CYBER IMPLEMENTATION LANGUAGE

Miscellaneous Routines Interface

Reference Manual

REVISION DEFINITION SHEET

REV	DATE	DESCRIPTION
1	12/13/83	Preliminary manual released.
2	06/22/84	Updated preliminary manual.
3	09/24/84	This revision reflects the CYBIL 170 Code Generator at Level 617 and incorporates the following features and changes: interfaces to the NOS Screen Formatting facilities; formatted I/O routines; square root and absolute value routines.
4	12/17/84	New interface routines have been added: string to real conversions, exponential and natural logarithm functions, real to an integer power, real to a real power, and three NOS 170 interfaces.

CYBER IMPLEMENTATION LANGUAGE

Miscellaneous Routines Interface Reference Manual

17 DEC 84
REV: 4

Table of Contents

1.0 INTRODUCTION

The programming language used in this implementation is the CYBIL Implementation Language (CYBIL). The details of the interface are defined in terms of CYBIL structures. The interfaces described in this document are available for use through the source maintenance utility provided with the product offering.

On NOS this includes Madify, on NOS/BE it's UPDATE, and on NOS/VE its SCU. References in the document to the *CALL directive (Update), to the *CALLC directive (MADIFY) or the *COPYC directive (SCU) should be interpreted as equivalent functions.

1.1 SCOPE OF DOCUMENT

A separation has been made in this document to simplify documentation efforts and to exemplify the natural modularity.

This document contains information necessary for the understanding and use of Miscellaneous Routines available through the CYBIL Services Library (i.e., CYBCLIB on the C170).

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 Miscellaneous Routines.

CYBIL Reference Manual (60455280)

CYBIL for NOS/VE Language Definition (60464113)

CYBIL for NOS/VE System Interface Usage (60464115)

CYBIL I/O Reference Manual (60460300)

CYBER 180 System Interface Standard (S2196)

SES User's Handbook (60457250)

NOS Version 2 Reference Set Vol 4 Program Interface (60459690)

NOS Version 2 Screen Formatting Reference Manual (60460430)

System Command Language (SCL) ERS (SES Internal)

Input Output Control (IOC) ERS (SES Internal)

Message Generator ERS (SES Internal)

Command Processor (CP) ERS (SES Internal)

1.3 NAMING CONVENTIONS

The following naming conventions have been imposed upon the Miscellaneous Routines in general.

Decknames are of the form Zpcyxxx where:

Z universal identifier

pc two character interface identifier
- OS NOS/VE operating system compatible
- PM NOS/VE program management compatible
- CY NOS/VE CYBIL compatible
- N7 directly related to or dependent upon a feature of NOS 170
- UT utility (none of the above)

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).

Procedure names are of the form pcp$xxxxxxxxxxxxxxx where

pc two character product identifier

p CYBIL procedure identifier

xxxxxxxxxxxx a meaningful, descriptive name of procedure. All procedure names are limited to 31 characters (including prefix).

1.4 DISCLAIMER RELATED TO NOS 170

Procedures using "N7" as an interface identifier have direct NOS 170 dependencies. It is assumed the user of "N7" procedures has indepth experience and knowledge of NOS 170 (the user of "N7" must know what he is doing).

Compatibility between "N7" procedures and future supplied procedures is not guaranteed. Therefore, use the "N7" procedures at your own discretion.

1.5 MISCELLANEOUS ROUTINES USAGE

All procedures described in this document are available for MADIFY, UPDATE and Source Code Utility users. MADIFY common decks are made available by specifying the "CYBCCMN" keyword on the SES.GENCOMP procedure. SCU users can use the GETCOMN procedure to acquire CYBCCMN for subsequent use by the SCU EXPAND_DECK command. The binaries are available for linking by specifying the "CYBCLIB" keyword on the SES.LINK170 procedure.

2.0 MISCELLANEOUS ROUTINES DESCRIPTION

The routines classified as miscellaneous are a disjunct set. They perform services such as CYBIL to NOS interfaces, conversion of data, file manipulations, system utility operations, CYBIL program control, and string manipulations. Note that all interfaces are not available on all operating systems. The reader is alerted to the matrix at the back of this document for further understanding of which interfaces are available on which systems.

Any commonality among these routines lies in the fact that they are self contained primitives. It is intended that the miscellaneous routines are to be an "on-going", growing group.

If you, the reader, have additional routines which are useful make them known. It is the the intent of this document to consolidate individual efforts.

2.1 OBJECTIVES OF MISCELLANEOUS ROUTINES

The objectives of the Miscellaneous Routines are:

1) Provide a documented base of diversified routines which may be added to when a need arises,

2) Provide a group of CYBIL routines each serving a unique purpose,

3) Provide general purpose, independent routines (routines may have basic operating system or file dependencies).

2.2 PHILOSOPHY OF MISCELLANEOUS ROUTINES

When there is a task to be done provide a routine to do it. The routine is an "end condition" or a function that has no lateral dependencies. It can have limited upward or downward dependencies if necessary. Each routine exhibits the characteristic of filling a unique purpose whose scope is limited to that routine.

3.0 MISCELLANEOUS ROUTINES INTERFACES

3.1 DESCRIPTION

The Miscellaneous Routine interfaces consist solely of a set of procedure interfaces. They are grouped in terms of what they interface to or what data they operate upon. This set is intended to be in a constant state of growth as coding within the Tools Group proceeds and new requirements arise.

A summary of each procedure is given along with the procedure reference in the form of a common deck. Where necessary, to further explain parameters, other common decks are included. All information included is intended to be self explanatory.

An appendix is included with a list of all common decks that exist as *callc within the procedure reference common decks. This alphabetic appendix lists contents that are composed of TYPE and CONST information.

Since it is not intended to rewrite any of the available operating system manuals, any procedures which require knowledge of specific areas of the manuals will reference them. The user should consult that document. Also see "N7" disclaimer in this document.

3.2 PROCEDURES

3.2.1 GENERAL PROCEDURES

3.2.1.1 Generate Unique Alphanumeric Strings

There are three procedures available to generate unique alphanumeric character strings. These procedures generate unique strings, labels and file names. There are three separate procedure reference common decks.

```
{ ZUTPUQS    Generates unique string. }

  PROCEDURE [XREF] utp$generate_unique_string ALIAS 'zutpuqs' (VAR
    unique_string: string ( * ));
```

```
{ ZUTPUQL    Generates unique label. }

  PROCEDURE [XREF] utp$generate_unique_label ALIAS 'zutpuql' (VAR
    unique_label: string (7));
```

Note that all labels produced are of the form 9Qxxxxx where xxxxx are the unique characters generated.

```
{ ZUTPUQF    Generates unique file name. }

  PROCEDURE [XREF] utp$generate_unique_file_name ALIAS 'zutpuqf' (VAR
    unique_file_name: string (7));
```

The file names are of the form ZQxxxxx where xxxxx are the unique characters generated. Note that the generated file name is guaranteed to be different from the name of any file currently assigned to the job.

3.2.2 MATHEMATICAL FUNCTIONS

3.2.2.1 Calculate REAL Absolute Value

This function computes the absolute value for the single precision floating point value given.

```
{ ZCYFABS     Calculates the absolute value of the REAL argument.

  FUNCTION [XREF] cyf$abs ALIAS 'ZCYFABS' (arg: real): real;
```

3.2.2.2 Calculate Natural Logarithm

This function calculates the value of ln (x) where x is a single precision floating point number greater than zero.

```
*callc osdstat

{ ZCYFALG   Compute the natural logarithm of arg.

  FUNCTION [XREF] cyf$alog ALIAS 'ZCYFALG'
   (    arg: real;
    VAR status: ost$status): real;
```

3.2.2.3 Compute e to the power x

This function computes e ** x, where e is equal to 2.718281828459045... and x is a single precision floating point value.

```
*callc osdstat

{  ZCYFEXP   Compute e ** arg, where e = 2.718281828459045.....

  FUNCTION [XREF] cyf$exp ALIAS 'ZCYFEXP'
   (    arg: real): real;
```

3.2.2.4 Calculate REAL Square Root

This function computes the square root for the single precision floating point value given.

```
*callc osdstat

{ ZCYFSQR   Calculates the square root of source.

  FUNCTION [XREF] cyf$sqrt ALIAS 'ZCYFSQR'
   (    source: real;
    VAR status: ost$status): real;
```

3.2.2.5 Raise a REAL Value to an INTEGER Power

This function raises the single precision floating point value to the integer power given.

```
*callc osdstat

{ ZCYFX2I   Calculates real_value ** integer_power.

  FUNCTION [XREF] cyf$xtoi ALIAS 'ZCYFX2I'
   (    real_value: real;
        integer_power: integer;
    VAR status: ost$status): real;
```

3.2.2.6 Raise a REAL Value to an REAL Power

This function raises the single precision floating point value to the single precision floating point power given.

```
*callc osdstat

{ ZCYFX2X   Calculates real_value ** real_power.

  FUNCTION [XREF] cyf$xtox ALIAS 'ZCYFX2X'
   (    real_value: real;
        real_power: real;
    VAR status: ost$status): real;
```

3.2.3 DATA CONVERSION PROCEDURES

3.2.3.1 Capitalize String

The following procedure capitalizes the alphabetic characters in a string.

```
{ZUTPCAP    capitalize a string

  PROCEDURE [XREF] utp$capitalize_string ALIAS 'zutpcap' (VAR char_string:
    string ( * ));
```

3.2.3.2 Lowercase a String

The following procedure converts the alphabetic characters in a string to lowercase.

```
{ZCYPLOW    convert a string to lowercase.

  PROCEDURE [XREF] cyp$lowercased_string ALIAS 'zcyplow' (VAR char_string:
    string ( * ));
```

3.2.3.3 Display Code Name to CYBIL String

The purpose of this procedure is to convert a standard NOS or NOS/BE seven character display code name to a seven character CYBIL string which is left justified and blank filled.

```
*callc zuttdcn

{ ZUTPDNS    Converts NOS 170 7 char. disp. code name to CYBIL string. }

  PROCEDURE [XREF] utp$convert_dc_name_to_string ALIAS 'zutpdns' (dc_name:
    utt$dc_name;
    VAR result_string: string (7);
    VAR result_length: 0 .. 7);
```

3.2.3.4 CYBIL String to Zero Filled Display Code Name

This procedure converts CYBIL string to a standard NOS or NOS/BE seven character display code name. The conversion proceeds until either seven characters have been processed or a blank (space) character is encountered. The resulting name is zero filled for each character short of 7.

```
*callc zuttdcn

{ ZUTPSDN    Converts CYBIL string to NOS 170 7 char. disp. code name. }
{              zero filled.                                             }

  PROCEDURE [XREF] utp$convert_string_to_dc_name ALIAS 'zutpsdn'
    (source_string: string ( * );
    VAR dc_name: utt$dc_name);
```

3.2.3.5 CYBIL String to Blank Filled Display Code Name

This procedure converts CYBIL string to a standard NOS or NOS/BE seven character display code name. The conversion proceeds until either seven characters have been processed or a blank (space) character is encountered. The resulting name is blank filled for each character short of 7.

```
*callc zuttdcn

{ ZCYPSDB    Converts CYBIL string to NOS 170 7 char. disp. code name, }
{              blank filled.                                            }

  PROCEDURE [XREF] cyp$cnvt_str_to_dc_name_blank ALIAS 'zcypsdb'
    (source_string: string ( * );
    VAR dc_name: utt$dc_name);
```

3.2.3.6 CYBIL String to Display Code File Name

This procedure converts an adaptable CYBIL string to a standard NOS or NOS/BE seven character display code file name. The conversion proceeds until either seven characters have been processed or a non-alphanumeric character is encountered. The resulting display code file name is left justified, zero filled (e.g., for use in FET).

```
*callc zuttdcn

{ ZUTPSFN    Converts an adaptable CYBIL string to C170 display
{ code file name. }

  PROCEDURE [XREF] utp$convert_string_to_file_name ALIAS 'zutpsfn'
    (source_string: string ( * );
    VAR dc_file_name: utt$dc_name);
```

3.2.3.7 CYBIL String to Display Code String

The purpose of this procedure is to convert a CYBIL string to a display code string. The length of the display code string is in terms of words. Both a word index and character position within the word are input to and updated by the procedure. Conversion stops when the display code string is filled or the CYBIL string is exhausted. If the EOL parameter is TRUE when this procedure is called and there is room for the entire CYBIL string and an end-of-line in the display code string, then an end-of-line is generated in the display code string following the converted CYBIL string. If the EOL was generated, then the EOL parameter is set to TRUE, otherwise it is set false. An EOL is defined (in a display code string) as a right justified field of 12 to 66 bits of zeros.

```
*callc zoststr
*callc zuttenc

{ ZUTPS2D    Converts CYBIL string to display code string. }

  PROCEDURE [XREF] utp$convert_string_to_dc_string ALIAS 'zutps2d'
    (encoding: utt$encoding;
    VAR dc_string: array [ * ] OF packed array [0 .. 9] OF 0 .. 3f(16);
    VAR dc_string_word_index: integer;
    VAR dc_string_char_index: 0 .. 9;
    source_string: string ( * );
    VAR source_index: ost$string_index;
    VAR eol: boolean);
```

3.2.3.8 Display Code String to CYBIL String

The purpose of this procedure is to convert a display code string to a CYBIL string. The length of the display code string is in terms of words. Both a word index and a character position within the word are input to and updated by the procedure. Conversion stops when: 1) the CYBIL string is filled, 2) the display code string is exhausted, or 3) an EOL (end-of-line) is found. An EOL is defined (in a display code string) as a right justified field of 12 to 66 bits of zeros.

```
*callc zoststr
*callc zuttenc

{ ZUTPD2S    Converts display code string to CYBIL string. }

  PROCEDURE [XREF] utp$convert_dc_string_to_string ALIAS 'zutpd2s'
    (encoding: utt$encoding;
    VAR dc_string: {READ} array [ * ] OF packed array [0 .. 9] OF 0 ..
      63;
    VAR dc_string_word_index: integer;
    VAR dc_string_char_index: 0 .. 9;
    VAR result_string: string ( * );
    VAR result_length: ost$string_length;
    VAR eol_found: boolean);
```

3.2.3.9 Integer to String

The purpose of this procedure is to convert an integer to its string representation in the specified radix. If the integer is negative, the leftmost character of the resulting string is a '-'. The string containing the integer's representation left justified and the length of the representation is returned. This length is zero if the string is too small to represent the integer. This procedure can handle integers with values in the range -(2**59-1) .. 2**59-1 on the C170.

```
*callc zoststr

{ ZUTPI2S    Converts integer to string rep. in specified radix. }

  PROCEDURE [XREF] utp$convert_integer_to_string ALIAS 'zutpi2s' (VAR
    result_string: string ( * );
    VAR result_length: ost$string_length;
    source_integer: integer;
    radix: 2 .. 16);
```

3.2.3.10 Integer to Right Justified String

The purpose of this procedure is to convert an integer to its string representation in the specified radix. The resultant string contains the string representation of the integer right justified and zero filled. If the integer is negative, the leftmost character of the resulting string is a '-'. Should the procedure fail, a boolean is set false. Conditions causing failure are overflow (result string too small) and an invalid source string. This procedure can handle integers with values in the range -(2**59-1) .. 2**59-1 on the C170.

```
*CALLC zoststr

{ ZUTPIRS    procedure to convert integer to right justified string

PROCEDURE [XREF] utp$convert_integer_to_rjstring ALIAS 'zutpirs'
    (VAR result_string : string (*);
     VAR conversion_okay : BOOLEAN;
         source_integer: integer;
         radix: 2 .. 16);
```

3.2.3.11 String to Integer

The purpose of this procedure is to convert the string representation of an integer to an integer value. The procedure begins its examination of the source string at the position specified by the source index. That index is incremented by one for each character of the string that is used. The integer may be preceeded by a sign (+ or -). The first character of the integer must be a decimal digit, however, subsequent characters of the integer may be decimal digits or letters (case ignored) A through F (representing hex digits 10 through 15). The integer itself can optionally be immediately followed by a radix specification (unsigned integer 2 through 16 with no leading zeros and enclosed in parentheses). In the absence of a radix specification, 10 is assumed. The radix value must be larger than the largest digit value in the integer. This procedure can handle integers with values in the range -(2**59-1) .. 2**59-1 on the C170.

```
*callc zoststr

{ ZUTPS2I    Converts string rep. of integer to integer value. }

  PROCEDURE [XREF] utp$convert_string_to_integer ALIAS 'zutps2i' (VAR
    source_string: {READ} string ( * );
    VAR source_index: ost$string_index;
    VAR result_integer: integer;
    VAR conversion_worked: boolean);
```

