PLUS Source Library Definitions

by

Alan Ballard

Computing Centre

UNIVERSITY OF BRITISH COLUMBIA
6356 Agricultural Road
Vancouver, B.C., Canada V6T 1W5

September 1979
Revised January 1983

Note

This edition of the PLUS library manual corresponds to version 24 of the 470 PLUS compiler (distributed in July, 1982). Revision bars indicate changes since the previous (October, 1981) version.

Table of Contents

A. INTRODUCTION

The file *PLUS.SOURCELIB is a PLUS source library containing many declarations that may be useful when writing PLUS programs. This library is assumed by default if unit 0 is not specified when running the PLUS compiler. If the standard source library is required in addition to a private library, then *PLUS.SOURCELIB should be concatenated to the private library when running the compiler.

Any definitions required from *PLUS.SOURCELIB must be explicitly included with the %Include compiler procedure. In general, each library member contains %Includes for any other library definitions that it requires.

The contents of *PLUS.SOURCELIB fall roughly into the following classes:

1. Definitions for standard run-time support. The standard PLUS execution environment is implemented by a number of PLUS procedures. It is possible to replace some or all of this environment with routines written by the user. The declarations for the standard routines are contained in the library. The object for the standard versions of these routines is in a resident system library.

2. Macros and procedures which provide extensions to the PLUS language. These facilities will be incorporated (in some form) into a future version of the language.

3. Useful common types and constants. Several common types and constants (e.g., "Integer", "Boolean") which are not built-in are defined by declarations in the library.

4. Definitions of a number of 370/470-dependent types, constants, etc.

5. String-handling and conversion routines. A number of useful procedures for processing character strings and for converting between numeric and string values are contained in the resident system. The declarations required to use these routines are in *PLUS.SOURCELIB.

6. Numeric routines. The resident PLUS library also contains some useful simple numeric routines whose declarations are in *PLUS.SOURCELIB.

7. I/O interfacing definitions. *PLUS.SOURCELIB contains several macros and definitions for interfacing with the MTS I/O subroutines.

8. MTS system subroutines definitions. *PLUS.SOURCELIB contains procedure definitions for most of the MTS system subroutines. There are also various types required to describe parameters and results of these routines, and a number of auxiliary macros to simplify the use of some

subroutines.

A listing of any source library member is easily obtained by using the PLUS compiler. To obtain a listing of member "xxxx", run the compiler with input

```
%List := 2;
%Include(xxxx)
```

| The entry points of PLUS procedures in the resident system PLUS library are defined by a "low core symbol table" called CCSYMBOL (for "Coding Conventions Symbols"). PLUS programs using any resident PLUS routines must include appropriate library records to reference this symbol table. In most cases, it is also necessary to reference the predefined pseudo-register definition QLCSPR. The necessary loader records are generated by the compiler if the standard main program definition is used by %Include(Main), as described below, or by setting the compiler option %Library to True. The file *PLUS.ENDJUNK also contains a copy of the usual loader records, which can be copied to the end of an object file if the records were not produced by the compiler, or have been discarded by *LINKEDIT or *OBJUTIL.

B. STANDARD PLUS RUN-TIME SUPPORT

| When entering a PLUS program from another environment, it is necessary to establish the environment expected by PLUS. In most cases this is done by a special "linkage routine" which performs the necessary setup. The resident system contains linkage routines that can be used for main programs written in PLUS (i.e., entered from MTS command mode via $RUN), or for procedure called with an OS S-type linkage as is used by FORTRAN. To use one of these linkage routines, the PLUS procedure being entered must specify it in the procedure declaration LINKAGE phrase.
|
| The linkage procedure used when entering a main program allocates a stack and storage for global variables, processes certain parameters from the PAR field of the $RUN command, and provides a simple program interrupt handler which interprets PLUS run-time error conditions. The standard run-time support should be adequate for most applications. The only run-time support actually required by PLUS is to set up a stack and global storage before entering any PLUS procedure. A number of the pieces of the standard support are defined in *PLUS.SOURCELIB to assist in "roll-your-own" linkage requirements.
|
| See the PLUS User's Manual for further information about PLUS run-time organization, linkage options, and the %Entry and %Library compiler options.

The standard run-time support consists of the following components. The declaration for each is in *PLUS.SOURCELIB (except as noted), and may be Included if you wish to write your own version or use it in a special application.

type Check_Kind_Type

Purpose

Defines the values used for PLUS run-time error trap codes.

Description

This member consists of a PLUS identifier-list type defining the values used for PLUS run-time error codes.

When a run-time error is detected, an invalid operation (operation code of 0) will be executed. The byte following the operation code contains the trap code.

The library member also contains a constant array Check_Name, containing character-string descriptions of the trap kinds.

procedure Main

Purpose

Defines the user main program.

Parameters

1- reference, character (0 to 256)

The PAR string passed from the $RUN command, possibly modified by the linkage routine.

Result

Integer

The result returned is set as the return code from the program.

Description

Procedure Main is written by the user.

%Include(Main) may be used to obtain the definition of a main program, called Main, using linkage routine "PLUSENTR".

This library member also sets the program entry point to

|
|
| "MAIN" and sets the %Library option to True so the loader records required to access the resident PLUS library will be generated.
|
| The library declaration does not specify a stack size, but leaves it up to PLUSENTR to determine the amount of stack to be allocated.
|
| It is not necessary to use this declaration or to call your main program MAIN. You can write an equivalent declaration using any program name you prefer.

linkage procedure PLUSENTR

Purpose

To set up for execution of a PLUS procedure called Main.

Description

| This procedure is a PLUS linkage routine to set up for execution of a PLUS main program. It may be named in the LINKAGE phrase of the procedure declaration for the main program.
|
| The library member Main may be used to include such a declaration. PLUSENTR itself is not defined in *PLUS.SOURCELIB.

This routine allocates a pseudo-register for global variables, and a small stack. It calls Runtime_Initialize to perform further initialization and allocate the actual stack to be used for execution of the main program. It then returns to the main program. When the main program returns, it calls Runtime_Highwater and Runtime_Terminate. The result of Main is returned by PLUSENTR as a return code in register 15.
|
|
| linkage procedure QSACHAIN
|
| Purpose
|
| To retrieve a PLUS execution environment by chasing back the save-area chain.
|
| Description
|
| This procedure is a PLUS linkage routine to set up for execution of a PLUS procedure called from an S-type procedure which was itself called from a PLUS procedure.
|
| The routine is used by specifying QSACHAIN in the linkage phrase of the declaration for the PLUS procedure to be called. There is no definition of QSACHAIN in

*PLUS.SOURCELIB.

The routine retrieves the stack and pseudo-register vector by chasing the savearea exactly one level.

This can be used to define the linkage for procedures called from the MTS system subroutine SDUMP and STDDMP, and probably some others also.

procedure Runtime Initialize

Purpose

To complete initialization for a PLUS main program.

Parameters

1- reference, pointer to character(0 to 256)

A pointer to a character string containing the PAR field of the $RUN command.

2- Integer

The stack size specified in the procedure declaration for the program being entered (or defined subsequently with a DEF record).

Result

pointer to unknown

Address of a stack allocated by the routine.

Description

Runtime_Initialize is called from the linkage routine. It has only a small (256 byte) stack, so should not make nested procedure calls or have many local variables. The global base has been set up before it is called.

It is passed the MTS par-field as a parameter. It must allocate a stack and return the address of the stack allocated. It must also leave the address, size and stack limit and save the global base in the global Runtime_Storage. It may modify the first parameter to point to a different string that is to replace the original PAR field passed to the main program.

The standard support processes the STACK=... and HIGH_WATER parameters and sets up the standard program interrupt exit routine. The stack allocated by the standard support is always "fenced" with a protected half-page at the end to detect most stack overflows.

procedure Runtime_Highwater

Purpose

To process the PLUS HIGH_WATER parameter.

Description

Runtime_Highwater is called when the main program returns, using the stack allocated by initialization. The standard support produces the High_Water mark message if it was requested.

procedure Runtime_Terminate

Purpose

To free the stack allocated by initialization.

