SIZE EXCEEDED',

      osv$template_4010 : [static] string (44) :=
        '+ TOKEN RETRIEVED FROM INPUT LINE IS UNKNOWN',

      osv$template_4011 : [static] string (19) :=
        '+ COMMAND TOO LARGE',

      osv$template_4012 : [static] string (23) :=
        '+ INPUT STACK EXHAUSTED';

    CONST
      osc$ziovmt0_count = 12;
  ?? fmt ( format := on ) ??


  ?? fmt ( format := off ) ??
        [4001, osc$error_status, ^osv$template_4001],
        [4002, osc$error_status, ^osv$template_4002],
        [4003, osc$error_status, ^osv$template_4003],
        [4004, osc$error_status, ^osv$template_4004],
        [4005, osc$error_status, ^osv$template_4005],
        [4006, osc$error_status, ^osv$template_4006],
        [4007, osc$error_status, ^osv$template_4007],
        [4008, osc$error_status, ^osv$template_4008],
        [4009, osc$error_status, ^osv$template_4009],
        [4010, osc$error_status, ^osv$template_4010],
        [4011, osc$error_status, ^osv$template_4011],
        [4012, osc$warning_status, ^osv$template_4012],
  ?? fmt ( format := on ) ??
```

Table of Contents