3.2.3.12 String to Real

The purpose of this procedure is to convert the string representation of a real to a single precision floating point value. The procedure begins its examination of the source string at the position specified by the source index. That index is incremented by one for each character of the string that is used. The integer may be preceeded by a sign (+ or -). Only decimal values are allowed.

```
*callc osdstat
*callc osdstr

{ ZCYPS2R    Converts string rep. of a real to a real value.

  PROCEDURE [XREF] cyp$convert_string_to_real ALIAS 'ZCYPS2R'
   (    source_string: string ( * );
    VAR source_index: ost$string_index;
    VAR result_real: real;
    VAR status: ost$status);
```

3.2.3.13 Character Translation (Conversion) Structure

The following are used as translation tables in the conversion to/from ascii. Conversion of ascii to ascii612, ascii612 to ascii, ascii to ascii64, and ascii64 to ascii are available.

```
{ ZUTVCTT    Translation table used in conversion to/from ascii. }

  VAR
    utv$convert_ascii_to_ascii612 ALIAS 'cvas612': [XREF, READ] array
      [char] of packed record
      case long: boolean of
      = FALSE =
        f1: set of 1 .. 53,
        ch: 0 .. 3f(16),
      = TRUE =
        f2: set of 1 .. 47,
        escape_ch: 0 .. 3f(16),
        follower_ch: 0 .. 3f(16),
      casend,
    recend,
    utv$convert_ascii612_to_ascii ALIAS 'cv612as': [XREF, READ] array [0
      .. 3f(16)] of packed record
      case escape: boolean of
      = FALSE =
        f1: set of 1 .. 51,
        ch: char,
      = TRUE =
        f2: set of 1 .. 41,
        conv: ^array [0 .. 3f(16)] of char,
      casend,
    recend,
    utv$convert_ascii_to_ascii64 ALIAS 'cvasc64': [XREF, READ] array
      [char] of 0 .. 3f(16),
    utv$convert_ascii64_to_ascii ALIAS 'cv64asc': [XREF, READ] array [0
      .. 3f(16)] of char;
```

3.2.3.14 Word to Hexadecimal String

The purpose of this procedure is to produce the hexadecimal interpretation of the contents of a C170 word.

```
{ ZUTPW2H    Produces hex interpretation of contents of word. }

  PROCEDURE [XREF] utp$word_to_hexadecimal_string ALIAS 'zutpw2h'
    (pointer_to_word: ^cell;
    VAR hexadecimal_string: string (15));
```

3.2.3.15 Word to Octal String

The purpose of this procedure is to produce the octal interpretation of the contents of a C170 word.

```
{ ZUTPW2O    Produces octal interpretation of contents of word. }

  PROCEDURE [XREF] utp$word_to_octal_string ALIAS 'zutpw2o'
    (pointer_to_word: ^cell;
    VAR octal_string: string (20));
```

3.2.3.16 String to Variable(s)

The common decks ZCYPSCF and ZCYPSSF contain external procedure declarations for routines that will provide a method for programs to format input and to decode strings into program declared variables of various types. The CYBIL function, STRINGREP is available to reverse this process and will take program declared variables of various types and put them in a string.

The CYP$SCANF_n routines read character data from the specified file and decodes the characters into the variables based on the conversion string specification. This is a decode operation, decoding characters from a file into numbers or ASCII strings. These routines are on the ZCYPSCF common deck. The specified file must be opened using the LG#OPEN routine prior to calling any of the CYP$SCANF_n routines.

The user should be wary of using LG#GET or LG#GETPART and any CYP$SCANF_n routines on the same file. CYP$SCANF_n routines read one character at a time from the specified file and continue reading where the previous read operation quit. LG#GET, LG#GETPART, and CYP$SCANF_n can be used together on the same file, but be aware of how they work. See the CYBIL I/O Reference Manual (60460300) for descriptions of LG#OPEN, LG#GET, and LG#GETPART.

The CYP$S_SCANF_n routines decodes a character buffer into the variables based on the conversion string specification. This is a decode operation, decoding characters from a string in memory into numbers or ASCII strings. These routines are on the ZCYPSSF common deck.

3.2.3.16.1 CONVERSION STRING SPECIFICATIONS

A conversion string is a character string which specifies how input or a string is decoded into variables. The format for a conversion string is:

```
' <directive>[<directive> ... ] '
```

The directives can provide the following functions.

- o A numerical format directive causes a variable argument to be interpreted and formatted as an octal, decimal, hexadecimal, or real number.

- o The ASCII format directive causes an argument to be interpreted as an ASCII character string.

- o Miscellaneous format directives exist to generate a field of spaces and tab to a particular character position in the buffer.

Each directive is preceded by the percent sign character, %. Each ends with the directive for the function to be performed. Between the % and the directive can be a number of options which enhance the directive. Conversion strings for decode operations can contain only directives. Directives can be given in either lowercase or uppercase. Spaces within directives are ignored.

3.2.3.16.2 MISCELLANEOUS DIRECTIVES

% fw X

fw If the optional fw field is not specified, the default width is 1.

% $

This directive causes processing to terminate and return to the caller.

3.2.3.16.3 NUMERIC FORMATTING DIRECTIVES

The numeric directives uses 1 parameter to transform a string into a number. The number is returned in the parameter. If no field width is specified, the string or file is searched until a blank or end of line is found.

For the CYP$SCANF_n and CYP$S_SCANF_n routines:

```
% fw  O | D | H | F
```

fw — An optional decimal integer between 1 and 65535: The optional field width will be the number specified, or until a blank if the field width is not specified, or end of line is found.

O,D,H, F — This option indicates the base of the number to be processed and is required.

- O Specifies the number to be an octal integer.
- D Specifies the number to be an decimal integer.
- H Specifies the number to be an hexadecimal integer.
- F Specifies the number to be a decimal real. The input format is an optional sign, a string numbers possibly containing a decimal point and an optional exponent field containing an E or e followed by a possibly signed integer.

Example:

An example of a control specification to read from a file or convert a string into numeric data is:

```
'%7f %4x%4H %  2  x %8F %O'
```

This specification would read 7 characters and create a real, skip 4 spaces, read 4 characters and create a hexadecimal integer, skip 2 spaces, read 8 characters into a real number, and then the remaining characters would be processed as an octal integer. All blanks in the conversion specification are ignored.

The following LOCAL_STRING would be converted using the conversion specification above.

```
'3.14159    ABCD  6.02e+2264'
```

The call would look like this:

```
cyp$s_scanf_4 (local_string, '%7f%4x%4H%2x%8F%O', local_real_1, hex,
      local_real_2, octal);
```

LOCAL_REAL_1 would contain the value of 3.14159, HEX would contain the value of ABCD(16), LOCAL_REAL_2 would contain the value of 6.02E+22, and OCTAL would have the value of 64(8).

3.2.3.16.4 ASCII FORMATTING DIRECTIVES

The ASCII directives may use 2 arguments to process a string. The first argument is used for the string, if no field width is specified, the second contains the string length. The string length is changed if the number of characters read is less than the length of the string.

For the CYP$SCANF_n and CYP$S_SCANF_n routines:

```
% fw  U|L  A
```

fw An optional decimal integer between 1 and 65535: The number of characters read will be the optional field width specified, the length of the string, or until a blank or end of line is found, whichever is smaller. If the field width is omitted, then the other conditions are considered.

U,L This optional argument forces all characters to be either uppercase (U), or lowercase (L). If this option is not specified, then all characters are left untouched.

A The A indicates the conversion string specification is to convert an ASCII string. This is required.

Example:

An example of a control specification to read from a file or convert a string as ASCII data is:

```
'%31U A%$'
```

If the following string was on a file, all the characters would be read from the file, including blanks, and converted to uppercase letters.

```
'help Figure this string out and'
```

The call would look like this:

```
cyp$scanf_2 (local_file, '%31U A%$', local_string, dummy);
```

The variable dummy would not be changed because the field width was given in the conversion specification. LOCAL_STRING would look like this:

```
'HELP FIGURE THIS STRING OUT AND'
```

```
*callc pxiotyp
*callc osdstat
*callc cydefie

{ ZCYPSCF     Read characters from a file and interpret according to the
{             conversion specification and store the results in the remaining
{             arguments.

  PROCEDURE [XREF] cyp$scanf_2 ALIAS 'cypsf2' (f: file;
        conversion_specification: string ( * );
        substitution_parameter1: ^cell;
        substitution_parameter2: ^cell;
    VAR status: ost$status);

  PROCEDURE [XREF] cyp$scanf_3 ALIAS 'cypsf3' (f: file;
        conversion_specification: string ( * );
        substitution_parameter1: ^cell;
        substitution_parameter2: ^cell;
        substitution_parameter3: ^cell;
    VAR status: ost$status);

  PROCEDURE [XREF] cyp$scanf_4 ALIAS 'cypsf4' (f: file;
        conversion_specification: string ( * );
        substitution_parameter1: ^cell;
        substitution_parameter2: ^cell;
        substitution_parameter3: ^cell;
        substitution_parameter4: ^cell;
    VAR status: ost$status);

  PROCEDURE [XREF] cyp$scanf_5 ALIAS 'cypsf5' (f: file;
        conversion_specification: string ( * );
        substitution_parameter1: ^cell;
        substitution_parameter2: ^cell;
        substitution_parameter3: ^cell;
        substitution_parameter4: ^cell;
        substitution_parameter5: ^cell;
    VAR status: ost$status);

  PROCEDURE [XREF] cyp$scanf_6 ALIAS 'cypsf6' (f: file;
        conversion_specification: string ( * );
        substitution_parameter1: ^cell;
        substitution_parameter2: ^cell;
        substitution_parameter3: ^cell;
        substitution_parameter4: ^cell;
        substitution_parameter5: ^cell;
        substitution_parameter6: ^cell;
    VAR status: ost$status);
```

```
PROCEDURE [XREF] cyp$scanf_7 ALIAS 'cypsf7' (f: file;
      conversion_specification: string ( * );
      substitution_parameter1: ^cell;
      substitution_parameter2: ^cell;
      substitution_parameter3: ^cell;
      substitution_parameter4: ^cell;
      substitution_parameter5: ^cell;
      substitution_parameter6: ^cell;
      substitution_parameter7: ^cell;
  VAR status: ost$status);

PROCEDURE [XREF] cyp$scanf_8 ALIAS 'cypsf8' (f: file;
      conversion_specification: string ( * );
      substitution_parameter1: ^cell;
      substitution_parameter2: ^cell;
      substitution_parameter3: ^cell;
      substitution_parameter4: ^cell;
      substitution_parameter5: ^cell;
      substitution_parameter6: ^cell;
      substitution_parameter7: ^cell;
      substitution_parameter8: ^cell;
  VAR status: ost$status);

PROCEDURE [XREF] cyp$scanf_9 ALIAS 'cypsf9' (f: file;
      conversion_specification: string ( * );
      substitution_parameter1: ^cell;
      substitution_parameter2: ^cell;
      substitution_parameter3: ^cell;
      substitution_parameter4: ^cell;
      substitution_parameter5: ^cell;
      substitution_parameter6: ^cell;
      substitution_parameter7: ^cell;
      substitution_parameter8: ^cell;
      substitution_parameter9: ^cell;
  VAR status: ost$status);

PROCEDURE [XREF] cyp$scanf_10 ALIAS 'cypsf10' (f: file;
      conversion_specification: string ( * );
      substitution_parameter1: ^cell;
      substitution_parameter2: ^cell;
      substitution_parameter3: ^cell;
      substitution_parameter4: ^cell;
      substitution_parameter5: ^cell;
      substitution_parameter6: ^cell;
      substitution_parameter7: ^cell;
      substitution_parameter8: ^cell;
      substitution_parameter9: ^cell;
      substitution_parameter10: ^cell;
   VAR status: ost$status);

*callc osdstat
*callc cydefie

{ ZCYPSCF     Read a string and interpret according to the conversion
{             specification and store the results in the remaining arguments.

  PROCEDURE [XREF] cyp$s_scanf_2 ALIAS 'cypss2' (input_string: string ( * );
       conversion_specification: string ( * );
       substitution_parameter1: ^cell;
       substitution_parameter2: ^cell;
    VAR status: ost$status);

  PROCEDURE [XREF] cyp$s_scanf_3 ALIAS 'cypss3' (input_string: string ( * );
       conversion_specification: string ( * );
       substitution_parameter1: ^cell;
       substitution_parameter2: ^cell;
       substitution_parameter3: ^cell;
    VAR status: ost$status);

  PROCEDURE [XREF] cyp$s_scanf_4 ALIAS 'cypss4' (input_string: string ( * );
       conversion_specification: string ( * );
       substitution_parameter1: ^cell;
       substitution_parameter2: ^cell;
       substitution_parameter3: ^cell;
       substitution_parameter4: ^cell;
    VAR status: ost$status);

  PROCEDURE [XREF] cyp$s_scanf_5 ALIAS 'cypss5' (input_string: string ( * );
       conversion_specification: string ( * );
       substitution_parameter1: ^cell;
       substitution_parameter2: ^cell;
       substitution_parameter3: ^cell;
       substitution_parameter4: ^cell;
       substitution_parameter5: ^cell;
    VAR status: ost$status);

  PROCEDURE [XREF] cyp$s_scanf_6 ALIAS 'cypss6' (input_string: string ( * );
       conversion_specification: string ( * );
       substitution_parameter1: ^cell;
       substitution_parameter2: ^cell;
       substitution_parameter3: ^cell;
       substitution_parameter4: ^cell;
       substitution_parameter5: ^cell;
    substitution_parameter6: ^cell;
  VAR status: ost$status);

 PROCEDURE [XREF] cyp$s_scanf_7 ALIAS 'cypss7' (input_string: string ( * );
    conversion_specification: string ( * );
    substitution_parameter1: ^cell;
    substitution_parameter2: ^cell;
    substitution_parameter3: ^cell;
    substitution_parameter4: ^cell;
    substitution_parameter5: ^cell;
    substitution_parameter6: ^cell;
    substitution_parameter7: ^cell;
  VAR status: ost$status);

 PROCEDURE [XREF] cyp$s_scanf_8 ALIAS 'cypss8' (input_string: string ( * );
    conversion_specification: string ( * );
    substitution_parameter1: ^cell;
    substitution_parameter2: ^cell;
    substitution_parameter3: ^cell;
    substitution_parameter4: ^cell;
    substitution_parameter5: ^cell;
    substitution_parameter6: ^cell;
    substitution_parameter7: ^cell;
    substitution_parameter8: ^cell;
  VAR status: ost$status);

 PROCEDURE [XREF] cyp$s_scanf_9 ALIAS 'cypss9' (input_string: string ( * );
    conversion_specification: string ( * );
    substitution_parameter1: ^cell;
    substitution_parameter2: ^cell;
    substitution_parameter3: ^cell;
    substitution_parameter4: ^cell;
    substitution_parameter5: ^cell;
    substitution_parameter6: ^cell;
    substitution_parameter7: ^cell;
    substitution_parameter8: ^cell;
    substitution_parameter9: ^cell;
  VAR status: ost$status);

 PROCEDURE [XREF] cyp$s_scanf_10 ALIAS 'cypss10' (input_string: string ( * );
    conversion_specification: string ( * );
    substitution_parameter1: ^cell;
    substitution_parameter2: ^cell;
    substitution_parameter3: ^cell;
    substitution_parameter4: ^cell;
    substitution_parameter5: ^cell;
    substitution_parameter6: ^cell;
```

```
        substitution_parameter7: ^cell;
        substitution_parameter8: ^cell;
        substitution_parameter9: ^cell;
        substitution_parameter10: ^cell;
    VAR status: ost$status);
```