Description

Runtime_Terminate is called following the high_water routine. It has the small stack (256 bytes) used for initialization. The standard version frees the stack allocated by initialization.

procedure Runtime_Interrupt_Handler

Purpose

To process program interrupts during execution of a program.

Parameters

1- pointer to Exit_Area_Type

Points to a control block in which registers etc. have been saved. See "Exit_Definitions" for details.

Description

Runtime_Interrupt_Handler intercepts program interrupts (and run-time check errors, which cause an operation exception), and performs the standard PLUS processing.

Note

The macro Set_Exit may be used to set up an alternative program interrupt exit. If the PLUS program was compiled with run-time checks enabled, the exit routine should be prepared to handle any run-time error traps.

global Runtime Storage

Purpose

Contains addresses of stack and global pseudo-register vector.

Description

Runtime_Storage is a global block containing the stack and global pseudo-register vector addresses etc. This is expected to be set up by the initialization. It will be used when stack limit checking is implemented.

type Plus_Linkage_Parameters_Type

Purpose

To describe the "parameters" passed to a linkage routine.

Description

When a PLUS linkage routine is entered, the return address (in R15) provides access to a number of data items that may be of interest to the linkage routine, including the stack size requested, if any, and global size required. These data items can be interpreted as "parameters" passed to the linkage routine. This type definition may be useful in interpreting the parameters.

global Program Interrupt Definitions

Purpose

Defines global storage used by the standard program interrupt handler.

Description

This library member defines a number of global variables used by the standard PLUS runtime support. Generally, they should not be touched by other programs.

C. PLUS LANGUAGE EXTENSIONS

procedure Return_Code

Purpose

To set the return code from a PLUS procedure.

Parameters

1- bit(32)

The return code to be given.

Description

This procedure provides an interim mechanism by which a PLUS procedure may deliver a return code in register 15 to its caller. The procedure wishing to deliver a return code should call Return_Code specifying the value to be returned.

This value will be stored in the save area of the caller in such a way that it will be in register 15 when the caller eventually returns.

Note

It is planned to provide an option of the PLUS return statement to allow specification of the return code. At that time, this procedure will be removed.

macro Setup_Return_From

Purpose

To set up a control block to enable a subsequent return from the current procedure.

Parameters

1- reference, Return_Control_Block_Type

A control block to save information about how to effect the return.

2- name, any type

The name of the return variable of the current procedure. Note that the procedure calling Setup_Return_From must have a result, and its name must be used here. The macro uses this parameter to determine both the location of the result where the result is to be returned, and the number of bytes to return.

Description

See description of Return_From for explanation of how this is used.

type Return_Control_Block_Type

Description

See description of Return_From for explanation of how this is used.

procedure Return_From

Purpose

To return from a previous procedure call.

Parameters

1- reference, Return_Control_Block_Type

A control block, previously initialized with Setup_Return_From, defining how to return.

2- reference, value, unknown

The result to be returned. Note that no type checking is possible here. The variable specified must be the same as the result of the procedure being returned from. Just assignment compatible is not enough -- it must occupy the same number of bytes.

Description

Setup_Return_From and Return_From together provide a form of multi-level procedure return. That is, a routine may force a return, not to the point from which it was called, but to the point from which its caller, or a higher-level caller, was called.

A program which wishes to use this must declare a variable of type Return_Control_Block_Type. This will normally be global; in any case it must be accessible by both the routine to be returned from, and any routine which wants to force such a return.

The routine from which a return is to be forced must initialize the control block, using the macro Setup_Return_From. This will save the return information. Subsequently, another procedure may cause a return, as if from the routine that called Setup_Return_From, by calling procedure Return_From. This is valid only when the original routine has not yet returned. Once it has

returned -- either normally, or because of Return_From, the information in the Return_Control_Block is invalid and the effect of using it is undefined.

It is possible to issue a Return_From from an interrupt routine (timer, attention, or program interrupt), as long as the procedure which initialized the control block has not yet returned. In effect, the interrupt routine can be viewed as being called asynchronously at some point during the execution of the subordinate routine.

The call to Setup_Return_From specifies the return variable of the procedure to be returned from; the call to Return_From specifies a value to be returned. The value returned must be of the same size as the return variable. It should, of course, normally be the same type.

Example

```
%Include(Return_From, Setup_Return_From);
procedure Sub1 is
      procedure
      result Success is Boolean
      end;
procedure Sub2;
variable Rcb is Return_Control_Block_Type;
procedure Main
definition
   ...
   if Sub1()
   then
      ...
   end if;
end Main;
definition Sub1
   Setup_Return_From(Rcb, Success);
   ...
   Sub2();
   return with True
end Sub1;
definition Sub2
   ...
   Return_From(Rcb, Boolean(False));
   ...
end Sub2;
```

If Sub2 calls Return_From, the effect will be to return to the point in procedure Main at which Sub1 was called, a return value of False. (Exactly as if Sub1 had executed a "return with False.)

Note that the return value is given in the form of a display Boolean(False); this is to ensure that the type representation of the constant being returned is that of a Boolean.

D. STANDARD DEFINITIONS

type Boolean

Description

Type Boolean is defined as (False to True).

global Numeric Types

Purpose

To define standard integer numeric types and constants.

Description

This global defines the constants Maximum_Integer, Minimum_Integer, Maximum_Short_Integer and Minimum_Short_Integer.

It defines the types Short_Integer (halfword) and Integer (fullword).

Note

In general, numeric variables in PLUS programs should be defined using an explicit range rather than the general type Integer.

global More Numeric Types

Description

This global defines some additional numeric types that are useful in some situations. The types defined are Positive_Integer, Non_Negative_Integer, Negative_Integer, Non_Positive_Integer, Positive_Short_Integer, Non_Negative_Short_Integer, Negative_Short_Integer, and Non_Positive_Short_Integer.

global Real Types

Purpose

To define some standard real (floating point) types and constants.

Description

This global defines the sizes of some "standard size" floating point types. Since real types are not yet implemented in PLUS, they are defined as bit types currently. The definitions will be replaced when type

real is implemented.

The global defines constants Maximum_Short_Real, Maximum_Long_Real and Maximum_Extended_Real (these constants are the number of decimal digits of precision in three standard-precision reals). It defines types Short_Real, Long_Real and Extended_Real.

global String Types

Purpose

To define some useful string types and constants.

Description

This global defines fixed and varying length character strings of 255 characters. The types defined are used as parameters and results of various library procedures.

Note that PLUS does not restrict character types to this length; however, this is considered to be a reasonable length for many applications.

The types defined are called Fixed_String and Varying_String. The type Varying_String_Structure_Type is also defined as a record that may be equated to a Varying_String if it is desired to access the length and text parts.

The constant Standard_String_Length is defined (as 255), and the type String_Length_Type is defined as a numeric range for the length of these standard string types.

global More String Types

Description

This global is similar to String_Types, but defines fixed and varying strings of up to 32767 characters.

E. MACHINE-DEPENDENT DEFINITIONS

type Bc Mode Psw Type

Description

This type describes the format of a BC-mode PSW for the IBM 370. See the library member, and IBM Principles of Operation for details.

type Ccw Type

Description

This type describes the format of an IBM Channel Command Word. See the library member, and IBM Principles of Operation for details.

type Csw Type

Description

This type describes the format of the IBM Channel Status Word. See the library member, and IBM Principles of Operation for details.

global S370 Interrupt Code Definitions

Description

This global contains constant declarations defining the program interrupt codes for the IBM 370.

The global also defines a constant array Interrupt_Name containing character string descriptions of the interrupt codes. The codes serve as indexes into the array to select the description.

global S370 Opcodes

Description

This global contains constant declarations defining all IBM 370 operation code mnemonics. It is primarily used in conjunction with Inline.

All mnemonics are the standard assembler ones, except that the OR instruction is defined as OR#, since or is a reserved word in PLUS.

constant Opcode Mnemonics

Description

This library member defines a constant array containing the mnemonics for all IBM-370 operation codes. An operation code can be used as an index into the array to select the mnemonic. All array entries are character(4), padded on the right with blanks if necessary.

The member also defines constant array Bc_Mnemonics and Bcr_Mnemonics which contain the branch code mnemonics.

For these arrays, the branch condition mask is used as an index into the array.

global Machine_Carriage_Control_Definitions

Description

This global defines the codes used for machine carriage control and CCW operation codes for the IBM 1403/3211 printers.