The following common deck contains the error code constants returned by any of these routines.

```
*callc cydeccr
?? NEWTITLE := 'CYDEFIE :        Formatted Input : 576100 .. 576199', EJECT ??
?? FMT (FORMAT := OFF) ??

  CONST
    cyc$min_ecc_formatted_input     = cyc$min_ecc + 6100,

    cye$invalid_argument            = cyc$min_ecc_formatted_input + 5,
    {E Invalid argument +P in directive}

    cye$duplicate_argument          = cyc$min_ecc_formatted_input + 10,
    {E Duplicate argument +P in directive}

    cye$multiple_directive          = cyc$min_ecc_formatted_input + 15,
    {E Multiple directive +P in conversion specification}

    cye$unfinished_directive        = cyc$min_ecc_formatted_input + 20,
    {E Unfinished directive in string +P'

    cye$duplicate_field_width       = cyc$min_ecc_formatted_input + 25,
    {E Duplicate field width in directive +P'

    cye$illegal_directive           = cyc$min_ecc_formatted_input + 30,
    {E Illegal directive specification +P'

    cye$illegal_exponent            = cyc$min_ecc_formatted_input + 35,
    {E Illegal character for exponent found +P'

    cye$no_string_length            = cyc$min_ecc_formatted_input + 40,
    {E No string length given using field width or parameter}

    cye$non_numerical_character     = cyc$min_ecc_formatted_input + 45,
    {E Non numerical character found for digit +P}

    cye$invalid_number_format       = cyc$min_ecc_formatted_input + 50,
    {E Character found for number +P, does not match base}
    cye$found_file_mark              = cyc$min_ecc_formatted_input + 55,
    {E Operation read file mark before directive finished}

    cye$nil_parameter                = cyc$min_ecc_formatted_input + 60,
    {E Parameter given is NIL +P}

    cyc$max_ecc_formatted_input      = cyc$min_ecc_formatted_input + 99;

?? FMT (FORMAT := ON) ??
?? OLDTITLE ??
```

3.2.4 STRING AND CHARACTER PROCEDURES

3.2.4.1 Compare CYBIL Strings

The purpose of this procedure is to compare CYBIL strings which may be of different lengths. The comparison_result field contains the result of the comparison.
-1 if left < right string
0 if left = right string
+1 if left > right string.

```
{ ZUTPCPS    Compares CYBIL strings which may be of different lengths. }

  PROCEDURE [XREF] utp$compare_strings ALIAS 'zutpcps' (left_operand:
    string ( * );
    right_operand: string ( * );
    VAR comparison_result: - 1 .. 1);
```

3.2.4.2 Build Display Code String Pointer

The purpose of this procedure is to build a display code string pointer. Given a cell (word) address and a character position (0 .. 9) within that word receive a display code string pointer. All display code strings are accessed via such pointers.

```
{ ZUTPDCP    Builds a display code string pointer. }

  PROCEDURE [XREF] utp$create_dc_string_ptr ALIAS 'zutpdcp' (word: ^cell;
    pos: 0 .. 9;
    VAR dc_string_ptr: cell);
```

3.2.4.3 Get Display Code Character From String

The purpose of this procedure is to get the next display code character from a string. The next display code character designated by the display code pointer is returned and the display code pointer is advanced to designate the following character. Note the following character may be in the next word.

```
{ ZUTPDCG    Gets next display code character from a string. }

  PROCEDURE [XREF] utp$get_next_dc_char ALIAS 'zutpdcg' (VAR dc_string_ptr:
    cell;
    VAR dc_char: 0 .. 3f(16));
```

3.2.4.4 Insert Display Code Character

The purpose of this procedure is to insert a display code character at a place designated by the display code pointer. The display code pointer is advanced to designate the display code character which follows the one inserted. Note that this character may be in the next word.

```
{ ZUTPDCI    Inserts disp. code char. at place designated by pointer. }

  PROCEDURE [XREF] utp$insert_next_dc_char ALIAS 'zutpdci' (VAR
    dc_string_ptr: cell;
    dc_char: 0 .. 3f(16));
```

3.2.5 CYBIL SCREEN FORMATTING PROCEDURES

The CYBIL Screen Formatting procedures are intended to imitate the procedures for FORTRAN and COBOL documented in the NOS Screen Formatting Reference Manual (60460430). These interfaces were extended with an eye towards rehosting these interfaces onto NOS/VE. The names for the common decks with the procedure declaration are the same as the FORTRAN and COBOL names preceded with a 'Z'.

For a description of how to generate a panel with PDU, please refer to the Screen Formatting Reference Manual.

The reference describes a routine called SFCSET. There is no corresponding CYBIL procedure because CYBIL only deals with the 7-bit ASCII code.

The CYBIL Screen Formatting procedures are not applicable without the NOS Screen Formatting feature. These routines also need support routines from the NOS system library, SFLIB. A 'LIBRARY,SFLIB.' statement should be included before trying to execute or trying to use SES.LINK170.

3.2.5.1 Close a Panel

This procedure closes the panel specified by the name passed as a parameter. Once the panel is closed, no further operations can be processed unless the panel is reopened. It is not necessary to close a panel before opening another one. Up to 10 panels can be opened at the same time.

The mode parameter specifies whether or not the screen is cleared and the terminal reverts back to line mode when the panel is closed. If the panel specified in the CLOSE call is the last panel displayed by the program, the procedure call should specify reversion to line mode.

```
*callc zostnam
*callc osdstat
*callc zcytslm

{ ZSFCLOS     Closes (unloads) a panel.

  PROCEDURE [XREF] cyp$close_panel ALIAS 'cy$clos'
   (    panel_name: ost$nos170_name;
        mode: cyt$set_screen_or_line_mode;
    VAR status: ost$status);
```

The following common deck has the constant definitions for the mode parameter.

```
{ ZCYTSLM     Defines the terminal mode setting when a panel is closed.

  TYPE
    cyt$set_screen_or_line_mode = (cyc$set_screen_mode,
      cyc$set_line_mode_and_clear, cyc$set_line_mode_unchanged);
```

3.2.5.2 Get an Integer Value

This procedure returns the current value of the named variable field as an integer value.

```
*callc osdstat
*callc zostnam

{ ZSFGETI     Gets an integer value.

  PROCEDURE [XREF] cyp$get_integer ALIAS 'cy$geti'
   (    field_name: ost$nos170_name;
    VAR integer_returned: integer;
    VAR status: ost$status);
```

3.2.5.3 Get a Real Value

This procedure returns the current value of the named variable field as an real value.

```
*callc osdstat
*callc zostnam

{ ZSFGETR     Gets a real value.

  PROCEDURE [XREF] cyp$get_real ALIAS 'cy$getr'
   (    field_name: ost$nos170_name;
    VAR real_returned: real;
    VAR status: ost$status);
```

3.2.5.4 Get the Last Function Key Entered

This procedure returns the value of the last function key pressed on a CDC 721 terminal.

```
*callc osdstat
*callc zcytfkv

{ ZSFGETK     Gets last function key pressed.

  PROCEDURE [XREF] cyp$get_key_value ALIAS 'cy$getk'
   (VAR key_value: cyt$function_key_value;
    VAR status: ost$status);
```

The following common deck has the constant definitions of the possible values returned for the last function key.

```
{ ZCYTFKV     Defines function key values.

  TYPE
    cyt$function_key_value = (cyc$f1_key, cyc$f2_key, cyc$f3_key, cyc$f4_key,
      cyc$f5_key, cyc$f6_key, cyc$f7_key, cyc$f8_key, cyc$f9_key, cyc$f10_key,
      cyc$f11_key, cyc$f12_key, cyc$f13_key, cyc$f14_key, cyc$f15_key,
      cyc$f16_key, cyc$shifted_f1_key, cyc$shifted_f2_key, cyc$shifted_f3_key,
      cyc$shifted_f4_key, cyc$shifted_f5_key, cyc$shifted_f6_key,
      cyc$shifted_f7_key, cyc$shifted_f8_key, cyc$shifted_f9_key,
      cyc$shifted_f10_key, cyc$shifted_f11_key, cyc$shifted_f12_key,
      cyc$shifted_f13_key, cyc$shifted_f14_key, cyc$shifted_f15_key,
      cyc$shifted_f16_key, cyc$next_key, cyc$back_key, cyc$help_key,
      cyc$stop_key, cyc$down_key, cyc$up_key, cyc$fwd_key, cyc$bkw_key);
```

3.2.5.5 Get the Last Cursor Position

This procedure returns values that define the last position of the screen cursor. FIELD_NAME indicates the variable field in which the cursor was last positioned. INDEX is the character position within the variable field where the cursor was last positioned. ROW is the row number of the variable field if the variable is an element of a table. If the variable is not part of a table, ROW is returned as 0.

```
*callc zostnam
*callc osdstat

{  ZSFGETP    Gets last position of screen cursor.

  PROCEDURE [XREF] cyp$get_cursor_position ALIAS 'cy$getp'
   (VAR field_name: ost$nos170_name;
    VAR index: integer;
    VAR row: integer;
    VAR status: ost$status);
```

3.2.5.6 Open a Panel

This procedure loads a panel and prepares it for use. It also sets the terminal to screen mode if it is not already in screen mode. To locate the specified panel, the system searches first a library contained in a local file named PANELIB, (if one exists,) then the user's global library set and finally, the system libraries. CYP$OPEN_PANEL does not display the panel on the screen.

A panel must be opened before is can be used by any other panel processing procedure. If a procedure attempts to use a panel before the panel is opened, the program is terminated abnormally.

```
*callc zostnam
*callc osdstat
*callc cydesfe

{ ZSFOPEN     Opens a panel and prepares it for use.

  PROCEDURE [XREF] cyp$open_panel ALIAS 'cy$open'
   (VAR panel_name: ost$nos170_name;
    VAR status: ost$status);
```

The following common deck contains the error code constants returned by CYP$OPEN_PANEL.

```
*callc cydeccr
?? NEWTITLE := 'CYDESFE :     Screen Formatting : 576000 .. 576099', EJECT ??
?? FMT (FORMAT := OFF) ??

  CONST
    cyc$min_ecc_screen_formatting    = cyc$min_ecc + 6000,

    cye$panel_not_found              = cyc$min_ecc_screen_formatting + 1,
    {E Panel +P not found}

    cye$panel_format_wrong           = cyc$min_ecc_screen_formatting + 2,
    {E Panel +P capsule incorrectly formatted}

    cye$too_many_open_panels         = cyc$min_ecc_screen_formatting + 3,
    {E Too many panels already open}

    cye$panel_already_open           = cyc$min_ecc_screen_formatting + 4,
    {E Specified panel +P is already open}

    cye$internal_errors              = cyc$min_ecc_screen_formatting + 5,
    {E Internal errors}

    cye$terminal_not_identified      = cyc$min_ecc_screen_formatting + 6,
    {E No screen or line command: terminal is not identified}

    cye$terminal_not_supported       = cyc$min_ecc_screen_formatting + 7,
    {E Terminal is not supproted by NOS}

    cye$panel_not_open               = cyc$min_ecc_screen_formatting + 8,
    {E Panel +P is not open}

    cyc$max_ecc_screen_formatting    = cyc$min_ecc_screen_formatting + 99;

?? FMT (FORMAT := ON) ??
?? OLDTITLE ??
```

3.2.5.7 Position Row in a Table

This procedure establishes a current row in the named table and is used in conjunction with the CYP$GET_INTEGER and CYP$GET_REAL procedures. Before calling CYP$GET_INTEGER or CYP$GET_REAL that references a table variable, your program must call CYP$POSITION_ROW to specify the row number of the desired variable. The row number established remains in effect for all following CYP$GET_INTEGER and CYP$GET_REAL procedure calls until the row number is changed by another call to this procedure.

```
*callc zostnam
*callc osdstat

{ ZSFPOSR     Sets the current row in the specified table.

  PROCEDURE [XREF] cyp$position_row ALIAS 'cy$posr'
   (    table_name: ost$nos170_name;
        row: integer;
    VAR status: ost$status);
```

3.2.5.8 Set Cursor Position

This procedure sets the screen cursor to a selected input variable field in the displayed panel. CYP$SET_CURSOR_POSITION can be called before an CYP$READ_PANEL or CYP$SHOW_PANEL procedure call to modify the default variable entry sequence. The default sequence procedes from the first variable field in the panel to the last.

```
*callc zostnam
*callc osdstat

{ ZSFSETP     Sets the screen cursor to the specified input field_name field.

  PROCEDURE [XREF] cyp$set_cursor_position ALIAS 'cy$setp'
   (    field_name: ost$nos170_name;
        index: integer;
        row: integer;
    VAR status: ost$status);
```

3.2.5.9 Read a Panel

This procedure permits the user to enter input data at the terminal. Data entered is returned to the application program. If the panel has not been previously displayed on the screen, CYP$READ_PANEL displays it using initial variable values specified for the panel (using the VAR statement VALUE parameter). INPUT_STRING is the string where CYP$READ_PANEL will return the input data entered at the terminal for the panel specified by PANEL_NAME. The string returned is a character string formed by concatenating the contents of all variable fields in the panel.

```
*callc zostnam
*callc osdstat

{ ZSFSREA     Displays panel on terminal and data can be entered.

  PROCEDURE [XREF] cyp$read_panel ALIAS 'cy$srea'
   (    panel_name: ost$nos170_name;
    VAR input_string: string ( * );
    VAR status: ost$status);
```

3.2.5.10 Show a Panel

This procedure displays the specified panel with the current variable values and allows the user to enter additions or modifications to the variable values. If the panel is not already displayed on the screen, CYP$SHOW_PANEL displays it using OUTPUT_STRING for the variable field values. CYP$SHOW_PANEL is equivalent to an CYP$WRITE_PANEL followed by CYP$READ_PANEL.

OUTPUT_STRING is where CYP$SHOW_PANEL will get the variable values to display before modification by the user. This parameter is a character string with the contents of all the variable fields concatenated.

INPUT_STRING is where CYP$SHOW_PANEL will return the variable values after modification by the user.

```
*callc zostnam
*callc osdstat

{ ZSFSSHO     Displays panel with current information and reads data.

  PROCEDURE [XREF] cyp$show_panel ALIAS 'cy$ssho'
   (    panel_name: ost$nos170_name;
    VAR output_string: string ( * );
    VAR input_string: string ( * );
    VAR status: ost$status);
```

3.2.5.11 Write a Panel

This procedure displays the specified panel with the current variable values. If the panel is not already displayed on the screen, CYP$WRITE_PANEL displays it using OUTPUT_STRING for the variable field values.

OUTPUT_STRING is where CYP$WRITE_PANEL will get the variable values to display.

```
*callc zostnam
*callc osdstat

{ ZSFSWRI     Displays panel with current field_name field values.

  PROCEDURE [XREF] cyp$write_panel ALIAS 'cy$swri'
   (    panel_name: ost$nos170_name;
    VAR output_string: string ( * );
    VAR status: ost$status);
```

3.2.6 CYBIL PROGRAM PROCEDURES

3.2.6.1 Initiate CYBIL Program Environment

The purpose of this procedure is to initiate the 'environment' for a CYBIL program. The current version just returns the command program name, a pointer to the command line (control statement) that caused the program to be executed, and the command line index ready for scanning the command's parameter list.

```
*callc zostnam
*callc zoststr
*callc osdstat
*callc zutpsmt
*callc zutprmt

{ ZOSPINI    Initiates environment for a CYBIL program. }

  PROCEDURE [XREF] osp$initiate ALIAS 'zospini' (VAR command_name:
    ost$name_descriptor;
    VAR command_line_pointer: ^string ( * );
    VAR command_line_index: clt$string_index;
    VAR status: ost$status);
```

The command_line_pointer is obtained using the osp$get_control_statement procedure. See the description of that routine for more information.

3.2.6.2 Terminate a CYBIL Program

The purpose of this procedure is to terminate a CYBIL program. If status.normal is FALSE the message designated by the remaining fields of the status record is issued to the dayfile. If status.normal is FALSE and status.state is osc$error_status or osc$fatal_status the program is aborted, otherwise the program is terminated normally.

```
*callc zostnam
*callc osdstat

{ ZOSPEND    Terminates a CYBIL program. }

  PROCEDURE [XREF] osp$terminate ALIAS 'zospend' (VAR command_name: {READ}
    ost$name_descriptor;
    VAR status: {READ} ost$status);
```

3.2.6.3 Terminate a CYBIL Program with Generated Message

The purpose of this procedure is to terminate a CYBIL program with the option of sending a message to a specified file. The presence of the message generator template array is required. (See the Message Generator ERS for a description of the template array.) The user has the option to pass a pointer to a CYBIL I/O legible file descriptor, in which case the message is written to the file. If no file output is desired the pointer must be set NIL. If status.normal is FALSE the Message Generator is used to generate a message to the dayfile and optionally the specified legible file. If status.normal is FALSE and status.state is osc$error_status or osc$fatal_status the program is aborted, otherwise the program is terminated normally.

```
*CALLC osdstat
*CALLC zostnam

{ZOSPTWM   procedure to terminate CYBIL program with message

PROCEDURE [XREF] osp$terminate_with_message ALIAS 'zosptwm'
    (VAR command_name : {READ} ost$name_descriptor;
     VAR status : {READ} ost$status;
     VAR file_descriptor : file);
```

3.2.6.4 End CYBIL Program

The purpose of this procedure is to terminate a CYBIL program. This is accomplished by executing an ENDRUN call.

```
{ ZUTPEND    Terminates a CYBIL program. }

  PROCEDURE [XREF] utp$end ALIAS 'zutpend';
```

3.2.6.5 Abort CYBIL Program

The purpose of this procedure is to provide an interface for CYBIL programs to turn off reprieve processing and abort the program.

```
{ ZUTPABT    Aborts a CYBIL program. }

  PROCEDURE [XREF] utp$abort ALIAS 'zutpabt';
```

An additional procedure which can be declared as:

```
  PROCEDURE [XREF] abort;
```

is available also. It differs from the above procedure in that it does not turn off reprieve processing and gives a CYBIL Post Mortem dump.

Another procedure proves useful to suppress terminal output of the last control statement on a voluntary abort. A blank message is written to the user dayfile, then a utp$abort is done. The net effect of the blank message is no terminal message.

```
{ZUTPCAA    advance user dayfile with null message and abort(reprieve off)

  PROCEDURE [XREF] utp$clear_and_abort ALIAS 'zutpcaa';
```

3.2.6.6 CYBIL to Compass Interface

The purpose of these macros is to provide a consistent, standard interface from CYBIL to a Compass routine. Three macros are included: ENTR for entry into the compass routine, DONE for exit from the compass routine, and CALL for calling another procedural interface from within a Compass routine.

```
* ZPXIDEF    PROVIDES CYBIL TO COMPASS STANDARD INTERFACE

          CTEXT  ZPXIDEF - CYBIL INTERFACE DEFINITIONS
          SPACE  2
          B1=1
          SPACE  4
*** THE FOLLOWING DEFINES THE NIL POINTER, INDICATING IN CYBIL
*   A POINTER POINTING TO NOTHING


NIL       EQU    377777B
          SPACE  4
*** THE FOLLOWING MACROS DEFINE THE ENTRY/EXIT SEQUENCE OF
*   CYBIL PROCEDURES.

*   ENTRY CONDITIONS
*     B1   1 - THE GENERATED CODE COUNTS ON THIS
*     B2   POINTER TO CALLER'S STACK FRAME / TOP OF STACK (TOS)
*     B3   STACK LIMIT

*     X1
*     X2   LAST 5 PARAMETERS PASSED TO CALLEE, THAT FIT INTO AN
*     X3   X REGISTER, STARTING WITH X1
*     X4
*     X5

*     B5   POINTER TO ARGUMENT EXTENSION LIST (IF ANY)
*     X7   PROCEDURE LINKAGE WORD (RETURN ADDRESS)

*   EXIT CONDITIONS
*     B1   1
*     B2   AS ON ENTRY
*     B3   AS ON ENTRY
*     X1   AS X7 ON ENTRY
```

```
          SPACE  4
*** THE FOLLOWING MACRO DEFINES THE ENTRY SEQUENCE
*   USING THE CYBIL STACK DISCIPLINE.


          PURGMAC ENTR

          MACRO  ENTR,NAME
          LOCAL  MORE
MORE      RJ     =XCIL#SPE    * CALL PROLOG EXCEPTION ROUTINE
NAME      SX0    B2          * COPY POINTER TO CALLER'S STACK FRAME
          LX0    18          * POSITION IT
          SB7    6           * SET ROUTINE STACK FRAME SIZE
          BX6    X7+X0       * MERGE IT INTO LINKAGE WORD
          SB2    B2-B7       * ADJUST STACK FRAME POINTER
          GE     B3,B2,MORE  * CHECK IF ROOM IN STACK SEGMENT
          SA6    B2          * STORE LINKAGE WORD INTO STACK
          ENDM
          SPACE  4
*** DONE DEFINES THE CODE SEQUENCE TO RETURN FROM A
*   CYBIL PROCEDURE.


          PURGMAC DONE

DONE      MACRO
          SB1    1
          SA1    B2          * LOAD LINKAGE WORD
          SB7    X1          * GET RETURN ADDRESS
          SB2    B2+6        * RESTORE CALLER'S STACK POINTER
          JP     B7          * RETURN
          ENDM
          SPACE  4
*** THE FOLLOWING MACRO DEFINES THE CALLING SEQUENCE FOR A CYBIL
*   PROGRAM.  IT IS ASSUMED, THAT ARGUMENTS ARE ALREADY SET UP.


          PURGMAC CALL

CALL      MACRO  P
          LOCAL  RETAD
          SX7    RETAD       * SET RETURN ADDRESS
          EQ     P           * TRANSFER CONTROL TO PROCEDURE
RETAD     BSS    0
          ENDM
          SPACE  2
          ENDX
```

3.2.7 POINTER MANIPULATION PROCEDURES

3.2.7.1 Offset of Pointer From Base

This procedure returns the offset (in terms of cells) of an address (pointer) from a base address (pointer). This is comparable to the CYBIL language relative pointer feature.

```
{ ZUTPCOP    Returns offset of address from base address. }

  PROCEDURE [XREF] utp$compute_offset_of_pointer ALIAS 'zutpcop' (base:
    ^cell;
    pointer: ^cell;
    VAR offset: integer);
```

3.2.7.2 Compute Pointer From Offset

This procedure returns a pointer to the "offset-th" cell from a base address (pointer). This is comparable to the CYBIL language relative pointer feature.

```
{ ZUTPCPO    Returns pointer to offset-th cell from base address. }

  PROCEDURE [XREF] utp$compute_pointer_from_offset ALIAS 'zutpcpo' (base:
    ^cell;
    offset: integer;
    VAR pointer: ^cell);
```

3.2.8 CYBIL OVERLAY LOADING ON NOS

3.2.8.1 Overlay Structures

A parameter for an overlay load is the level numbers. The following structure provides for simple assignment of level numbers prior to the call. It is found on common deck ZUTTOVL.

```
{ZUTTOVL    primary and secondary overlay level numbers

  TYPE
    utt$overlay_level = packed record
      primary: 0 .. 3f(16),
      secondary: 0 .. 3f(16),
    recend;
```

3.2.8.2 Load Overlay

The following procedure provides the CYBIL user with the capability to load overlays from an overlay file created via LINK170 using an input file created by SES.BOVLAY. Common deck ZUTPOVL provides the procedure interface.

```
*callc zostnam
*callc zuttovl
*callc osdstat

{ZUTPOVL    load an overlay

  PROCEDURE [XREF] utp$load_overlay ALIAS 'zutpovl' (file_name:
    ost$nos170_name;
    level_numbers: utt$overlay_level;
    ptr_to_ovl_proc: ^cell;
    VAR status: ost$status);
```

Before this interface is invoked the environment must be established. An overlay load must be executed from a file. This file must be created by a SES.LINK170 request. The contents of the LGO file given to LINK170 must be of a special form. It must contain specially created compass object routines that externally reference the entry points of procedures/programs which are the outermost of each overlay. It is recommended that the main overlay (0,0) serve as the procedure to contain the overlay loading.

The lower level overlays need no modification except that the outermost procedure must have an XDCLed entry point. The main procedure, which will load the overlays, is expected to use pointers to the procedures which will execute as overlays. When the load of an overlay procedure is executed the #LOC of the pointer to the procedure is passed to the overlay loader. Upon return from the overlay loader the contents of the pointer may be executed as the procedure. Parameters may be passed to the procedure if they were declared as part of the pointer to procedure description. The procedure to be executed as the overlay is never directly referenced in the procedure loading the overlay. Procedures to be used as overlays to a procedure may be XREFerenced in that procedure, but it cannot be performed by name.

Any variables shared by overlayed procedures may be accomodated by XDCL,XREF. Lower level overlays may not have more than one entry point.

Below is an example of what is required to alter the CYBIL source code to use overlays.

The following is a sketch of some basic given elements of an existing program.

```
         .
         .
         .
procedure [XREF] utp$initiate (VAR status : ost$status);
         .
         .
         .
utp$initiate (status);
         .
         .
         .
```

The following is a sketch of an overlay load:

```
VAR
  ptr_to_utp$initiate : ^procedure (VAR status: ost$status);

    .
    .
  overlay_level.primary := 1;
  overlay_level.secondary := 2;

  utp$load_overlay (overlay_file, overlay_level,
    #LOC(ptr_to_utp$initiate), status);
      .
      .
  ptr_to_utp$initiate^ (status);
      .
      .
```

The assumptions are:

(1) the procedure pointed to by ptr_to_utp$initiate is a procedure XDCLed and having a single parameter status,

(2) the overlay file name is the name of the overlay file (b parameter) created by SES.LINK170 using a lgo created by SES.BOVLAY.

The overlayed procedure may be XREFerenced in the procedure calling it but referencing it by name will cause it to link there.

Note the convention used to label the pointer to procedure variable. It is suggested that overlay procedures use a common deck of the following format:

```
PROCEDURE [XREF] utp$initiate alias 'zutpini' (VAR status : ost$status);

VAR
  ptr_to_utp$initiate: ^procedure (VAR status: ost$status);
```

This allows a program library cross reference to locate the place of use of a procedure. XREFed CYBIL procedures do not link unless they are performed by name in the compiled code.

3.2.8.3 Create Overlay Tree Structure (SES.BOVLAY)

This procedure is intended to create a LGO file consisting of overlay cards and relocatable binaries that have external references to entry points of the overlay procedures. This file is input to SES.LINK170 to create an executable overlay file.

```
SES.BOVLAY  i =          l =      b =
```

i (optional) the name of input file containing data used to create compass source code decks (default is INPUT).

l (optional) name of file to contain the list of assembled code composing generated overlay linkage decks. Default is LIST.

b (optional) name of a load and go format file created by assembling the generated compass source decks (default is LGO). It is intended that this LGO file is used as input (f parameter) to SES.LINK170 to create an overlay file.

The input data to SES.BOVLAY is of the following form:

ext (required) 7 character name of entry point (alias) of procedure (program if level 0,0) which was XDCLed and is the outermost of the overlay.

ovl (required) the decimal level number of the overlay associated with the entry point (p,s) where p is the primary level and s is secondary level.

ent (optional) 1..7 character name of entry point (which is the transfer address) for the deck created. It may prove useful to indicate the level number as part of the name. If not specified a unique name is generated.

ident (optional) name of ident card of compass routine created. It may prove useful to use a symbol which indicates the overlay level number. If not specified a unique name is generated.

Note that the input data must be ordered in overlayed sequence-- that is (0,0) (1,0) (1,1) (2,0) etc. Up to 77(8) levels are allowed. See the CYBER Loader Reference Manual for further details on overlaying.

When input is not specified a prompt is issued giving a suggested data ordering. Then data is input without keyword assignment ('keyword =...') it must be in the suggested order. When assignment is used the parameters may be scrambled on the input line. Blanks or commas may be used as separators. If a parameter is omitted prior to one which is specified, commas are required to note the absence. An example of input data follows:

```
zutpini 1 2 zuteol2 zutiol2
zutpout 1 3, , zutiol3
zutpexp 1 4
zutpmin 1 5 zuteol5
```

The binary decks are created from generated source code:

```
          IDENT    ident
          ENTRY    ent
          EXT      ext
          LCC      OVERLAY(,p,s)
 ent      EQ       ext
          END      ent
```

The deck simply causes generation of an overlay card and a binary which references the entry point of the CYBIL procedure which is to become an overlay.

3.2.9 SYSTEM UTILITY PROCEDURES

3.2.9.1 Current Date

The purpose of this procedure is to return the current date in a user selectable format. The procedure reference is on common deck ZPMPDAT and the date formats are on deck ZOSTDAT.

```
{ ZOSTDAT    Returns current date in a user selectable format. }
*callc osddate
```

```
*callc osdstat
*callc osddate

{ ZPMPDAT    Contains current date. }

  PROCEDURE [XREF] pmp$get_date ALIAS 'zpmpdat' (format: ost$date_formats;
    VAR date: ost$date;
    VAR status: ost$status);
```

3.2.9.2 Current Time

The purpose of this procedure is to return the current time of day in a user selectable format. The procedure reference is found on common deck ZPMPTIM and the time formats are found on deck ZOSTTIM.

```
{ ZOSTTIM    Returns current time of day in user selectable format. }
*callc osdtime



*callc osdstat
*callc osdtime

{ ZPMPTIM    Contains current time. }

  PROCEDURE [XREF] pmp$get_time ALIAS 'zpmptim' (format: ost$time_formats;
    VAR time: ost$time;
    VAR status: ost$status);
```

3.2.9.3 Get NOS 170 Control Statement Arguments

The purpose of this procedure is to make available to a CYBIL/CC program, control statements which have been cracked by NOS. Only positional arguments are handled (separators are ignored). The arguments are returned in an adaptable array of seven character strings. The array is both an input and output parameter such that if an actual argument is omitted from the program call, the corresponding element of the array is unaltered. This means the array can be preset with default values for arguments. When more actual arguments are specified on the call than there are elements in the array, the procedure aborts the program with an appropriate dayfile message.

The procedure reference is found on common deck ZUTPCSA.

```
{ ZUTPCSA    Makes continued statements cracked by NOS available to }
{             the calling CYBIL program.                             }

  PROCEDURE [XREF] utp$get_control_statement_args ALIAS 'zutpcsa' (VAR
    args: array [ * ] OF string (7));
```

Note that this procedure should be used prior to execution of any OPEN via CYBIO else the first array entry may not contain the contents of the first positional argument.

3.2.9.4 Program's Control Statement as CYBIL String

The purpose of this procedure is to obtain a program's control statement (card) as a CYBIL string. Upon return the control_statement_pointer points to a string of the precise length of the control statement (allocated in the system heap).

Continuation lines are allowed either from a batch job/procedure file stream or from a terminal. Continuation is signaled by terminating the line(s) with an ellipsis (two or more periods). The first character of the continuation line logically replaces the first period of the continuation ellipsis. A control statement of up to 2000 characters can be constructed using continuation. When reading continuation lines from an interactive terminal, the prompt:

```
..?
```

is issued and should be interpreted as: "enter continuation line". If a control statement longer than 2000 characters is entered, the program is aborted by this procedure. The procedure reference is found on common deck ZOSPGCS.

```
*callc osdstat

{ ZOSPGCS    Obtains program's control stmt. (card) as CYBIL string. }

  PROCEDURE [XREF] osp$get_control_statement ALIAS 'zospgcs' (VAR
    control_statement_pointer: ^string ( * );
    VAR status: ost$status);
```

3.2.9.5 Get Current User Name

The purpose of this procedure is to return the current user name. The procedure reference is found on common deck ZUTPGUN.

```
{ ZUTPGUN    Returns the current user name. }

  PROCEDURE [XREF] utp$get_user_name ALIAS 'zutpgun' (VAR user_name: string
    (7);
    VAR user_name_length: 1 .. 7);
```

3.2.9.6 Issue Dayfile Message

The purpose of this procedure is to send a message string to the job's dayfile. The string is converted to the 64 character set (6-bit display code) before being sent to the dayfile and line one of the control point. This includes conversion of all lower case letters to upper case. The procedure reference is found on common deck ZUTPMSG. A string of up to 256 characters may be used without fear of truncation.

```
{ ZUTPMSG    Sends message string to job's dayfile. }

  PROCEDURE [XREF] utp$issue_dayfile_message ALIAS 'zutpmsg' (message:
    string ( * ));
```

3.2.9.7 Log Metrics Data

Use of these routines will issue a dayfile message in the following format.

```
  - TOOLNAME    ET          CPTIME    MSACT
```

Where:

TOOLNAME = Name of tool, 1 to 10 characters.
ET = Elapsed time used by tool, formatted as hh.mm.ss.
CPTIME = Central processor time in seconds.
MSACT = Mass storage activity in KUNS.

An initial call must be made to start recording the data. A second call is then required to issue the dayfile message.