See *PLUS.SOURCELIB for details of the constant names used. See IBM documentation for further information.

global Machine_Storage_Types

Purpose

To define machine-dependent constants and storage types.

Description

This global provides definitions of the machine dependent constants Bits_Per_Byte, Bits_Per_Halfword, Bits_Per_Word, Bits_Per_Address, Characters_Per_Word, Maximum_Address and Maximum_Displacement.

It defines the types Byte, Halfword, Fullword, Doubleword

type Psw_Type

Description

This type defines the format of the PSW for the IBM 370. It currently defines only the BC-mode PSW. (See Bc_Mode_Psw_Type.)

global Savearea_Types

Description

This global defines two common savearea formats used various places within MTS.

It consists of types Register_Savearea_Type (an array of 16 fullwords) and Os_Savearea_Type (an 18-word record with the standard OS savearea organization).

F. STRING-HANDLING AND CONVERSION ROUTINES

This section describes various routines for converting to and from character form, and performing other character handling. They can be divided into roughly the following categories.

1. "Input Conversion" routines. These are procedures to convert character strings to various internal forms. The following procedures are in this group:

   Hex_String_To_Bits
   Hex_String_To_Varying
   String_To_Integer
   String_To_Real

   The following are also input conversion routines, but are "lower-level" routines called from the above. These may also be used directly when appropriate.

   Hex_Chars_To_Bits
   Hex_Chars_To_Varying
   Digits_To_Integer
   Construct_Real

2. "Output Conversion" routines. These are procedures which take various internal forms and produce character strings. The following procedures are in this group:

   Address_To_Varying
   Bits_To_Hex_Varying
   Chars_To_Hex_Varying
   Integer_To_Varying
   Line_Number_To_Varying
   Picture_Format
   String_To_Hex_Varying

3. Miscellaneous character handling routines. The following are in this group:

   Append_Varying
   Case_Conversion
   Fill
   Fill_Fixed_String
   Fill_Varying_String
   Pad
   Pad_Varying_String

The descriptions of all these follow, in alphabetical order.

procedure Address_To_Varying

Purpose

To convert an address to a string of the form "symbol+offset" if possible.

Parameters

1- bit(24)

A value to be interpreted as an address. Normally, type cheating will be required to provide this.

Result

Varying_String

A string containing the converted address.

Description

If the address is in a csect of the loaded program, it is returned in the form "symbol+offset", where symbol is the closest entry point, and "offset" is a hexadecimal string giving the offset from the entry point. The "+offset" part is omitted if the address is equal to an entry point address.

If the address is not in a csect of a loaded routine, it is returned as a hexadecimal string.

procedure Append_Varying

Purpose

To append one string to another.

Parameters

1- reference, Varying_String

The string to be appended to.

2- reference, value, Varying_String

The string to append.

Description

The second parameter string is appended to the first. However, if the result would be too long for a Varying_String, it will be truncated to fit.

procedure Bits To Hex Varying

Purpose

To convert a numeric bit-value to a hexadecimal character string.

Parameters

1- bit(32)

A number of bit value to be converted.

2- String_Length_Type (a numeric value)

A field width for the result.

Result

Varying_String

A string of hexadecimal characters.

Description

This routine converts a numeric or bit value to hexadecimal.

The second parameter specifies the length of result wanted. If this is less than eight, the right-most characters of the result are returned. (Thus, for example, a single byte can be converted by specifying an output width of 2.) If the length specified is greater than eight, the value will be converted to eight characters, then padded on the left with blanks.

macro Case Conversion

Purpose

To convert lower-case to upper case.

Parameters

1- name, unknown

The location of the region to be converted.

2- value, numeric

The number of characters to be converted. Must be ð 256.

Description

Uses Inline and the MTS case-conversion table CASECONV to convert a given number of characters from lower to upper case.

Note

If the string to be converted is a varying length character string "x", a call of the form

```
Case_Conversion(Substring(x,0,0), Length(x))
```

may be used.

procedure Chars_To_Hex_Varying

Purpose

To convert a specified number of bytes to hexadecimal characters.

Parameters

1- pointer to value unknown

A pointer to the first byte to be converted.

2- numeric value

The number of bytes to be converted. Must be <= 127.

Result

Varying_String

The hexadecimal representation of the locations specified.

Description

This procedure converts an arbitrary value to a hexadecimal string.

Example

```
variable Char, Res are Varying_String;
   ...
Res := Chars_To_Hex_Varying(
           Address(Substring(Char,0,0)),
           Length(Char));
```

This illustrates how the routine might be used to convert the value of a Varying_String.

Note

See also String_To_Hex_Varying.

procedure Construct_Real

Purpose

To convert a string of decimal digits to a real number.

Note

This is an internal routine used by procedure String_To_Real. It may be called directly when appropriate.

Parameters

1- pointer to Varying_String

A pointer to a string of decimal digits. The string must contain only decimal characters "0" to "9". The result is undefined if other characters are included.

2- Integer

A scale factor to be applied to the number.

3- Boolean

True if the number is to be made negative, false if positive.

4- pointer to Integer

Points to an integer which is set to the number of significant digits in the converted number.

5- pointer to Varying_String

Points to a string which is used to return an error message if the conversion fails.

Null may be used for this parameter to ignore errors and return a default value.

Result

bit(128)

The internal form of an extended-precision real containing the converted number. This will be returned as a value of type real, when real is implemented.

Description

This routine is called with the "parsed" pieces of a real number. These are a string containing a string of digits, with the decimal point assumed to be at the right-hand end, a integer scale factor, and whether or not the number has a negative sign.

It converts the number to an extended precision real.

procedure Digits To Integer

Purpose

To convert a sequence of decimal digits to a binary integer.

Note

This is an internal routine used by procedure String_To_Integer. It may be called directly when appropriate.

Parameters

1- pointer to unknown

A pointer to the first digit to be converted.

2- String_Length_Type (a numeric value)

The number of digits to be converted. The specified number of bytes, starting at the position specified by the first parameter, must contain only decimal characters "0" to "9". The result is undefined if it contains any other characters.

3- Boolean

True if the number is to be made negative, False if positive.

4- pointer to Varying_String

Points to a string which is used to return an error message if the conversion fails because the number is out of the machine range.

If Null is used for this parameter, no error indication will be returned, and a default value will be returned.

Result

Integer

The converted value.

Description

This routine is used to convert the string of digits to a binary integer. It differs from String_To_Integer in that it does not check for valid digits, and is passed a pointer and length rather than the string.

macro Fill

Purpose

To fill an area of memory with a specified byte-value.

Parameters

1- name, unknown

The first location to be filled.

2- value, numeric

The number of bytes to be filled.

3- value, bit(8)

Value to be used to fill the specified locations. May be any type compatible with bit(8).

Description

This macro uses Inline to fill an arbitrary number of bytes of memory with the specified value.

macro Fill Fixed String

Purpose

To fill a fixed-length character string with a given value.

Parameters

1- name, character(n)

A fixed-length character string.

2- value, bit(8)

Value to be used to fill the specified string. May be any type compatible with bit(8).

Description

This macro just uses the macro Fill to initialize the string variable to the given value.

Example

```
variable Char is character(10);
  ...
Fill_Fixed_String(Char," ");
```

This would initialize the string to all blanks.

macro Fill Varying String

Purpose

To initialize a Varying_String with a given value.

Parameters

1- name, Varying_String

A string to be initialized.

2- value, numeric

The length of the string to be assigned.

3- value, bit(8)

Value to be used to initialize the string. May be any type compatible with bit(8).

Description

This macro just uses the macro Fill to initialize the string variable to the specified length, using the given value for each character.

Example

```
variable Char is Varying_String'
  ...
Fill_Varying_String(Char,80," ");
```

This would initialize the variable to a string of 80 blanks.

procedure Hex Chars To Bits

Purpose

To convert a sequence of hexadecimal characters to a numeric binary value.

Parameters

1- pointer to value unknown

The address of the first character to be converted.

2- (0 to 8)

The number of characters (starting at location specified by parameter 1) to be converted.

The locations specified by the first two parameters must consist only of valid hexadecimal characters "0" to "9", "A" to "F", or "a" to "f".