From a CYBIL Program:

The initial call from a CYBIL program is on common deck ZUTPSMT. This routine is called automatically if the OSP$INITIATE routine is used.

```
 PROCEDURE [XREF] utp$start_metrics_time ALIAS 'zutpsmt';
```

The second call from a CYBIL program is on common deck ZUTPRMT.

```
 PROCEDURE [XREF] utp$report_metrics_time ALIAS 'zutprmt' (toolname: string
    (*));
```

From a COMPASS Program:

Both calls are made by doing:

```
   RJ  GETIMEC
```

The initial call has A1=0, the second call has A1=TOOLNAME. TOOLNAME must be left justified and blank filled.

3.2.9.8 Determine If Job Origin Is Batch

The purpose of this function is to inform the caller whether or not the executing job was initiated from a batch source. The function examines the NOS 170 job communication area to acquire job origin information. The returned value of the function is set to true or false based on the following job origin types (defined in the NOS V1 Reference Manual Volume 2):

DESCRIPTION	RETURNED VALUE
System	FALSE
Local Batch	TRUE
Remote Batch	TRUE
Time-Sharing	FALSE
Multi-Terminal	FALSE

```
{ ZUTFBOJ    Determine if the job is of batch origin }

  FUNCTION [XREF] utf$batch_origin_job ALIAS 'zutfboj': boolean;
```

Two applications of this function are illustrated below:

```
 [ use in IF statements ]

     IF utf$batch_origin_job () THEN
       { do something }
     IFEND;

 [ use in assignment statements ]

     var_name := utf$batch_origin_job ();
```

3.2.10 TERMINAL INTERRUPT PROCEDURES

User condition processing is a limited set based upon the existence of extended reprieve capabilities within CYBIL runtime. This also depends on a NOS 170 system at or later than R4 level.

3.2.10.1 Interruptable Condition Request Codes

The request codes correspond to the condition mask bits of the NOS 170 REPRIEVE/RECOVR. Note that a limitation stated in the NOS 170 reference manual excludes terminal interrupts, but since CYBIL CC has pseudo RECOVR code the terminal interrupt is allowed. The common deck is ZUTTRCV.

```
{ ZUTTRCV    RECOVR mask conditions

  TYPE
    utt$request_codes = (terminal_interrupt, normal_end, cp_abort,
      pp_abort, operator_action, limits_exceeded, pp_error, mode_error),
      { 200, 100, 040, 020, 010, 004, 002, 001 }

    utt$recovr_request = set of utt$request_codes;
```

3.2.10.2 Initialize Terminal Interrupt Detection

This procedure establishes a user recovery routine to detect terminal interrupt conditions. This is accomplished by using utp$activate_recovr_request with a mask only for terminal interrupt condition and a pointer to procedure utp$record_terminal_interrupt. An externally declared variable utv$terminal_interrupt_count is incremented each time the procedure is invoked. After a count of three is reached the program is aborted. It is expected that the user will periodically use utp$terminal_interrupt_detected to examine the count and clear it.

```
{ZUTPITI    initialize terminal interrupt detection

  PROCEDURE [XREF] utp$init_term_interrupt_detect ALIAS 'zutpiti';
```

3.2.10.3 Was Terminal Interrupt Detected

This function detects the occurrence of a terminal interrupt at a user selected time during the execution of a user program. It is intended to be used at a time convenient to the user. The variable utv$terminal_interrupt_count is checked for a non zero value and reset to zero. The boolean value is set true when the count was non zero else false. The user is free to direct action depending upon his needs. Checking for past occurence of a terminal interrupt avoids the immediate danger of attempting to invoke a user process that may use non-reentrant code of the NOS 170 system.

```
{ZUTFTID    (function)latent check for terminal interrupt

  FUNCTION [XREF] utf$terminal_interrupt_detected ALIAS 'zutftid'
    : boolean;
```

3.2.10.4 Ask for Direction

This procedure is supplied for the user who wishes to ask the question 'QUIT OR RESUME' upon detecting a terminal interrupt. A response of "RESUME" sets end_operation equal false while "QUIT" sets end_operation to "true". This routine has a dependency on the message template array of the message generator and requires use of product code 'UT' when the SES.GENMAR proc is used to create the template array.

```
*callc zn7txch

{ZUTPASK    procedure to prompt user for direction when terminal
{interrupt recognized

  PROCEDURE [XREF] utp$ask_for_direction ALIAS 'zutpask' (VAR
    end_operation: boolean);
```

3.2.11 FILE SYSTEM PROCEDURES

3.2.11.1 Acquire a File

This procedure provides a "high-level" interface to the facility made available via procedure n7p$acquire_file (see the description of that procedure for details).

```
*callc zn7ppfm
*callc zuttaqr

{ ZUTPAQR    Interfaces facility made avail. to localize a file. }

  PROCEDURE [XREF] utp$acquire_file ALIAS 'zutpaqr' (local_file_name:
    string ( * );
    permanent_file_name: string ( * );
    user_name: string ( * );
    password: string ( * );
    pack_name: string ( * );
    mode: n7t$pfm_modes;
    request: utt$acquire_request_codes;
    VAR response: utt$acquire_response_codes);
```

3.2.11.2 File Assigned to Job?

The purpose of this procedure is to determine if a file is assigned (local) to a job.

```
{ ZUTPIFL    Determines if file is assigned (local) to a job. }

  PROCEDURE [XREF] utp$is_file_local ALIAS 'zutpifl' (file_name: string
    ( * );
    VAR is_file_local: boolean);
```

3.2.11.3 Return a File

The purpose of this procedure is to remove the assignment of a file to the current job.

```
{ ZUTPRTF    Removes assignment of file to current job. }

  PROCEDURE [XREF] utp$return_file ALIAS 'zutprtf' (file_name: string (
    * ));
```

3.2.11.4 Rewind a File

The purpose of this procedure is to position a file at its beginning of information.

```
{ ZUTPRWF    Positions file at its Beginning Of Information. }

  PROCEDURE [XREF] utp$rewind_file ALIAS 'zutprwf' (file_name: string (
    * ));
```

3.2.11.5 Message About NOS 170 Permanent File

The purpose of this procedure is to issue an informative message to the dayfile concerning a permanent file. The format of the message is:

```
xxxxxxxx PFN=nnnnnnn UN=uuuuuuuu
```

Where xxxxxxxx is the supplied message string
nnnnnnn is the permanent file name (obtained from the FET)
uuuuuuu is the user name of the owner of the file (obtained from the FET).
If the user name field of the FET is 0, the UN= part of the message is omitted.

```
*callc zn7tfet

{ ZN7PPIM    Issues message to dayfile concerning permanent file. }

  PROCEDURE [XREF] n7p$pf_info_message ALIAS 'zn7ppim' (main_message:
    string ( * );
    VAR fet_with_pfn_and_un: n7t$fet);
```

3.2.11.6 Localize File

The purpose of this procedure is to locate (acquire -- make local) the file specified by a FET. Searching for the file is restricted by the nature of the request code. If the file was local, it is rewound. If the file is accessed via PFM, both an attach and a get are attempted. If necessary a wait is done for the file to become not busy or for PFM to become not busy, providing the error processing bit in the FET is set. If the bit is set the procedure assumes that the erad field of the FET is set as well. The pfm_error_occurred parameter is set TRUE if PFM gives a response other than:
n7c$pfm_file_found,
n7c$pfm_file_not_found,
n7c$pfm_file_busy, or
n7c$pfm_pf_utility_active otherwise it is unaltered.

```
*callc zn7tfet
*callc zuttaqr

{ ZN7PAQR    Localizes a file specified by an FET. }

  PROCEDURE [XREF] n7p$acquire_file ALIAS 'zn7paqr' (VAR fet: n7t$fet;
    request: utt$acquire_request_codes;
    VAR response: utt$acquire_response_codes);
```

The acquire request and response codes are found on deck ZUTTAQR.

```
{ ZUTTAQR    Contains acquire request and response codes. }

  TYPE
    utt$acquire_request_codes = (utc$acquire_anywhere,
      utc$acquire_local_only, utc$acquire_permanent_only),
    utt$acquire_response_codes = (utc$acquire_not_found,
      utc$acquire_was_local, utc$acquire_was_indirect,
      utc$acquire_was_direct, utc$acquire_error);
```

3.2.11.7 Extract a Record

EXTRACT is a procedure that enables easy retrieval of records from permanent file (or local) libraries.

EXTRACT is similar in function to the NOS "GTR" statement. It differs from "GTR" in the following ways:

o EXTRACT insists that the library to be searched has a directory (this can be built using the SES object management facilities or by using the NOS utility "LIBEDIT").

o The record type parameter for EXTRACT, if given, applies to all records to be extracted, and if not given, only the names of the records are used when searching the library.

o Each extracted record is copied to its own local file by EXTRACT, rather than all to the same file.

o EXTRACT does not insist that the library to be searched be local to the job when it's called, but will ACQUIRE the library from a permanent file catalog.

The procedure call format is:

```
PROCEDURE utp$extract_record_from_library (
  VAR file_list: {READ} utt$local_file_and_record_list;
      abort_when_record_not_found: boolean;
      requested_record_type: any_type .. n7c$proc;
      library_local_file_name: string ( * );
      library_to_be_searched: string ( * );
      user_name: string ( * );
      library_password: string ( * );
      library_packname: string ( * );
  VAR status: ost$status);
```

file_list is an array, each entry containing the local file name given to the record once it's extracted, and the name of the record to be extracted.

abort_when_record_not_found indicates that extracting should stop when a record is not found. If it is false, the procedure continues extracting other records in the file_list. Error status is set abnormal in either case.

requested_record_type specifies the record type (if given, it applies to all records being extracted; if omitted, only the record names are used when searching the library).

library_local_file_name specifies the local file name for the library This is the name used to make the "is file local?" test when ACQUIRing the library.

library_to_be_searched specifies the name of the library to be searched for the records.

user_name is the user name of the permanent file catalog to be searched for "library_to_be_searched" if it's not already local.

library_password specifies the library's permanent file password.

library_packname specifies the library's permanent file packname.

Valid record type designators are documented under the description of the "CATALOG" control statement in the NOS Reference Manual.

In addition to these standard types, there's one more "type" processed by EXTRACT, which is designated by "TXT". This "type" is used to denote "TEXT" records that, when extracted, are to have their first line (which contains the record's name) "stripped off". This is useful if, for example, one has records containing directives for a NOS utility, in which case the name of such a record is in all likelihood an illegal directive to the utility program.

EXTRACT will return an error status under any of the following conditions:

- the specified library could not be ACQUIREd

- the library file does not have a directory as the last record before end-of-information

- one or more of the requested records could not be found

Note, however, that EXTRACT won't return immediately if it does not find one of the requested records unless the abort_when_record_not_found parameter was coded on the call. Error status will be set in either case.

If the library file was not local to the job when EXTRACT was called, it will be RETURNed when EXTRACT terminates normally; but, if the library file was local, EXTRACT will REWIND it prior to normal termination.

File_list is declared on deck ZUTTLRT.

```
{ ZUTTLRT    Type declaration for EXTRACT file name and record list. }


*callc zostnam

  TYPE
    utt$local_file_and_record_list = ARRAY [1 .. *] OF record
      local_file_name: ost$nos170_name,
      record_name: ost$nos170_name,
    recend;
```

Valid record types are found on deck ZN7TSRT.

```
{ ZN7TSRT    Contains type information for records. }

  CONST { NOS 170 symbols for 'logical' record types }
    n7c$text = 0(16),
    n7c$pp = 1(16),
    n7c$cos = 2(16),
    n7c$rel = 3(16),
    n7c$ovl = 4(16),
    n7c$ulib = 5(16),
    n7c$opl = 6(16),
    n7c$op1c = 7(16),
    n7c$op1d = 8(16),
    n7c$abs = 9(16),
    n7c$ppu = 0a(16),
    n7c$cap = 0e(16),
    n7c$proc = 10(16);
```

```
{ ZUTPEXT    Extract records from a library. }

*callc zuttlrt
*callc zn7tsrt
*callc zostnam
*callc osdstat

  CONST
    n7c$any_type = - 2,
    n7c$txt_type = - 1;

  PROCEDURE [XREF] utp$extract_record_from_library ALIAS 'zutpext' (VAR
    file_list: {READ} utt$local_file_and_record_list;
        abort_when_record_not_found: boolean;
        requested_record_type: n7c$any_type .. n7c$proc;
        library_local_file_name: string ( * );
        library_to_be_searched: string ( * );
        user_name: string ( * );
        library_password: string ( * );
        library_packname: string ( * );
    VAR status: ost$status);
```

Error codes are found on deck ZUTCERO.

```
{ ZUTCERO    Misc utility routine error codes. }

  CONST
    utc$format_error = 11,
    utc$record_not_found_on_error = 12,
    utc$missing_or_bad = 13,
    utc$library_not_found = 14,
    utc$library_acquire_error = 15,
    utc$record_not_found_error = 16;
```

3.2.11.8 Set Record Type

This procedure determines the name and type of a NOS 170 logical record from the first 64 words located in a working buffer. The procedure reference is found on common deck ZN7PSRT while type information for records is found on deck ZN7TSRT.

```
*callc zuttdnv

{ ZN7PSRT    Determines name and type of NOS 170 logical record. }

  PROCEDURE [XREF] n7p$set_record_type ALIAS 'zn7psrt' (ptr_to_record:
    ^cell;
    VAR record_name_and_type: utt$dc_name_and_value);
```

3.2.11.9 Is File Writable?

This procedure determines if a file is writable (e.g., attached in write mode) by the current job.

```
{ ZUTPIFW   Determines if file is writable by current job. }

  PROCEDURE [XREF] utp$is_file_writable ALIAS 'zutpifw' (file_name:
    string ( * );
    VAR is_file_writable: boolean);
```

3.2.11.10 Get Directory Record from Binary File

The purpose of this procedure is to read, from a (binary) file, a directory record (type OPLD). The directory (if it exists) must be the last record in the file (optionally followed by an end_of_file mark. If no directory is found a NIL pointer is returned. If the directory is found, space is allocated for it in the system heap, the record descriptor entries are read into that space and a pointer to the space is returned.

```
*callc pxiotyp
*callc zn7tdir

{ ZN7PRDR    Reads from a binary file a directory record. }

  PROCEDURE [XREF] n7p$get_opld_directory ALIAS 'zn7prdr' (binary_file:
    file;
    VAR opld_directory_pointer: ^n7t$opld_directory);
```

3.2.11.11 Assign Legible File to Terminal

This procedure assigns a legible file to a terminal. The assignment should be done before the file is opened. It is the users responsibility to open and close the file. Any problems encountered result in the file_assigned variable set false.

```
{ ZUTPAFT    Assign file to a terminal }

  PROCEDURE [XREF] utp$assign_file_to_terminal ALIAS 'zutpaft' (file_name:
    string ( * );
    VAR file_assigned: boolean);
```

3.2.12 CYBIL TO NOS 170 PROCEDURES

A common deck which is needed in many of the NOS 170 interface procedures is ZN7TJCA. It contains the format of the Job Communication Area.

```
{ ZN7TJCA    Format of Job Communication Area for CYBIL to NOS 170. }

?? fmt ( format := off ) ??

  TYPE
    n7t$job_communication_area = PACKED RECORD
      res1    : SET OF 1 .. 45,                       { RA + 0 }
      cf      : BOOLEAN,               { CFO bit }
      res2    : SET OF 1 .. 1,
      p       : BOOLEAN,               { pause flag }
                                       { sense switches }
      ssw     : PACKED ARRAY[ - 6 .. - 1] OF BOOLEAN,
                                       { FORTRAN switches }
      fsw     : PACKED ARRAY[ - 6 .. - 1] OF BOOLEAN,
                                                      { RA + 1 }
      sname   : 0 .. 3FFFF(16),        { sys req name }
      unused1 : SET OF 1 .. 1,
      r       : BOOLEAN,               { auto recall flag }
      unused2 : SET OF 1 .. 4,
      sargs   : 0 .. 0FFFFFFFFF(16),   { sys req args }
                                                      { RA + 2..63 }
      argr    : ARRAY[1 .. 32(16)] OF PACKED RECORD
        arg   : 0 .. 3FFFFFFFFFF(16),  { parmater }
        sep   : 0 .. 3FFFF(16),        { separator }
      RECEND,
                                                      { RA + 64 }
      pgnr    : 0 .. 3FFFFFFFFFF(16),  { prog name }
      actr    : 0 .. 3FFFF(16),        { argument count }
                                                      { RA + 65 }
      cmur    : BOOLEAN,               { CMU flag }
      unused3 : SET OF 1 .. 40,
      lwpr    : BOOLEAN,               { loader flag }
      nwal    : ^CELL,                 { next word avail for load }
                                                      { RA + 66 }
      xjpr    : BOOLEAN,               { CEJ/MEJ flag }
      cpu0_is : BOOLEAN,               { CPU0 has inst stack }
      cpu1    : BOOLEAN,               { CPU1 is present }
      res3    : SET OF 1 .. 4,
      pp_#    : 0 .. 1F(16),            { number of PPs }
      cm_size : 0 .. 0FFF(16),          { CM size }
      jopr    : 0 .. 0FFF(16),          { job origin }
      unused4 : SET OF 1 .. 4,
      dis     : BOOLEAN,                { DIS flag }
      rss     : BOOLEAN,                { RSS flag }
      fwpr    : ^CELL,                  { first word of prog }
                                                       { RA + 67 }
      csmr    : BOOLEAN,                { char set mode }
      unused5 : SET OF 1 .. 29,
      ldrr    : BOOLEAN,                { loader completion flag }
      unused6 : SET OF 1 .. 29,
                                                       { RA + 70 }
      ccdr    : ALIGNED[0 MOD 8] ARRAY[1 .. 8] OF
                          PACKED ARRAY[0 .. 9] OF 0 .. 3F(16),
    RECEND;                                            { RA + 100 }

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

This can be made available as a variable in the program area with the following declaration:

```
  VAR
    jca ALIAS 'SW=RA0' : [XREF] n7t$job_communication_area;
```

3.2.12.1 NOS 170 Combined Input Output (CIO) Request

There are two procedures to interface CYBIL with NOS 170 CIO. The only difference between the two is the skip operations parameter. They are available on common deck ZN7PCIO.

This interface is intended primarily for use by "higher-level" utility routines.