3- pointer to Varying_String

A string in which to return an error message if the conversion fails (due to invalid characters).

If Null is specified, the routine will return without any error indication.

Result

Integer

The converted value.

Description

This procedure converts the locations specified by the first two parameters to a numeric value and returns it as a bit-type.

Note

See also Hex_String_To_Bits.

procedure Hex Chars To Varying

Purpose

To convert a sequence of hexadecimal characters to a character string.

Parameters

1- pointer to value unknown

The address of the first character to be converted.

2- String_Length_Type (a numeric value)

The number of characters (starting at location specified by parameter 1) to be converted.

The locations specified by the first two parameters should consist only of valid hexadecimal characters "0" to "9", "A" to "F", "a" to "f", with optional interspersed commas and blanks allowed.

3- pointer to Boolean

A flag which is set False if the conversion fails (due to invalid characters). In this case, an error message is returned as the value of the procedure instead of the converted value.

If Null is specified, the routine will return the error message without an indication that an error occurred.

| 4- reference, Varying_String
|
| The converted value.

Description

This procedure converts the locations specified by the first two parameters to a character value which is returned in the string passed as the fourth parameter.

Note

See also Hex_String_To_Varying.

procedure Hex String To Bits

Purpose

To convert a sequence of hexadecimal characters to a numeric binary value.

Parameters

1- value, Varying_String

A string containing at most 8 hexadecimal characters (digits 0-9, letters a-f, or A-F).

2- pointer to Varying_String

A string in which to return an error message if the conversion fails (due to invalid characters).

If Null is specified, the routine will return without any error indication.

Result

Integer

The converted value.

Description

This procedure converts the string specified by the first parameters to a numeric value and returns it as a bit-type.

Note

See also Hex_Chars_To_Bits.

procedure Hex_String_To_Varying

Purpose

To convert a string of hexadecimal characters to their internal value.

Parameters

1- value, Varying_String

A string containing a sequence of hexadecimal characters (digits 0-9, letters a-f or A-F), with optionally interspersed blanks or commas.

2- pointer to Boolean

If non-null, set to indicate the success or failure of the conversion.

Result

Varying_String

The converted value.

Description

The hex characters from the first parameter are converted to a character string (two hex characters per byte) and returned.

If the parameter contains invalid hexadecimal characters, an error message is returned as the result, and the second parameter is set to false.

Note

See also Hex_Chars_To_Varying.

procedure Integer_To_Varying

Purpose

To convert an integer to a character string.

Parameters

1- value, Integer

The number to be converted.

2- String_Length_Type (a numeric value)

The desired field width of the result.

Result

Varying_String

The character string representation.

Description

The number is converted to EBCDIC and returned as a string. It will be padded on the left with blanks to the specified field width. If it will not fit in the specified width, it will be returned in the smallest number of positions in which it fits. Thus a field width of 0 can be used to convert to the minimum length character representation.

| procedure Line_Number_To_Varying
|
| Purpose
|
| To convert an integer to the external representation for
| an MTS file line number.
|

Parameters

1- Integer

The internal representation of the line number to be converted.

2- String_Length_Type (a numeric value)

The field width in which to generate the line number representation.

Result

Varying_String

The external representation of the line number.

Description

The integer given is converted to the external format for MTS file line numbers, namely

nnnnnnnn.nnn

Trailing zeros after the decimal point are suppressed. Leading zeros are suppressed. A maximum of ten digits is printed in the specified field width. If the specified field width is greater than the length of the representation, it is padded on the left with blanks. The representation is never truncated, regardless of the field width.

procedure Pad

Purpose

To pad a string with blanks.

Parameters

1- reference, Varying_String

The string to be padded.

2- String_Length_Type (a numeric value)

The length to pad it to.

Description

The given Varying_String is padded on the right with blanks to the specified length. If it is already longer, it is left unchanged.

macro Pad Varying String

Purpose

To pad a Varying_String with a specified character.

Parameters

1- name, Varying_String

A Varying_String that is to be padded.

2- value, numeric

The length to which the string is to be expanded.

3- value, bit(8)

A byte value to be used for fill characters added to the string.

Description

This macro uses the given fill character to expand the specified string from its previous length to the length specified by parameter 2.

procedure Picture Format

Purpose

To convert a binary integer to a character string according to a picture format.

Parameters

1- value, Integer

The numeric value to be converted.

2- value, Varying_String

A character string containing a "picture" describing the conversion.

3- pointer to Boolean

If non-null, the Boolean variable pointed at is set to indicate success (True) or failure (False) of the conversion. If set false, an error message is returned as the value of the procedure.

Result

Varying_String

The result returned contains either the converted integer of an error message.

Description

This routine can be used to convert numeric values to many different forms of character representation. The form of the output string is described by a "picture" specification similar to those of Cobol or PL/1.

The number is first converted to a sequence of digits, which is then edited under control of the picture. Pictures can be used to effect scaling, left-or-right zero suppression, comma and decimal point insertion, fixed or varying field width of the converted item.

Pictures

A picture is sequence of characters describing the format desired for the converted string. The characters forming the picture may be any of the following:

9	specifies the position is to be occupied by a digit.
blank	used in place of "9" to indicate replacement of leading or trailing zeros with blanks.
Z, z	used in place of "9" to indicate suppression of leading or trailing zeros.
.	specifies literal insertion of a ".", if the position is followed by a digit. That is, decimal does not appear if it is passed over by right zero suppression. If it is preceded or followed by a blank, it will be replaced by a blank.
D, d	specifies literal insertion of a "." even if there is no following character.
,	specifies literal insertion of a comma; suppressed if not both preceded and followed by a digit. If it is preceded or followed by a blank, it will be replaced by a blank.
V, v	indicates the position at which to align the decimal point of the number being converted (i.e., the right-hand end of the number). If this is omitted, it is assumed to be at the right-hand end of the picture. The "V" has the effect of scaling the value.

P, p is used to allow a "V" to appear past the last digit character of the picture. "P"'s may appear only at the right of the picture. They have the effect of discarding the right-most digits.

A valid picture may have a format like

(Z)(blank)(9)Ý.(9)(blank)(Z)(P)¨

where (?) indicates 0 or more occurrences of the "?", and everything in Ý¨ is optional. Commas may appear anywhere in the picture, and a "D" may appear instead of ".". One "V" may appear, with the restriction that "Z", blank, and "P" are not allowed to the right of the "V".

If the number is negative, a sign will be placed in the right-most unused "Z" or " " position left of the decimal. The pattern must provide at least one "Z" or blank if the number may be negative.

Examples

The general form of a call is

```
%Include(Picture_Format);
    ...
Str := Picture_Format(<value>,<picture>,Address(<boolean
```

The following table indicates the results returned for some possible combinations of value and picture.

value	picture	result
123456	"999999"	"123456"
123456	"ZZZZZ9"	"123456"
123456	"ZZ,ZZ9.99"	"1,234.56"
123456	" 9.ZZZ"	" 123.456"
123456	"Z .ZZZ"	" 123.456"
123456	""	see note 1
123456	"ZZZZ"	see note 1
123456	" .PPPV"	" 123 "
0	"999999"	"000000"
0	"ZZZZZ9"	"0"
0	"Z,ZZZ,9.99"	"0.00"
0	" 9.ZZZ"	" 0"
0	"Z .ZZZ"	" "
0	""	""
0	"ZZZZ"	""
0	" .PPPV"	" "
-1	"999999"	see note 2
-1	"ZZZZZ9"	"-1"
-1	"Z,ZZZ,9.99"	"-0.01"
-1	" 9.ZZZ"	" -0.001"
-1	"Z .ZZZ"	" -.001"
-1	""	see note 1
-1	"ZZZZ"	"-1"
-1	" .PPPV"	" - "
3000	" 9.999"	" 3.000"
3000	" 9. "	" 3 "
3000	"ZZZZZZZ9DZZZ"	"3."
3000	"ZZZZZZZ9.ZZZ"	"3"
-123	"999V"	see note 2
-123	"999"	see note 2
-123	" 999"	"-123"
-123	"Z999"	"-123"

Note 1: The returned value is the error message "Picture_Format: number too big for picture".