```
*callc zn7tfet

{ ZN7PCIO    NOS 170 combined input output (CIO) request. }

  CONST { CIO request codes }
    n7c$cio_rphr = 0(16),
    n7c$cio_read = 8(16),
    n7c$cio_readskp = 10(16),
    n7c$cio_readcw = 80(16),
    n7c$cio_readls = 88(16),
    n7c$cio_rphrls = 98(16),
    n7c$cio_readns = 0a8(16),
    n7c$cio_readei = 180(16),
    n7c$cio_wphr = 4(16),
    n7c$cio_write = 0c(16),
    n7c$cio_writer = 14(16),
    n7c$cio_writef = 1c(16),
    n7c$cio_writecw = 84(16),
    n7c$cio_rewrite = 8c(16),
    n7c$cio_rewriter = 94(16),
    n7c$cio_rewritef = 9c(16),
    n7c$cio_open_read_norewind = 40(16),
    n7c$cio_open_read_rewind = 60(16),
    n7c$cio_open_write_norewind = 44(16),
    n7c$cio_open_write_rewind = 64(16),
    n7c$cio_open_alter_norewind = 50(16),
    n7c$cio_open_alter_rewind = 70(16),
    n7c$cio_close_norewind = 58(16),
    n7c$cio_close_rewind = 68(16),
    n7c$cio_close_unload = 78(16),
    n7c$cio_close_return = 7c(16),
    n7c$cio_bksp = 20(16),
    n7c$cio_bkspru = 24(16),
    n7c$cio_rewind = 28(16),
    n7c$cio_unload = 30(16),
    n7c$cio_return = 38(16),
    n7c$cio_evict = 4c(16),
    n7c$cio_skipr = 0a0(16),
    n7c$cio_skipf = 3c0a0(16),
    n7c$cio_skiprb = 1a0(16),
    n7c$cio_skipfb = 3c1a0(16),
    n7c$cio_skipei = n7c$cio_skipf,
    n7c$cio_eoi_skip_count = 3ffff(16);

  PROCEDURE [XREF] n7p$cio_with_skip ALIAS 'zn7pios' (VAR fet: n7t$fet;
    request_code: - 3ffff(16) .. 3ffff(16);
    skip_count: 0 .. n7c$cio_eoi_skip_count);

  PROCEDURE [XREF] n7p$cio ALIAS 'zn7pcio' (VAR fet: n7t$fet;
    request_code: - 3ffff(16) .. 3ffff(16));
```

For an explanation of the CIO functions consult the NOS 170 Reference Manual.

3.2.12.2 NOS 170 Control Point Manager (CPM)

There are two procedures to allow the CYBIL user to alter or interrogate parameters in the job control point area which controls his job in the system. The choice of procedure is dependent on the request code. The subfunction_code is usually set to zero. The procedure references and some constant request codes are available on common deck ZN7PCPM.

This interface is intended primarily for use by "higher-level" utility routines.

```
{ ZN7PCPM    NOS 170 Control Point Manager (CPM). }

  CONST { CPM request codes }
    n7c$cpm_setqp = 0(16),
    n7c$cpm_setpr = 1(16),
    n7c$cpm_mode = 2(16),
    n7c$cpm_setasl = 3(16),    {subcode = 2}
    n7c$cpm_setjsl = 3(16),    {subcode = 1}
    n7c$cpm_settl = 3(16),     {subcode = 0}
    n7c$cpm_erexit = 4(16),
    n7c$cpm_console = 5(16),
    n7c$cpm_rollout = 6(16),
    n7c$cpm_setssm = 8(16),
    n7c$cpm_onsw = 9(16),
    n7c$cpm_offsw = 0a(16),
    n7c$cpm_getjn = 0b(16),
    n7c$cpm_getqp = 0c(16),
    n7c$cpm_getpr = 0d(16),
    n7c$cpm_getem = 0e(16),
    n7c$cpm_gettl = 0f(16),    {subcode = 0}
    n7c$cpm_jsl = 0f(16),    {subcode = 1}
    n7c$cpm_getasl = 0f(16), {subcode = 2}
    n7c$cpm_setdfri = 10(16),
    n7c$cpm_setui = 11(16),
    n7c$cpm_setlc = 12(16),
    n7c$cpm_setrfl = 13(16),
    n7c$cpm_getjcr = 14(16),
    n7c$cpm_setjcr = 15(16),
    n7c$cpm_setss = 16(16),
    n7c$cpm_getjo = 17(16),
    n7c$cpm_getja = 18(16),
    n7c$cpm_usecpu = 19(16),
    n7c$cpm_usernum = 1a(16),
    n7c$cpm_getflc = 1b(16),
    n7c$cpm_setpacknam = 1d(16),
    n7c$cpm_getpacknam = 1e(16),
    n7c$cpm_getss = 1f(16),
    n7c$cpm_version = 24(16),
    n7c$cpm_getlc = 25(16),
    n7c$cpm_getgls = 26(16),
    n7c$cpm_setgls = 27(16),
    n7c$cpm_machid = 28(16),
    n7c$cpm_getact = 29(16),
    n7c$cpm_setmfl = 2a(16),
    n7c$cpm_getpfp = 2f(16),
    n7c$cpm_getlof = 31(16),
    n7c$cpm_setlof = 32(16),
    n7c$cpm_getjci = 3c(16),   {subcode = 0}
    n7c$cpm_setjci = 3c(16),   {subcode = 1}
    n7c$cpm_protect = 3d(16);

  PROCEDURE [XREF] n7p$cpm_with_value ALIAS 'zn7pcpm' (value: -
    1ffff(16) .. 1ffff(16);
    request_code: n7c$cpm_setqp .. n7c$cpm_protect;
    subfunction_code: 0 .. 3f(16));

  PROCEDURE [XREF] n7p$cpm_with_pointer ALIAS 'zn7pcpm' (pointer: ^cell;
    request_code: n7c$cpm_erexit .. n7c$cpm_protect;
    subfunction_code: 0 .. 3f(16));
```

For an explanation of the CPM functions consult the NOS 170 Reference Manual.

The following decks describe the format of information for use in making CPM requests.

```
{ ZN7TEMR    Contains type definition for exit mode. }

  TYPE
    n7t$exit_mode = packed record
      fill: set of 1 .. 48,
      em: 0 .. 0fff(16),
    recend;
```

```
{ ZN7TFLR    Contains type definition for field length. }
```

```
  TYPE
    n7t$field_length = packed record
      jmfl: 0 .. 0fff(16),
      icfl: 0 .. 0fff(16),
      fill: 0 .. 0fff(16),
      rifl: 0 .. 0fff(16),
      flir: 0 .. 0fff(16),
    recend;



{ ZN7TJCR    Contains type definition for job control registers. }

  TYPE
    n7t$job_control_registers = packed record
      ef: 0 .. 3f(16),
      r3: - 1ffff(16) .. 1ffff(16),
      r2: - 1ffff(16) .. 1ffff(16),
      r1: - 1ffff(16) .. 1ffff(16),
    recend;



{ ZN7TRCR    Contains type definition for rollout control. }

  TYPE
    n7t$rollout_control = packed record
      fill: set of 1 .. 30,
      evd: 0 .. 3ffff(16),
      rtp: 0 .. 0fff(16),
    recend;



{ ZN7TSET    Contains NOS 170 symbols for error types. }

  CONST
    n7c$aret = 1(16),
    n7c$pset = 2(16),
    n7c$ppet = 3(16),
    n7c$cpet = 4(16),
    n7c$pcet = 5(16),
    n7c$tlet = 6(16),
    n7c$flet = 7(16),
    n7c$tket = 8(16),
    n7c$sret = 9(16),
    n7c$fset = 0a(16),
    n7c$odet = 0b(16),
    n7c$spet = 0c(16),
    n7c$rret = 0c(16),
    n7c$oket = 0d(16),
    n7c$sset = 0e(16),
    n7c$ecet = 0f(16),
    n7c$peet = 10(16),
    n7c$syet = 11(16),
    n7c$oret = 12(16);



{ ZN7TSJO    NOS 170 symbols for job origin type. }

  CONST { NOS 170 symbols for job origin types }
    n7c$syot = 0,
    n7c$bcot = 1,
    n7c$eiot = 2,
    n7c$txot = 3;



{ ZN7TSSS    NOS 170 symbols for subsystems. }

  CONST { NOS 170 symbols for sub-systems }
    n7c$nuls = 0,
    n7c$bass = 1,
    n7c$fors = 2,
    n7c$ftns = 3,
    n7c$exes = 4,
    n7c$bats = 5,
    n7c$accs = 6,
    n7c$tras = 7;
```

3.2.12.3 NOS 170 Local File Manager (LFM)

This procedure allows the CYBIL user to interface to the NOS 170 Local File Manager. The setid_code parameter should be zero except for a request of n7c$lfm_setid. The procedure reference, function constants and error code constants are on common deck ZN7PLFM.

This interface is intended primarily for use by "higher-level" utility routines. It may be necessary to set other fields in the fet in order to meet request requirements. Consult the NOS Reference Manual for details.

```
*callc zn7tfet

{ ZN7PLFM    Allows CYBIL user interface to NOS 170 Local File Manager. }

  CONST { LFM request codes }
    n7c$lfm_rename = 0(16),
    n7c$lfm_assign01 = 1(16),
    n7c$lfm_common = 2(16),
    n7c$lfm_release03 = 3(16),
    n7c$lfm_print = 4(16),
    n7c$lfm_punch = 5(16),
    n7c$lfm_punchb = 6(16),
    n7c$lfm_p8 = 7(16),
    n7c$lfm_lock = 8(16),
    n7c$lfm_unlock = 9(16),
    n7c$lfm_status12 = 0a(16),
    n7c$lfm_status13 = 0b(16),
    n7c$lfm_request14 = 0c(16),
    n7c$lfm_request15 = 0d(16),
    n7c$lfm_batch = 0e(16),
    n7c$lfm_setid = 0f(16),
    n7c$lfm_assign20 = 10(16),
    n7c$lfm_accsf = 11(16),
    n7c$lfm_encsf = 12(16),
    n7c$lfm_pscsf = 13(16),
    n7c$lfm_label = 14(16),
    n7c$lfm_getfnt = 15(16),
    n7c$lfm_request26 = 16(16),
    n7c$lfm_entervsn = 17(16),
    n7c$lfm_release30 = 18(16),
    n7c$lfm_primary = 19(16),
    n7c$lfm_filinfo = 1a(16);

  CONST { LFM error codes }
```

```
    n7c$lfm_ok = 0(16),
    n7c$lfm_file_found = 0(16),
    n7c$lfm_file_not_found = 1(16),
    n7c$lfm_file_name_error = 2(16),
    n7c$lfm_illegal_file_type = 3(16),
    n7c$lfm_file_empty = 4(16),
    n7c$lfm_magnet_not_active = 5(16),
    n7c$lfm_duplicate_lib_file_name = 6(16),
    n7c$lfm_illegal_equipment = 7(16),
    n7c$lfm_equipment_not_available = 8(16),
    n7c$lfm_duplicate_file_name = 9(16),
    n7c$lfm_illegal_user_access = 0a(16),
    n7c$lfm_illegal_user_number = 0b(16),
    n7c$lfm_illegal_id_code = 0c(16),
    n7c$lfm_resex_detected_error = 0d(16),
    n7c$lfm_io_sequence_error = 0e(16),
    n7c$lfm_output_file_limit = 0f(16),
    n7c$lfm_local_file_limit = 10(16),
    n7c$lfm_no_mass_storage = 11(16),
    n7c$lfm_illegal_file_mode = 12(16),
    n7c$lfm_fet_too_short = 13(16),
    n7c$lfm_getfnt_table_too_large = 14(16),
    n7c$lfm_bad_change_file_org_typ = 15(16),
    n7c$lfm_parameter_block_busy = 16(16),
    n7c$lfm_address_out_of_range = 17(16);

  TYPE
    n7t$lfm_error_codes = 0 .. 0ff(16);

  PROCEDURE [XREF] n7p$lfm ALIAS 'zn7plfm' (request_code: n7c$lfm_rename ..
    n7c$lfm_filinfo;
    VAR fet: n7t$fet;
    setid_code: 0 .. 3f(16));
```

For an explanation of the LFM function and error codes consult the NOS 170 Reference Manual.

The ZN7TSFT common deck contains the file type constants.

```
{ ZN7TSFT    Contains NOS 170 symbols for file types. }

  CONST { NOS 170 symbols for file types }
    n7c$inft = 0(16),
    n7c$roft = 1(16),
    n7c$prft = 2(16),
    n7c$phft = 3(16),
    n7c$teft = 4(16),
    n7c$quft = 5(16),
    n7c$syft = 5(16),
    n7c$loft = 6(16),
    n7c$cmft = 7(16),
    n7c$lift = 8(16),
    n7c$ptft = 9(16),
    n7c$pmft = 0a(16),
    n7c$faft = 0b(16),
    n7c$hsft = 0c(16),
    n7c$lcft = 0d(16),
    n7c$cnft = 0e(16),
    n7c$mxft = 0f(16);
```

3.2.12.4 NOS 170 Dayfile Message

This procedure allows the CYBIL user to issue a display code message to the dayfile. Note that the message is also sent to line one of the control point. The procedure reference is on common deck ZN7PMSG.

```
{ ZN7PMSG    Allows CYBIL user to issue disp. code msg. to NOS 170
{dayfile. }

  PROCEDURE [XREF] n7p$issue_dayfile_message ALIAS 'zn7pmsg'
    (ptr_to_dc_message: ^cell;
    destination_code: 0 .. 7);
```

For a complete explanation of the options for destination code consult the MESSAGE description in the NOS 170 Reference Manual.

This interface is intended primarily for "higher-level" utility routines. Another routine (utp$issue_dayfile_message) is available that accepts a CYBIL string as the message text, and is therefore generally more useful for most applications.

3.2.12.5 NOS 170 Permanent File Manager (PFM)

This procedure allows the CYBIL user to interface to the NOS 170 Permanent File Manager. The procedure reference, request codes, category codes, access mode codes and error codes are on common deck ZN7PPFM.

This interface is intended primarily for use by "higher-level" utility routines.

```
*callc zn7tfet

{ ZN7PPFM    Allows CYBIL user interface to NOS 170 PFM. }

  CONST { PFM request codes }
    n7c$pfm_save = 1,
    n7c$pfm_get = 2,
    n7c$pfm_purge = 3,
    n7c$pfm_catlist = 4,
    n7c$pfm_permit = 5,
    n7c$pfm_replace = 6,
    n7c$pfm_append = 7,
    n7c$pfm_define = 8,
    n7c$pfm_attach = 9,
    n7c$pfm_change = 10;

  CONST { PFM file category codes }
    n7c$pfm_ct_private = 0,
    n7c$pfm_ct_semi_private = 1,
    n7c$pfm_ct_public = 2;

  CONST { PFM file access mode codes }
    n7c$pfm_m_write = 0,
    n7c$pfm_m_read = 1,
    n7c$pfm_m_append = 2,
    n7c$pfm_m_execute = 3,
    n7c$pfm_m_null = 4,
    n7c$pfm_m_modify = 5,
    n7c$pfm_m_read_modify = 6,
    n7c$pfm_m_read_append = 7;

  TYPE
    n7t$pfm_modes = n7c$pfm_m_write .. n7c$pfm_m_read_append;

  CONST { PFM error codes }
    n7c$pfm_ok = 0(16),
    n7c$pfm_file_found = 0(16),
    n7c$pfm_file_busy = 1(16),
    n7c$pfm_file_not_found = 2(16),
    n7c$pfm_file_empty = 3(16),
    n7c$pfm_file_not_on_mass_storag = 4(16),
    n7c$pfm_file_already_permanent = 5(16),
    n7c$pfm_file_not_local = 6(16),
    n7c$pfm_file_name_error = 7(16),
    n7c$pfm_illegal_user_access = 8(16),
    n7c$pfm_illegal_device_request = 9(16),
    n7c$pfm_file_too_long = 0a(16),
    n7c$pfm_illegal_request = 0b(16),
    n7c$pfm_device_unavailable = 0c(16),
    n7c$pfm_illegal_file_type = 0d(16),
    n7c$pfm_pf_utility_active = 0e(16),
    n7c$pfm_data_transfer_error = 0f(16),
    n7c$pfm_catalog_overflow_files = 10(16),
    n7c$pfm_catalog_overflow_size = 11(16),
    n7c$pfm_prus_not_available = 12(16),
    n7c$pfm_io_sequence_error = 13(16),
    n7c$pfm_local_file_limit = 14(16),
    n7c$pfm_pru_limit = 15(16),
    n7c$pfm_permit_limit_exceeded = 16(16),
    n7c$pfm_reserved_27 = 17(16),
    n7c$pfm_sys_resex_failure_30 = 18(16),
    n7c$pfm_sys_track_limit = 19(16),
    n7c$pfm_sys_file_length_error = 1a(16),
    n7c$pfm_sys_random_index_error = 1b(16),
    n7c$pfm_sys_dir_acc_file_error = 1c(16),
    n7c$pfm_sys_replace_error = 1d(16),
    n7c$pfm_sys_pfm_abort = 1e(16),
    n7c$pfm_sys_mass_storage_error = 1f(16),
    n7c$pfm_sys_file_data_error = 20(16),
    n7c$pfm_sys_permit_error = 21(16),
    n7c$pfm_sys_data_permit_error = 22(16),
    n7c$pfm_sys_eoi_changed = 23(16),
    n7c$pfm_sys_resex_failure_44 = 24(16),
    n7c$pfm_reserved_45 = 25(16),
    n7c$pfm_reserved_46 = 26(16),
    n7c$pfm_reserved_47 = 27(16),
    n7c$pfm_sys_file_structure_err = 28(16),
    n7c$pfm_sys_system_sector_error = 29(16),
    n7c$pfm_reserved_52 = 2a(16),
    n7c$pfm_reserved_53 = 2b(16),
    n7c$pfm_reserved_54 = 2c(16),
    n7c$pfm_reserved_55 = 2d(16),
    n7c$pfm_reserved_56 = 2e(16),
    n7c$pfm_reserved_57 = 2f(16),
    n7c$pfm_reserved_60 = 30(16),
    n7c$pfm_reserved_61 = 31(16),
    n7c$pfm_reserved_62 = 32(16),
    n7c$pfm_reserved_63 = 33(16),
    n7c$pfm_reserved_64 = 34(16),
    n7c$pfm_reserved_65 = 35(16),
    n7c$pfm_reserved_66 = 36(16),
    n7c$pfm_reserved_67 = 37(16),
    n7c$pfm_reserved_70 = 38(16),
    n7c$pfm_sys_staging_error = 39(16),
    n7c$pfm_file_being_staged = 3a(16),
    n7c$pfm_file_awaiting_staging = 3b(16),
    n7c$pfm_file_not_available = 3c(16),
    n7c$pfm_file_is_direct = 3d(16),
    n7c$pfm_file_is_indirect = 3e(16),
    n7c$pfm_reserved_77 = 3f(16),
    n7c$pfm_file_stagable = 40(16),
    n7c$pfm_sys_pfc_address_error = 41(16),
    n7c$pfm_sys_pfc_data_error = 42(16),
    n7c$pfm_non_stagable_request = 43(16),
    n7c$pfm_interlock_not_available = 44(16),
    n7c$pfm_alt_image_obsolete = 45(16),
    n7c$pfm_sys_alt_storage_error = 46(16),
    n7c$pfm_fnt_full = 47(16),
    n7c$pfm_alt_image_not_obsolete = 48(16),
    n7c$pfm_activity_count_limit = 49(16);

  TYPE
    n7t$pfm_error_codes = 0 .. 0ff(16);

  PROCEDURE [XREF] n7p$pfm ALIAS 'zn7ppfm' (request_code: n7c$pfm_save ..
    n7c$pfm_change;
    VAR fet: n7t$fet);
```

For further details on the request codes available consult the NOS 170 Reference Manual.

3.2.12.6 NOS 170 Recall

This procedure allows the CYBIL user to interface to the NOS 170 system RECALL facility. It enables the CYBIL user to relinquish the CPU until the recall time has elapsed. The procedure reference is available on common deck ZN7PRCL.

```
{ ZN7PRCL    Allows CYBIL user interface to NOS 170 RECALL facility. }

  PROCEDURE [XREF] n7p$recall ALIAS 'zn7prcl';
```

For a detailed explanation of RECALL consult the NOS 170 Reference Manual.

3.2.12.7 NOS 170 Translate Control Statement

This procedure allows the CYBIL user to interface to the NOS 170 translate control statement facility. A user may read a control statement from or place a control statement in the control statement stream. The procedure reference, request codes and sub-function codes are available on common deck ZN7PTCS.

This interface is intended primarily for use by "higher-level" utility routines.

```
{ ZN7PTCS    Allows CYBIL user interface to NOS 170 trans. cont. stmt. }

  CONST { TCS request codes }
    n7c$tcs_read = 4,
    n7c$tcs_execute = 5;

  CONST { TCS sub-function codes }
    n7c$tcs_read_and_advance = 0,
    n7c$tcs_read_if_not_local_file = 1,
    n7c$tcs_read_even_if_local_file = 2,
    n7c$tcs_add_for_nosbe_format = 4;

  PROCEDURE [XREF] n7p$translate_control_statement ALIAS 'zn7ptcs'
    (request_code: n7c$tcs_read .. n7c$tcs_execute;
    sub_function: 0 .. 6;
    ptr_to_dc_control_statement: ^cell);
```

A detailed explanation of the request codes and sub-function may be found in the NOS 170 Reference Manual.

3.2.12.8 NOS 170 Read or Backspace Control Statement

This procedure allows the CYBIL user to backspace to the previous control statement, or read the next control statement without having NOS try to crack the parameters. After reading the control statement, it will be in RA+70B--RA+77B. This procedure uses the Scope Function Processor/ACE routine to manipulate the control statement file.

```
{ ZN7PACE    Read or backspace control statement file. }

  CONST
    n7c$backspace_cs_file = 20(16),
    n7c$read_cs_file = 8(16);

  PROCEDURE [XREF] n7p$advance_control_card ALIAS 'zn7pace' (function_code:
    8(16) .. 20(16));
```

3.2.12.9 NOS 170 Time Processor

There are a variety of requests available to the CYBIL user through the n7p$time interface. Most of the requests are processed by the system monitor directly rather than through a specific function processor. There are requests to return the current time of day in display code, return the current date in display code, return the current Julian date, return the current date and time in packed format, return the real elapsed time since deadstart, return the accumulated system resource units, return the accumulated central processor time used by the job, and return the packed time as display code. The request codes and procedure reference are available on common deck ZN7PTIM.

This interface is intended primarily for use by "higher-level" utility routines. Other routines (pmpget_date, pmpget_time) exist to perform the more common requests for most applications.

```
{ ZN7PTIM    Contains NOS 170 time processor information. }

  CONST { TIM request codes }
    n7c$tim_jobs_cpu_time = 0,
    n7c$tim_date = 1,
    n7c$tim_clock = 2,
    n7c$tim_julian_date = 3,
    n7c$tim_jobs_real_time = 4,
    n7c$tim_time_since_deadstart = 5,
    n7c$tim_packed_date_and_clock = 6,
    n7c$tim_jobs_srus = 7;

  PROCEDURE [XREF] n7p$time ALIAS 'zn7ptim' (request_code:
    n7c$tim_jobs_cpu_time .. n7c$tim_jobs_srus;
    ptr_to_response_word: ^cell);
```

The details concerning the request codes may be found in the NOS 170 Reference Manual in the System Requests chapter.

The following decks describe the format of returned information.

```
{ ZN7TJCT    Contains type definition of job's cpu time. }

  TYPE
```

```
    n7t$jobs_cpu_time = packed record
      fill: set of 1 .. 3,
      ss: 0 .. 1fffffffffff(16),
      ms: 0 .. 0fff(16),
    recend;
```

```
{ ZN7TJDR    Contains type definition of julian date. }

  TYPE
    n7t$julian_date = packed record
      fill: set of 1 .. 30,
      jd: packed array[1 .. 5] of 0 .. 3f(16),
    recend;
```

```
{ ZN7TJRT    Contains type definition of job's real time. }

  TYPE
    n7t$jobs_real_time = packed record
      fill: set of 1 .. 24,
      seconds_times_4096: 0 .. 0fffffffff(16),
    recend;
```

```
{ ZN7TJST    Contains type definition of job's system time. }

  TYPE
    n7t$jobs_system_time = packed record
      fill: set of 1 .. 24,
      srus: 0 .. 0fffffffff(16),
    recend;
```

```
{ ZN7TPDC    Contains type definition of date and time. }
```

```
  TYPE
    n7t$date_clock = packed record
      fill: set of 1 .. 24,
      year_minus_1970: 0 .. 3f(16),
      month: 0 .. 3f(16),
      day: 0 .. 3f(16),
      hour: 0 .. 3f(16),
      minute: 0 .. 3f(16),
      second: 0 .. 3f(16),
    recend;
```

```
{ ZN7TTSD    Contains type definition of time since deadstart. }

  TYPE
    n7t$time_since_deadstart = packed record
      ss: 0 .. 0ffffff(16),
      ms: 0 .. 0ffffffff(16),
    recend;
```

3.2.12.10 NOS 170 Wait Not Busy

This procedure provides a mechanism for the CYBIL user to interface to the NOS 170 wait not busy process. This allows a user to wait for completion of an I/O operation. The status word is the word 0 bit 0 of the FET. It should be pointed out that waiting for an I/O operation is the most common usage but the status word could be used in other operations such as a memory request. In this case the status word may be any word in program central memory. The procedure reference is available on common deck ZN7PWNB.

```
{ ZN7PWNB    Allows CYBIL user interface to NOS 170 wait not busy. }

  PROCEDURE [XREF] n7p$wait_not_busy ALIAS 'zn7pwnb' (ptr_to_status_word:
    ^cell);
```

This amounts to a RECALL with a status word specified. For further detail see the RECALL explanation in the NOS 170 Reference Manual.

3.2.12.11 NOS 170 Execute Command

This procedure provides a mechanism for the CYBIL user to interface to the NOS 170 execute command statement process. This allows you to execute a command upon completion of your program. It should be pointed out that there is no return process from this procedure.

```
{ ZCYPEXC    Executes the command string and does not return to caller.

  PROCEDURE [XREF] cyp$execute_command ALIAS 'ZCYPEXC'
   (    command: string ( * <= 80));
```

3.2.12.12 NOS 170 Get Job Control Register Value

This procedure allows the CYBIL user to interface to the NOS 170 job control registers. The procedure will get the value from the specified register and return it back to the calling program.

```
{ ZN7PGCR    Returns the value of the specified job control register.

  PROCEDURE [XREF] n7p$get_job_control_register ALIAS 'zn7pgcr'
   (    register_number: 1 .. 3;
    VAR value: 0 .. 1ffff(16));
```

3.2.12.13 NOS 170 Set Job Control Register Value

This procedure allows the CYBIL user to interface to the NOS 170 job control registers. The procedure will set the specified register with the value.

```
{ ZN7PSCR    Sets the specified job control register to the value.

  PROCEDURE [XREF] n7p$set_job_control_register ALIAS 'zn7pscr'
   (    register_number: 1 .. 3;
        value: 0 .. lffff(16));
```

3.3 APPENDIX OF COMMON *CALLC DECKS

The following is an alphabetic list of contents of the common decks referenced by *callc within the procedure interfaces described previously.

```
{ Date request return value. }

TYPE
  ost$date = record
    CASE date_format: ost$date_formats OF
    =osc$month_date=
      month: ost$month_date, { month DD, YYYY }
    =osc$mdy_date=
      mdy: ost$mdy_date, { MM/DD/YY }
    =osc$iso_date=
      iso: ost$iso_date, { YYYY-MM-DD }
    =osc$ordinal_date=
      ordinal: ost$ordinal_date, { YYYYDDD }
    CASEND,
  recend,

  ost$date_formats = (osc$default_date, osc$month_date, osc$mdy_date,
    osc$iso_date, osc$ordinal_date),

  ost$month_date = string (18),
  ost$mdy_date = string (8),
  ost$iso_date = string (10),
  ost$ordinal_date = string (7);




  CONST
    osc$max_name_size = 31,
    osc$null_name = '

  TYPE
    ost$name_size = 1 .. osc$max_name_size;

  TYPE
    ost$name = string (osc$max_name_size);
```

```
*callc osdstr

{ OSDSTAT   Definition of request status record }

  CONST
    osc$max_condition = 999999,
    osc$status_parameter_delimiter = '"';

  TYPE
    ost$status_condition = 0 .. osc$max_condition,
    ost$status = record
      case normal: boolean of
      =FALSE=
        identifier: string (2),
        condition: ost$status_condition,
        text: ost$string,
      casend,
    recend;
```

```
*callc zoststr

{ OSDSTR    type definitions for string }

  CONST
    osc$max_string_size = 256;

  TYPE
    ost$string_size = 0 .. osc$max_string_size;

  TYPE
    ost$string_index = 1 .. osc$max_string_size + 1;

  TYPE
    ost$string = record
      size: ost$string_size,
      value: string (osc$max_string_size),
    recend;
```

```
{ Time request return value. }

TYPE
  ost$time = record
    CASE time_format: ost$time_formats OF
    =osc$ampm_time=
      ampm: ost$ampm_time, { HH:MM: AM or PM }
    =osc$hms_time=
      hms: ost$hms_time, { HH:MM:SS }
    =osc$millisecond_time=
      millisecond: ost$millisecond_time, { HH:MM:SS.MMM }
    CASEND,
  recend,

  ost$time_formats = (osc$default_time, osc$ampm_time, osc$hms_time,
    osc$millisecond_time),

  ost$ampm_time = string (8),
  ost$hms_time = string (8),
  ost$millisecond_time = string (12);
```

```
*callc zuttdcn

{ ZN7TDIR    Type description of NOS 170 library directory entry. }

  TYPE
    n7t$opld_directory_entry = packed record
      record_name: utt$dc_name,
      record_type: 0 .. 3ffff(16),
      fill: set of 1 .. 32,
      random_address: 0 .. 0ffffff(16),
    recend,
    n7t$opld_directory = array[ * ] of n7t$opld_directory_entry;
```

```
*callc zuttdcn
*callc pxiotyp

{ ZN7TFET    Type definition for NOS File Environment Table (FET). }

?? fmt ( format := off ) ??