Note 2: The returned value is the error message "Picture_Format: no room for sign".

procedure String To Hex Varying

Purpose

To convert a string to hexadecimal characters.

Parameters

1- value, Varying_String

A string to be converted to a sequence of hexadecimal characters.

Result

Varying_String

The hexadecimal representation of the string.

Description

This procedure converts an arbitrary string to a hexadecimal string.

Example

```
variable Char, Res are Varying_String;
   ...
Res := String_To_Hex_Varying(Char);
```

Note

See also Chars_To_Hex_Varying.

procedure String_To_Integer

Purpose

To convert a character string to an integer.

Parameters

1- value, Varying_String

A string containing the EBCDIC representation of an integer in the range allowed for type Integer. It may consist of a string of digits, optionally preceded and/or followed by blanks. An optional plus or minus sign may precede it.

2- pointer to Varying_String

Varying_String used to indicated the success or failure of the conversion.

Result

Integer

The converted value.

Description

If the first parameter contains a valid integer representation, then it is returned and the second parameter is set to "". Otherwise, the smallest possible integer value is returned and the second parameter is set to an error message.

A null pointer may be passed for the second parameter, in which case no error indication will be returned.

procedure String_To_Real

Purpose

To convert a character string to a floating-point (PLUS type real) value.

Note

Since real is not yet implemented in PLUS, this routine currently returns a bit-type result.

Parameters

1- value, Varying_String

A string containing the EBCDIC representation of a real constant in the range valid for the machine. It consists of a sign, integer part, decimal point, fractional part, and exponent part. Each of these pieces is optional. The exponent part consists of the letter "E" (or "e"), an optional sign, and an integer. Leading and/or trailing blanks are also allowed.

Note that the format is similar to that of Fortran real constants, except that the exponent is always introduced with "E".

2- pointer to Varying_String

Varying_String used to indicated the success or failure of the conversion.

Result

bit(128)

The converted value as an extended-precision real.

Description

If the first parameter contains a valid real representation, then it is returned and the second parameter is set to "". Otherwise, a value of 0.0 is returned and the the second parameter is set to an error message.

A null pointer may be passed for the second parameter, in which case no error indication will be returned.

G. NUMERIC ROUTINES

procedure Log2

Purpose

To compute integer base 2 "logarithms".

Parameters

1- Integer

Result

Integer

A number in the range 0 to 31.

Description

Given a parameter n1, returns the smallest number n2 such that 2**n2 > n1. (That is, returns the number of bits in the binary representation of n1.)

procedure Power2

Purpose

To compute integer powers of 2.

Parameters

1- (0 to 31)

Result

Integer

Description

Given parameter n1, returns 2**n1.

procedure Random_Integer

Purpose

To generate sequences of pseudo-random numbers.

Parameters

1- reference, Integer

A "seed" for the random number generator.

2- value, Integer

The desired upper bound, ub, for the random numbers.

Result

Integer

A number in the range 1 to ub (inclusive).

Description

If called with the ub parameter = 0, the random number generator is seeded from the time-of-day clock; otherwise the next random number in sequence is generated.

Note that the "seed" parameter must be maintained unchanged between calls. Thus it must not be a local variable in a scope that is exited between calls.

procedure Round_Down

Purpose

To truncate an integer down to the next lowest multiple of another number.

Parameters

1- Integer

The number to be rounded down.

2- Integer

The number of which a multiple is required.

Result

Integer

procedure Round_Up

Purpose

To round an integer up to the nearest multiple of another number.

Parameters

1- Integer

The number to be rounded up.

2- Integer

The number of which a multiple is required.

Result

Integer

procedure Shift_Left

Purpose

To shift a word left a given number of bits.

Parameters

1- Fullword

The value to be shifted.

2- (-32 to 32)

The number of bits to shift by.

Result

Fullword

Description

Performs a logical shift of the first parameter. If the second parameter is negative, the value is shifted right instead of left.

procedure Shift_Right

Purpose

To shift a word right a given number of bits.

Parameters

1- Fullword

The value to be shifted.

2- (-32 to 32)

The number of bits to shift by.

Result

Fullword

Description

Performs a logical shift of the first parameter. If the second parameter is negative, the value is shifted left instead of right.

H. I/O INTERFACING DEFINITIONS

The source library contains definitions and macros to simplify the use of MTS I/O from PLUS programs.

A PLUS program can perform I/O at one of three levels:

1. By using a set of macros which read and write character strings from appropriate MTS logical units. These macros build the required parameter lists for calling MTS I/O subroutines.

   The macros in this group are:

   Sprint_String
   Sprint_Varying
   Sercom_String
   Sercom_Varying
   Spunch_String
   Spunch_Varying
   Scards_Varying
   Guser_Varying

2. By using a set of macros which take as one parameter a control block of type Mts_File_Type. The control block is used to contain various parameters which must be pre-initialized. Macros are also provided to set fields of the control block.

   These macros and types allow full access to the MTS I/O facilities. The macros simplify the job of calling the procedures, and improve the efficiency by requiring parameters to be initialized only once.

   The macros in this group are

   Free_File
   Initialize_File
   Initialize_File_With_Name

```
Initialize_File_With_Unit#
Read_File
Write_File
Read_Varying
Write_Varying
Write_String
Read_Record
Write_Record
Set_Buffer
Set_Specific_Line
Set_Next_Line
Set_First_Line
Set_Last_Line
```

3. By using the system subroutines Read, Write, etc., directly. Definitions of all the I/O subroutines are provided by *PLUS.SOURCELIB. They are described in the next section, "MTS System Facilities".

The I/O macros are described below, in alphabetical order.

| macro Free_File
|
| Purpose
|
| To free the FDUB in a control block which has been initialized via Initialize_File_With_Name.
|
| Parameters
|
| 1- name, Mts_File_Type
|
| The control block containing the FDUB to be freed.
|
| Note
|
| An error will result if the control block was initialized using a logical I/O unit name or number, rather than an explicit file name. Only control blocks initialized via Initialize_File_With_Name are valid parameters.

macro Guser_Varying

Purpose

To read a string from logical unit GUSER.

Parameters

1- name, Varying_String

A variable into which the line will be read.

2- name, any numeric type

A variable to which the return code from the read operation will be assigned.

Note

A line is read using the @MAXLEN modifier to prevent reading data longer than the maximum length for a Varying_String.

macro Initialize_File

Purpose

To initialize some fields of an Mts_File_Type record.

Parameters

1- name, Mts_File_Type

The control block to be initialized.

2- character(8)

A logical unit name to be used.

3- bit(32) or numeric value

The modifiers to be used for operations on the file.

Note

This macro should only be used to initialize control blocks which are to correspond to named MTS I/O units. Control blocks which are to correspond to numbered units or explicitly named files must be initialized using Initialize_File_With_Unit# and Initialize_File_With_Name, respectively.

macro Initialize_File_With_Name

Purpose

To initialize an Mts_File_Type record, making it correspond to an explicitly named file.

Parameters

1- name, Mts_File_Type

The control block to be initialized.

|
|
| 2- Varying_String
|
| The file name to be used.
|
| 3- bit(32) or numeric value
|
| The modifiers to be used for operations on the file.
|
| 4- name, Integer
|
| A location into which the return code from Getfd may be
| placed.
|
|
| macro Initialize_File_With_Unit#
|
| Purpose
|
| To initialize an Mts_File_Type record, making it
| correspond to a numbered I/O unit.
|
| Parameters
|
| 1- name, Mts_File_Type
|
| The control block to be initialized.
|
| 2- numeric, 0 to 19
|
| A logical unit number to be used.
|
| 3- bit(32) or numeric value
|
| The modifiers to be used for operations on the file.
|
| Note
|
| This macro should only be used when a control block is
| desired for a numbered I/O unit. To initialize a control
| block with a named I/O unit, use Initialize_File.

    type Mts_File_Type

    Purpose

       To keep track of the parameters and status of I/O on an
       MTS logical unit or FDUB.

    Description

       MTS_File_Type is a control block in which the parameters
       and status information for I/O on a logical unit or Fdub
       is kept.

       Normally, a variable of type MTS_File_Type is declared for

each I/O stream used. These will usually be global variables. The macro Initialize_File can be used to initialize fields of the record. Other macros are provided to perform I/O on a given file, and to set various parameters.

The definition of this record uses definitions of various types defined in global Mts_Io_Types (described below). The record contains fields File_Modifiers, File_Line_Number and File_Unit, with obvious usage. File_Buffer may be set to the address of a buffer to be used by Read_File and Write_File. File_Length is a record (of type Mts_Io_Length_Type) consisting of three halfwords (for use with @MAXLEN). File_Simple_Length is equated to the first of these three. Field Last_Return_Code is set to the return code for each I/O operation performed. Last_Result is the result of the last operation, significant only if @NOTIFY or @NOPROMPT was specified.

macro Read File

Purpose

To read a line using a given Mts_File_Type control block.

Parameters

1- name, Mts_File_Type

Description

Calls READ, using parameters which must have been previously set in the control block. The return code and result (notification etc., if requested) are left in the control block.

macro Read Record

Purpose

To read a line into an arbitrary variable.

Parameters

1- name, Mts_File_Type

2- name, unknown

The location into which the record is to be read.

Description

Calls READ to input a value for the given variable. All other parameters must have been previously set in the Mts_File_Type control block. The return code and result (notification etc., if requested) are left in the control block.

The maximum-length parameter will be set to the size of the variable (parameter 2) before calling READ. It is up to the caller to set the @MAXLEN modifier in the control block before the call, if @MAXLEN is desired.

macro Read_Varying

Purpose

To read a string using a given Mts_File_Type control block.

Parameters

1- name, Mts_File_Type

2- name, Varying_String

A variable into which a line is to be read.

Description

Calls READ to input a string into the given Varying_String. All other parameters must have been previously set in the control block. The return code and result (notification etc., if requested) are left in the control block.

The maximum-length parameter will be set to the size of a Varying_String before calling READ. It is up to the caller to set the @MAXLEN modifier in the control block before the call, if @MAXLEN is desired.

macro Scards_Varying

Description

This is similar to Guser_Varying, except that logical unit SCARDS is used.

macro Sercom String

Purpose

To output a character string to logical unit SERCOM.

Parameters

1- string value

The string to be printed. This may be any string expression, provided the length is ð 255.

Description

The string is copied to a Varying_String, which is then output with all default modifiers.

macro Sercom Varying

Purpose

To output a Varying_String to logical unit SERCOM.

Parameters

1- name, Varying_String

The string to be printed.

Description

The string is output with all default modifiers.

Note

The difference between Sercom_String and Sercom_Varying is that Sercom_String copies the expression to a variable to be passed to SERCOM, while Sercom_Varying requires a variable (name) as its parameter.

macro Set Buffer

Purpose

To set the address of the buffer to be read or written by Read_File and Write_File.

Parameters

1- name, Mts_File_Type

2- name, unknown

The location to read to or write from.

Note

The maximum-length parameter will be set to the size of a variable specified. It is up to the caller to set the @MAXLEN modifier in the control block if @MAXLEN is desired.

macro Set_First_Line

Purpose

To set an Mts_File_Type to access the first line of the file next.

Parameters

1- name, Mts_File_Type

Description

This macro gets the line number of the first line of the file, and sets the @INDEXED modifier to cause it to be accessed by the next I/O operation. Note that the modifier must be explicitly reset if it is intended to then read sequentially from that point.

macro Set_Last_Line

Description

This is similar to Set_First_Line, with the obvious difference.

macro Set_Next_Line

Purpose

To set an Mts_File_Type record for sequential I/O.

Parameters

1- name, Mts_File_Type

Description

This macro just turns off the @INDEXED modifier.

macro Set_Specific_Line

Purpose

To cause the next operation on a given file to read or write a specific line.

Parameters

1- name, Mts_File_Type

2- value, integer

The internal line number of the line to be accessed.

Description

This macro sets the line number parameter and the @INDEXED modifier.

Note

If it is desired to continue reading sequentially from the given line, the modifier must be reset after reading the specified line.

macro Sprint_String

Description

This is similar to Sercom_String, except that logical unit SPRINT is used.

macro Sprint_Varying

Description

This is similar to Sercom_Varying, except that logical unit SPRINT is used.

macro Spunch_String

Description

This is similar to Sercom_String, except that logical unit SPUNCH is used.

macro Spunch Varying

Description

This is similar to Sercom_Varying, except that logical unit SPUNCH is used.

macro Write File

Purpose

To write a line using a given Mts_File_Type control block.

Parameters

1- name, Mts_File_Type

Description

Calls WRITE, using parameters which must have been previously set in the control block. The return code and result (notification etc., if requested) are left in the control block.

macro Write Record

Purpose

To write out an arbitrary variable to an MTS file or device.

Parameters

1- name, Mts_File_Type

2- name, unknown

The variable to be written.

Description

Calls WRITE to output the given variable. The size of the type of variable determines the number of bytes to be written. All other parameters must have been previously set in the Mts_File_Type control block. The return code and result (notification etc., if requested) are left in the control block.

macro Write_String

Purpose

To write a string using a given Mts_File_Type control block.

Parameters

1- name, Mts_File_Type

2- value, string

An arbitrary string expression, of length ð 255, to be written.

Description

Calls WRITE to output the given string. All other parameters must have been previously set in the control block. The return code and result (notification etc., if requested) are left in the control block.

macro Write_Varying

Purpose

To write the value of a Varying_String using a given Mts_File_Type control block.

Parameters

1- name, Mts_File_Type

2- name, Varying_String

A variable whose value is to be written.

Description

Calls WRITE to output the given string, using its current length. All other parameters must have been previously set in the control block. The return code and result (notification etc., if requested) are left in the control block.

I. MTS SYSTEM FACILITIES

This section outlines the definitions for MTS System Subroutines that are included in *PLUS.SOURCELIB.

The library includes members defining most of the system

procedures, and some additional macros, types, and constants that are useful in interfacing to these procedures. Most procedure declarations include constant declarations for return codes or parameters, which are provided when the procedures are %Included. While the constants are not named herein, their names may be determined by looking at the appropriate library members.

Many system subroutines involve input-output facilities. The member Mts_Io_Types defines several types which are used consistently in the definitions of these subroutines. The types defined include Mts_Io_Unit_Type, which is defined as a record which may contain a Fdub pointer, an 8-character logical unit name, or an integer logical unit number. This type is used in numerous system subroutines. The global Io_Subroutine_Return_Codes provides constants for the return codes from various input-output subroutines.

Note that most of the system subroutines require an OS S-type linkage sequence; hence the parameter definitions use PLUS reference parameters. The parameters are defined to be "reference value" parameters in cases where the subroutine will not change the value of the parameter. This allows the use of a constant in places where it is safe. Expressions, however, cannot be used for reference parameters.

This section of the writeup is intended to supplement, not replace, MTS Volume 3 and/or UBC System Subroutines.

procedure Attntrp

Purpose

To allow control to be returned to the user on an attention interrupt.

Parameters

1- system procedure

A procedure to be called when an attention interrupt occurs, or Null.

2- reference, unknown

An area to save the registers and PSW when the interrupt occurs. This will normally be a variable of type Mts_Exit_Area_Type. (See Exit_Definitions.)

Description

See MTS Volume 3

Note

The macro Set_Exit may be used to set up a PLUS procedure as an attention-exit routine.

procedure Blokletr

Purpose

To convert a character string into block letters.

Parameters

1- reference, value, unknown

A variable containing the string of characters to be converted. Normally, a PLUS fixed-length string type.

2- reference, value, Integer

A number from 1 to 12 specifying which line of the block-letter form is to be returned.

3- reference, unknown

The place to put the line of block-letters, which must be at least 14 times the length of the string to be converted. Normally a PLUS fixed-length string type will be used.

4- reference, value, Integer

The length of the string to be converted.

Description

See MTS Volume 3

procedure Bsrf

Purpose

To backward space records (lines) in a line file or sequential file.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the file or device.

2- reference, value, Integer

Specifies the number of records to skip.

Description

See MTS Volume 3

Note

Constants for the return codes of this procedure are in the global Fsrf_Bsrf_Return_Codes, which is included along with it.

procedure Canreply

Purpose

To determine whether the user is running in conversational mode or batch mode.