  TYPE                        { NOS File Environment Table (FET) }
```

```
    n7t$fet = PACKED RECORD
      CASE filename  : utt$dc_name OF              { fet +  0 }

      = 0 = { variation used to clear the n7t$fet }
        fet0     : 0 .. 3FFFF(16),
        fet1_22  : ARRAY[1 .. 22] OF INTEGER,

      = 1 = { main variation, good for most things }
        level_number       : 0 .. 0F(16),
        abnormal_termination  : 0 .. 0F(16),
        eoi                : BOOLEAN,
        request_code       : 0 .. 7F(16),
        binary_operation   : BOOLEAN,
        completed          : BOOLEAN,
        device_type        : 0 .. 0FFF(16),        { fet +  1 }
        random             : BOOLEAN,
        fill0              : SET OF 1 .. 1,
        user_processing    : BOOLEAN,
        error_processing   : BOOLEAN,
        fill1              : SET OF 1 .. 20,
        extension_length   : 0 .. 3F(16),
        first              : ^CELL,
        fill2              : SET OF 1 .. 42,       { fet +  2 }
        next_in            : ^CELL,
        fill3              : SET OF 1 .. 42,       { fet +  3 }
        next_out           : ^CELL,
        fntptr             : 0 .. 0FFF(16),        { fet +  4 }
        fill4              : SET OF 1 .. 12,
        pru_size           : 0 .. 3FFFF(16),
        limit              : ^CELL,
        fill5              : SET OF 1 .. 12,       { fet +  5 }
        fwa_ws             : ^CELL,
        fill6              : SET OF 1 .. 12,
        lwa1_ws            : ^CELL,
        cri                : 0 .. 3FFFFFFF(16),    { fet +  6 }
        rw                 : BOOLEAN,
        rr                 : 0 .. 1FFFFFFF(16),
        fill7              : SET OF 1 .. 24,       { fet +  7 }
        index_length       : 0 .. 3FFFF(16),
        fwa_index          : ^CELL,
        pfn                : utt$dc_name,          { fet +  8 }
        fill8              : SET OF 1 .. 5,
        fa                 : BOOLEAN,
        file_category      : 0 .. 3F(16),
        file_mode          : 0 .. 3F(16),
        optional_un        : utt$dc_name,          { fet +  9 }
        space              : 0 .. 3FFFF(16),
        file_password     : utt$dc_name,          { fet + 10 }
        erad              : ^CELL,
        user_cw           : INTEGER,              { fet + 11 }
        packname          : utt$dc_name,          { fet + 12 }
        fill9             : SET OF 1 .. 6,
        unit              : 0 .. OFFF(16),
        new_file_name     : utt$dc_name,          { fet + 13 }
        fill10            : SET OF 1 .. 18,

      = 2 = { variation used for PASCAL-X IO file descriptor }
        fill11               : SET OF 1 .. 18,
        fill12               : SET OF 1 .. 42,    { fet +  1 }
        first_as_integer     : 0 .. 3FFFF(16),
        fill13               : SET OF 1 .. 42,    { fet +  2 }
        next_in_as_integer   : 0 .. 3FFFF(16),
        fill14               : SET OF 1 .. 42,    { fet +  3 }
        next_out_as_integer  : 0 .. 3FFFF(16),
        fill15               : SET OF 1 .. 42,    { fet +  4 }
        limit_as_integer     : 0 .. 3FFFF(16),
        direct  : RECORD
          current_page  : INTEGER,                { fet +  5 }
          cri_rw_rr     : INTEGER,                { fet +  6 }
          current_word  : INTEGER,                { fet +  7 }
          record_length : INTEGER,                { fet +  8 }
          file_length   : INTEGER,                { fet +  9 }
          last_page     : INTEGER,                { fet + 10 }
        RECEND,
        reserved1  : INTEGER,                     { fet + 11 }
        reserved2  : INTEGER,                     { fet + 12 }
        legible  : RECORD
          column          : INTEGER,              { fet + 13 }
          remaining_chars : INTEGER,              { fet + 14 }
          string_ptr      : INTEGER,              { fet + 15 }
          buffer          : INTEGER,              { fet + 16 }
          codeset         : file_encoding,        { fet + 17 }
        RECEND,
        print  : RECORD
          limit     : INTEGER,                    { fet + 18 }
          line      : INTEGER,                    { fet + 19 }
          page_num  : INTEGER,                    { fet + 20 }
          page_proc : ^PROCEDURE  (               { fet + 21 }
                        print_file   : ^CELL;
                        next_page_#  : INTEGER ),
        RECEND,
        reserved3  : INTEGER,                     { fet + 22 }

      = 3 = { variation used for LFM and PFM interfacing }
```

```
        response_code     : 0 .. 0FF(16),
        fill16            : SET OF 1 .. 10,
        not_mass_storage  : BOOLEAN,                 { fet +  1 }
        fill17            : SET OF 1 .. 59,
        fet2_4            : ARRAY[2 .. 4] OF INTEGER,
        fnt  : PACKED RECORD                         { fet +  5 }
          lfn            : utt$dc_name,
          fill18         : SET OF 1 .. 1,
          extend_only    : BOOLEAN,
          alter_only     : BOOLEAN,
          execute_only   : BOOLEAN,
          fill19         : SET OF 1 .. 1,
          write_lockout  : BOOLEAN,
          file_type      : 0 .. 3F(16),
          fill20         : SET OF 1 .. 1,
          control_point  : 0 .. 1f(16),
        RECEND,
        fst  : PACKED RECORD                         { fet +  6 }
          id_code             : 0 .. 3F(16),
          equipment_number    : 0 .. 3F(16),
          first_track         : 0 .. 0FFF(16),
          current_track       : 0 .. 0FFF(16),
          current_sector      : 0 .. 0FFF(16),
          fill21              : SET OF 1 .. 3,
          file_opened         : BOOLEAN,
          file_written_since_opened   : BOOLEAN,
          file_written        : BOOLEAN,
          fill22              : SET OF 1 .. 2,
          write_read_status   : 0 .. 3,
          last_operation_was_write    : BOOLEAN,
          busy                : BOOLEAN,
        RECEND,
        fet7  : INTEGER,                             { fet +  7 }
        getfnt  : PACKED RECORD                      { fet +  8 }
          nf      : 0 .. 0FFF(16),
          fill23  : SET OF 1 .. 10,
          loft    : BOOLEAN,
          syft    : BOOLEAN,
          faft    : BOOLEAN,
          pmft    : BOOLEAN,
          ptft    : BOOLEAN,
          lift    : BOOLEAN,
          fill24  : SET OF 1 .. 3,
          teft    : BOOLEAN,
          phft    : BOOLEAN,
          prft    : BOOLEAN,
          roft    : BOOLEAN,
          inft     : BOOLEAN,
          fill25   : SET OF 1 .. 3,
          cb       : 0 .. 7,
          ta       : ^CELL,
        RECEND,

      = 4 = { variation used for LFM RENAME and ACCSF functions }
        fill26                 : SET OF 1 .. 18,
        fetl_5                 : ARRAY[1 .. 5] OF INTEGER,
        new_lfn                : utt$dc_name,        { fet +  6 }
        old_statement_count    : 0 .. 3FFFF(16),

      = 5 = { variation used for LFM PSCSF function }
        fill27                 : SET OF 1 .. 18,
        fill28                 : ARRAY[1 .. 5] OF INTEGER,
        fill29                 : SET OF 1 .. 12,     { fet +  6 }
        new_statement_count    : 0 .. 0FFFFFF(16),
        new_word_count         : 0 .. 0FFFFFF(16),

      = 6 = {variation used for LFM FILINFO function }
        len                   : 0 .. 3f(16),        { fet + 0 }
        fill30                : SET OF 1 .. 12,
        dt6                   : 0 .. 0fff(16),      { fet + 1 }
        fill31                : SET OF 1 .. 35,
        permission            : 0 .. 7f(16),
        fill33                : SET OF 1 .. 6,
        fet2                  : integer,            { fet + 2 }
        file_length           : 0 .. 0ffffff(16),   { fet + 3 }
        fill32                : SET OF 1 .. 36,
        fet4                  : integer,            { fet + 4 }

      = 7 .. 3FFFFFFFFFF(16) = { 'unused' variations }
        ,
      CASEND,
    RECEND;

?? fmt ( format := on ) ??




{ ZN7TTSR    Contains type definition of terminal status. }

  TYPE
    n7t$terminal_status = packed record
      tid: 0 .. 3ffffffffff(16),
      sys: 0 .. 3f(16),
      tn: 0 .. 0fff(16),
      fill1: set of 1 .. 24,
      int: ^cell,
      fill2: set of 1 .. 6,
      fill3: set of 1 .. 7,
      tape_mode: boolean,
      duplex: boolean,
      cset: boolean,
      init_cset: boolean,
      parity: boolean,
    recend;




  TYPE
    n7t$exchange_package = record
      abregs: array[0 .. 7] of packed record
        something: 0 .. 3f(16),
        supplement,
        areg,
        breg: - 1ffff(16) .. 1ffff(16)
      recend,
      xreg: array[0 .. 7] of integer,
      raplusone: integer
    recend;



*callc osdname

{ ZOSTNAM    Defines names. }

  CONST
    osc$max_name_length = osc$max_name_size,
    osc$max_nos170_name_length = 7;

  TYPE
    ost$name_types = (clc$nos170_name, clc$short_name, clc$long_name),
    ost$name_length = 1 .. osc$max_name_length,
    ost$nos170_name = string (osc$max_nos170_name_length),
    ost$name_descriptor = record
      typ: ost$name_types,
      length: ost$name_length,
      str: ost$name,
    recend;



*callc osdstr
*callc zcltstr

{ ZOSTSTR    Defines the bounds of strings. }

  CONST
    osc$max_string_length = osc$max_string_size;

  TYPE
    ost$string_length = 0 .. osc$max_string_length;
```

```
{ ZUTTDCN    Type definition for display code name. }

  TYPE
    utt$dc_name = 0 .. 3fffffffff(16);



*callc zuttdcn

{ ZUTTDNV    Type definition for display code name and value. }

  TYPE
    utt$dc_name_and_value = packed record
      dc_name: utt$dc_name,
      value: - 1ffff(16) .. 1ffff(16),
    recend;
```

```
{ ZUTTENC    Defines possible 6 bit character sets. }

  TYPE
    utt$encoding = (utc$ascii64, utc$ascii612);
```

4.0 SYSTEM AVAILABILITY MATRIX

A - Available

NA - Not Applicable

NI - Not Yet Implemented

Feature System	NOS	NOS/BE	NOS/VE
General Procedures			
utp$generate_unique_string	A	A	NI
utp$generate_unique_label	A	A	NI
utp$generate_unique_file_name	A	A	NI

4.0 SYSTEM AVAILABILITY MATRIX

A - Available

NA - Not Applicable

NI - Not Yet Implemented

Feature / System	NOS	NOS/BE	NOS/VE
Mathematical Functions			
cyf$abs	A	NI	NI
cyf$alog	A	NI	NI
cyf$exp	A	NI	NI
cyf$sqrt	A	NI	NI
cyf$xtoi	A	NI	NI
cyf$xtor	A	NI	NI

4.0 SYSTEM AVAILABILITY MATRIX

A - Available

NA - Not Applicable

NI - Not Yet Implemented

Feature System	NOS	NOS/BE	NOS/VE
Data Conversion Procedures			
utp$capitalize_string	A	A	NI
cyp$lowercased_string	A	A	NI
utp$convert_dc_name_to_string	A	A	NA
utp$convert_string_to_dc_name	A	A	NA
cyp$cnvt_str_to_dc_name_blank	A	A	NA
utp$convert_string_to_file_name	A	A	NA
utp$convert_string_to_dc_string	A	A	NA
utp$convert_dc_string_to_string	A	A	NA
utp$convert_integer_to_string	A	A	NI
utp$convert_integer_to_rjstring	A	A	NI
utp$convert_string_to_integer	A	A	NI
utp$convert_string_to_real	A	NI	NI
translation table conversion	A	A	NA
utp$word_to_hexadecimal_string	A	A	NA
utp$word_to_octal_string	A	A	NA
cyp$scanf_n	A	A	NI
cyp$s_scanf_n	A	A	NI

4.0 SYSTEM AVAILABILITY MATRIX

A - Available

NA - Not Applicable

NI - Not Yet Implemented

Feature / System	NOS	NOS/BE	NOS/VE
String & Character Procedures			
utp$compare_strings	A	A	NI
utp$create_dc_string_ptr	A	A	NA
utp$get_next_dc_char	A	A	NA
utp$insert_next_dc_char	A	A	NA

4.0 SYSTEM AVAILABILITY MATRIX

A - Available

NA - Not Applicable

NI - Not Yet Implemented

Feature / System	NOS	NOS/BE	NOS/VE
CYBIL Screen Formatting Procedures			
cyp$close_panel	A	NA	NI
cyp$get_integer	A	NA	NI
cyp$get_real	A	NA	NI
cyp$get_key_value	A	NA	NI
cyp$get_cursor_position	A	NA	NI
cyp$open_panel	A	NA	NI
cyp$position_row	A	NA	NI
cyp$set_cursor_position	A	NA	NI
cyp$read_panel	A	NA	NI
cyp$show_panel	A	NA	NI
cyp$write_panel	A	NA	NI

4.0 SYSTEM AVAILABILITY MATRIX

A - Available

NA - Not Applicable

NI - Not Yet Implemented

Feature System	NOS	NOS/BE	NOS/VE
CYBIL Program Procedures			
osp$initiate	A	NA	NA
osp$terminate	A	NA	NA
osp$terminate_with_message	A	NA	NA
utp$end	A	A	NI
utp$abort	A	A	NI
abort	A	A	NI
utp$clear_and_abort	A	A	NA

4.0 SYSTEM AVAILABILITY MATRIX

A - Available

NA - Not Applicable

NI - Not Yet Implemented

Feature / System	NOS	NOS/BE	NOS/VE
Pointer Manipulation Procedures			
utp$compute_offset_of_pointer	A	A	NI
utp$compute_pointer_for_offset	A	A	NI
CYBIL Overlay Loading	A	NA	NA

4.0 SYSTEM AVAILABILITY MATRIX

A - Available

NA - Not Applicable

NI - Not Yet Implemented

Feature / System	NOS	NOS/BE	NOS/VE
System Utility Procedures			
pmp$get_date	A	A	NI
pmp$get_time	A	A	NI
utp$get_control_statement_args	A	NA	NA
osp$get_control_statement	A	NA	NA
utp$get_user_name	A	NA	NA
utp$issue_dayfile_message	A	A	NA
utp$batch_origin_job	A	NA	NA

4.0 SYSTEM AVAILABILITY MATRIX

A - Available

NA - Not Applicable

NI - Not Yet Implemented

Feature System	NOS	NOS/BE	NOS/VE
Terminal Interrupt Procedures			
utp$init_term_interrupt_detect	A	NA	NA
utp$terminal_interrupt_detected	A	NA	NA
utp$ask_for_direction	A	NA	NA

4.0 SYSTEM AVAILABILITY MATRIX

A - Available

NA - Not Applicable

NI - Not Yet Implemented

Feature System	NOS	NOS/BE	NOS/VE
File System Procedures			
utp$acquire_file	A	NA	NA
utp$is_file_local	A	NA	NA
utp$return_file	A	NI	NI
utp$rewind_file	A	A	NI
n7p$pf_info_message	A	NA	NA
n7p$acquire_file	A	NA	NA
utp$extract_record_from_library	A	NA	NA
n7p$set_record_type	A	NA	NA
utp$is_file_writable	A	NA	NA
n7p$get_opld_directory	A	NA	NA
utp$assign_file_to_terminal	A	NA	NA

4.0 SYSTEM AVAILABILITY MATRIX

A - Available

NA - Not Applicable

NI - Not Yet Implemented

Feature / System	NOS	NOS/BE	NOS/VE
CYBIL to NOS 170 Procedures			
n7p$cio	A	NA	NA
n7p$cpm_with_value	A	NA	NA
n7p$cpm_with_pointer	A	NI	NI
n7p$lfm	A	NA	NA
n7p$issue_dayfile_message	A	NI	NI
n7p$recall	A	NI	NI
n7p$translate_control_statement	A	NA	NA
n7p$advance_control_card	A	NI	NI
n7p$time	A	NI	NI
n7p$wait_not_busy	A	NI	NI
cyp$execute_command	A	NI	NI
n7p$get_job_control_register	A	NI	NI
n7p$set_job_control_register	A	NI	NI