Result

This routine sets the return code instead of returning a result. The constants Can_Reply and Cannot_Reply are provided for comparison with the return code.

Description

See MTS Volume 3

global Carriage_Control_Characters

Purpose

To define mnemonic constants for the carriage control characters.

Description

The following constants are defined.

Double_Space_Cc ("0")
First_Line_Cc (";")
Half_Page_Cc ("2")
Hang_On_Eol_Cc ("&") (for terminals only)
Page_Bottom_Cc ("<")
Page_Front_Cc (":")
Page_Top_Cc ("1")
Quarter_Page_Cc ("4")
Single_Space_Cc (" ")
Single_Space_Noskip_Cc ("9")
Triple_Space_Cc ("-")

procedure Cfdub

Purpose

To determine whether two FDUB-pointers, logical I/O unit numbers, or logical I/O unit names refer to the same file or device.

Parameters

Two references of type value Mts_Io_Unit_Type specifying the items to be compared.

Result

The return code is set rather than a result.

Description

See MTS Volume 3

procedure Charge

Purpose

To computer the charge for the given quantities of resources using the current rates for the signed on CCid.

Parameters

1- reference, value, Integer

The number of elements of the second (and possibly third) parameter(s) to be used.

2- reference, value, array (1 to 14) of Integer

An array specifying amounts of resources for which the charge is to be calculated.

3- reference, value, array (1 to 14) of Integer

An array specifying what resources the amounts in the second parameter represent.

Result

Integer

The amount that would be charged for the indicated resources in centicents. (ten-thousandths of a dollar).

procedure Chgfsz

Purpose

To change the size or maxsize of a file either absolutely or incrementally.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the file to be changed.

2- reference, value, Integer

The size or increment in pages.

3- reference, value, Integer

Flag specifying how size is to be changed.

Description

See MTS Volume 3

Note

This library member also defines constants that may be used as the third parameter. The constants defined are

```
Absolute_Size (0)
Change_In_Size (1)
Absolute_Maxsize (2)
Change_In_Maxsize (3)
```

procedure Chgmbc

Purpose

To change dynamically the number of page-sized buffers used by the file system to read and write a particular file.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the file whose buffer allocation is to be changed.

2- reference, value, Integer

The maximum number of buffers to use.

Description

See MTS Volume 3

procedure Chgxf

Purpose

To change the expansion factor of a file.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the file to be changed.

2- reference, value, Integer

The new expansion factor.

Description

See MTS Volume 3

procedure Chkacc

Purpose

To determine the access that a signon ID, project number, and program key "triple" has to a particular file.

Parameters

1- reference, value, unknown

Location containing the file name, with a trailing blank. Normally, this will be a PLUS fixed-length string type.

2- reference, value, unknown

Location containing the CCID, project, and program key (with a trailing blank) for which access is to be checked. This will normally be a PLUS fixed-length string type or a record type.

Result

bit(32)

Specifies the access. The possible values (or'd together) are described by Mts_File_Access_Codes.

Description

See MTS Volume 3

procedure Chkfdub

Purpose

To obtain a FDUB-pointer for a specified logical I/O unit; to verify that a given FDUB-pointer is legal.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the logical unit or Fdub.

Result

optional, Mts_Fdub_Type

The returned Fdub.

Note that the return code indicates whether an invalid Fdub or unassigned logical unit was specified.

Description

See MTS Volume 3

procedure Chkfile

Purpose

To determine whether a file exists, as well as what access the calling program has to the file.

Parameters

1- reference, value, unknown

Location containing the file name, with a terminating blank. This will usually be a PLUS fixed-length string type.

Result

optional, bit(32)

Specifies the access allowed. The possible values are described by Mts_File_Access_Codes.

Note that the return code specifies whether the file exists.

Description

See MTS Volume 3

procedure Closefil

Purpose

To close a file and release its file buffers.

Parameters

1- reference, value, MTS_Io_Unit_Type

Specifies the logical unit or Fdub.

Description

See MTS Volume 3

| procedure Command
|
| Purpose
|
| To execute an MTS command from a program and return to the program after the command has been executed.
|
| Parameters
|
| 1- reference, value, unknown
|
| A location containing the MTS command. This will usually be a PLUS fixed-length string type.
|
| 2- reference, value, Integer
|
| The length of the command.
|
| 3- reference, value, bit(24)
|
| A series of switches indicating whether the command and any output generated by the command is to be echoed. The two sets of constants, Command_Echo_Never and Command_Echo_Always, and Command_Commentary_Never and

|
|
| Command_Commentary_Always are supplied. A zero value
| indicates that the routine is to follow the setting of the
| MTS echo switch.
|
| Description
|
| See MTS Volume 3

procedure Cmd

Purpose

To execute an MTS command from a program and return to the program after the command has been executed.

Parameters

1- reference, value, unknown

A location containing the MTS command. This will usually be a PLUS fixed-length string type.

2- reference, value, Integer

The length of the command.

Description

See MTS Volume 3

Example

The following program segment issues the MTS command "$DISPLAY VMSIZE".

```
%Include(Cmd);
   ...
constant Display_Cmd is "$DISPLAY VMSIZE";
   ...
Cmd(Display_Cmd, Length(Display_Cmd));
   ...
```

Note that, in this example, Length(...) can be used because it is a constant, and the parameter allows a value.

procedure Cmdnoe

Purpose

To execute an MTS command from a program and return to the program after the command has been executed.

Parameters

1- reference, value, unknown

A location containing the MTS command. This will usually be a PLUS fixed-length string type.

2- reference, value, Integer

The length of the command.

Description

See MTS Volume 3

external variable Cnfginfo

Purpose

To obtain information about the system on which the program is running.

Description

CNFGINFO is a system table containing various items about the current system. The PLUS definition describes it as a complex PLUS record definition.

See the contents of the library member Cnfginfo, and MTS Volume 3, for details.

procedure Cntlnr

Purpose

To count all or a subset of the lines in a line file.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the logical unit or Fdub.

2- reference, value, Mts_Line_Number_Type

Specifies the line number of the first line to be counted.

3- reference, value, Mts_Line_Number_Type

Specifies the line number of the last line to be counted.

4- reference, Integer

Specifies a location in which to store the count.

See Mts_Io_Types for descriptions of the types of the first parameters.

Description

See MTS Volume 3

procedure Control

Purpose

To perform control operations on files and devices.

Parameters

1- reference, unknown

Specifies the control information. This will usually be a PLUS fixed-length string type. Note that a constant is not allowed, since Control may return information in this parameter (e.g., for the SNS control command).

2- reference, Short_Integer

A halfword integer giving the length of the control information.

3- reference, value, Mts_Io_Unit_Type

Specifies the unit or Fdub to be controlled.

4- optional, reference, Control_Return_Info_Type

A location at which the error information is returned. Control_Return_Info_Type is described below.

Description

See MTS Volume 3

Note

Control_Return_Info_Type describes the structure of the returned information. Its definition is included with library member Control. The definition is as follows:

```
record
   Dsr_Return_Code is Integer,
   Dsr_Message_Length is Integer,
   Dsr_Message is character(100)
end
```

procedure Cost

Purpose

To obtain the accumulated costs incurred by the current signon.

Result

Integer

Returns the cost in centicents.

Description

See MTS Volume 3

procedure Create

Purpose

To create a file.

Parameters

1- reference, value, unknown

Specifies the name of the file, with a trailing blank. This is usually a PLUS fixed-length string type.

2- reference, value, Create_Size_Type

A record specifying the maximum and initial sizes of the file. Create_Size_Type is described below.

3- reference, value, unknown

The volume. This is normally a location of type Integer, containing the value 0.

4- reference, value, Integer

A code specifying the type of file to create.

Library member Mts_File_Organizations contains constants which may be assigned to a variable to be used for this parameter. Normally, 256 should be added to those codes to indicate that the size is specified in pages.

Description

See MTS Volume 3

Note

The definition of Create_Size_Type is included in library member Create. It is the following record:

```
record
   Maximum_Size, Initial_Size are Short_Integer
end
```

Example

The following creates a line file of 10 pages, with name "-LOAD":

```
%Include(Create, Mts_File_Organizations);
   ...
constant Default_File_Name is "-LOAD ",
   Create_Size is Create_Size_Type(0,10); /* 10P, no maximum */
   Create_Kind is Line_File+256; /* Line, size in pages */
   ...
Create(Default_File_Name, Create_Size, 0, Create_Kind);
```

procedure Cuinfo

Purpose

To change various items of information about the user or task.

Parameters

1- reference, value, unknown

Specifies the item to change. May be a character(8) or an Integer.

2- reference, value, unknown

Specifies the new value. The type and format depends on the item being changed.

Description

See MTS Volume 3

Note

Constants for information returned from Cuinfo can be had by %Include-ing Guinfo_Cuinfo_Constants. Return code constants are in the global Guinfo_Cuinfo_Return_Codes.

procedure Destroy

Purpose

To destroy a file.

Parameters

1- reference, value, unknown

The name of the file to destroy, with trailing blank. This is usually a PLUS fixed-length string type.

Description

See MTS Volume 3

procedure Dismount

Purpose

To release magnetic and paper tapes, network devices, etc.

Parameters

1- reference, value, unknown

Specifies the pseudo-device(s) to dismount. This may be a PLUS fixed-length string type (if parameter (2) is given), or a character(0 to 256) (halfword length followed by the text).

2- optional, reference, value, Short_Integer

Gives the length of the first parameter.

Description

See MTS Volume 3

Example

To dismount the pseudo-devices *T1* and *T2*, either of the following forms might be used.

```
constant Pdns is "*T1* *T2*";
   ...
Dismount(Pdns, Short_Integer(Length(Pdns)));
   ...
Dismount(Long_Varying_String(Pdns));
```

Note the use of constant displays to force the constants passed into the correct form for the procedure.

procedure Edit

Purpose

To call the MTS system editor from a program.

Note

This is an extremely complex system subroutine. The procedure definition in *PLUS.SOURCELIB is designed to be as convenient as possible for the most common cases. However, some type-cheating will often be required in building parameters to be passed.

Parameters

1- reference, pointer to unknown

Points to editor dsect allocated by the editor. The pointer passed should be null on the first call.

| 2- reference, value, Cls_Transfer_Vector_Type

System subroutines transfer vector. For most purposes, however, a fullword -1 will be passed instead of a transfer vector, causing use of the standard system subroutines.

| 3- reference, value, Ed_Special_Io_Type

Transfer vector of routines to replace normal edit file interface routines. For most purposes, a fullword -1 will be passed instead of a transfer vector.

4- reference, value, unknown

The name of the file to edit. Normally a fixed-length character string.

5- reference, value, Integer

The length of the file-name passed as parameter 4.

6- reference, value, unknown

An initial edit command to be executed.

7- reference, value, Integer

The length of the edit command passed as parameter 6.

8- reference, value, Mts_Line_Number_Type

The minimum line number to be allowed, in internal form.

9- reference, value, Mts_Line_Number_Type

The maximum line number to be allowed, in internal form.

10-
reference, value, Integer

A line-number relocation factor to be subtracted from real line numbers.

11-
reference, value, Ed_Preprocess_Procedure_Type

A routine called by the editor to examine each command before it is processed by the editor. For most purposes, a fullword -1 will be passed instead of a procedure, meaning no routine is to be called.

12-
reference, value, Integer

This parameter is not used, but must be passed as a fullword -1.

13-
reference, value, bit(32)

A fullword of bit-switches specifying various edit subroutine options. Note that the setting of various bits in this word determine whether certain other parameters are processed or ignored. See description below.

14-
reference, character(20)

Variable in which the current file name may be stored on return.

15-
reference, Mts_Line_Number_Type

Variable in which the number of the editor current line may be returned.

16-
reference, bit(32)

Variable in which a set of return-status switches may be returned. See below.

Description

See MTS Volume 3 for a description of how the EDIT subroutine operates.

The library member also includes a number of type and

constant definitions that may be useful when using Edit. The other definitions are:

return-code definitions

Constants defining the return codes from the EDIT subroutine. The constants defined are:

Ed_Rc_Normal_Unloaded (0)
Ed_Rc_Normal_Loaded (4)
Ed_Rc_Error_Loaded (8)
Ed_Rc_Error_System (12)

procedure type definitions

A number of procedure types, with names of the form Rtn_Ed..._Type, for the special file-interface procedures that may be provided. Note that if these procedures are written in PLUS, the procedure declaration should specify "linkage system" or an alternate linkage procedure to reestablish the stack, since an S-type call is performed by the editor.

Ed_Special_Io_Type

Defines the transfer vector which may be used for parameter 3.

Ed_Preprocess_Procedure_Type

The type of the procedure that may be used for parameter 11. If this is written in PLUS, it is necessary to specify linakge system or an alternate linkage to reestablish the stack, since an S-type call is performed by the editor.

control switch constants

A number of bit-type constants which may be used to specify the value of parameter 13. A number of constants may be or'd together to set groups of options.

The constants defined are

Ed_Control_Sw_Use_Filename ('00000001')
Ed_Control_Sw_Initial_Ed_Cmd ('00000002')
Ed_Control_Sw_Cmds_Source ('00000004')
Ed_Control_Sw_Unload ('00000008')
Ed_Control_Sw_Inhibit_Edit ('00000010')
Ed_Control_Sw_Inhibit_Mts ('00000020')
Ed_Control_Sw_Inhibit_Copy ('00000040')
Ed_Control_Sw_Return_Any_Error ('00000080')
Ed_Control_Sw_Return_On_Null ('00000100')
Ed_Control_Sw_Return_On_Attn ('00000200')

```
      Ed_Control_Sw_No_Unload ('00000400')
      Ed_Control_Sw_Set_Current_Line ('00000800')
|     Ed_Control_Sw_Ignore_Initfile ('00001000')
```

return switches constants

Constants defining the codes which may be returned (possibly in combination), as the value of parameter 16. The constants defined are:

```
Ed_Proc_Eof_Enabled  (1) Ed_Proc_Success_Enabled (2)
Ed_Return_Stop_Or_Eof (4)
```

See *PLUS.SOURCELIB members Edit and Edit_Definitions for further details of these types and constants.

Example

The following illustrates a sequence of instructions that might be used to edit the file specified by the Varying_String Filename, and issue the initial command "VISUAL".

```
constant Visual_Cmd is "VISUAL",
   Switches is Ed_Control_Sw_Use_Filename
      | Ed_Control_Sw_Initial_Ed_Command
      | Ed_Control_Sw_Cmds_Source;

variable Ed_Cls_Vector is pointer to
      value Cls_Transfer_Vector_Type,
   Ed_Special_Io_Vector is pointer to
      value Ed_Special_Io_Type,
   Ed_Preprocess_Procedure is Ed_Preprocess_Procedure_Type,
   Ed_Dsect is pointer to unknown,
   Filenamelen is Integer,
   Rc is Integer,
   Return_File is character(20),
   Current_Line is Mts_Line_Number_Type,
   Return_Switches is bit(32);

/* Do the type-cheating needed to pass -1's to the
   editor for some of the parameters. */
equate Cheat1 to Ed_Cls_Vector as pointer to value Integer,
   Cheat2 to Ed_Special_Io_Vector as pointer to
       value Integer,
   Cheat3 to Ed_Preprocess_Procedure as Integer;
Cheat3 := -1;
/* Make others point to a fullword -1... */
Cheat1, Cheat2 := Address(Cheat3);

/* Initialize the dsect. */
Ed_Dsect := Null;
   ...
   ...
Filenamelen := Length(Filename);
Edit(Ed_Dsect, ED_Cls_Vector@, Ed_Special_Io_Vector@,
```

```
            Substring(Filename,0,0), Filenamelen,
            Visual_Cmd, Length(Visual_Cmd),
            Minimum_Integer, Maximum_Integer, 0,
            Ed_Preprocess_Procedure, -1,
            Switches, Return_File, Current_Line, Return_Switches,
            return code Rc);
```

procedure Empty

Purpose

To empty a file without destroying it.

Parameters

1- value, Mts_Fdub_Type

A Fdub for the file to be emptied.

Description

See MTS Volume 3

procedure Emptys

Purpose

To empty a file without destroying it.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the logical unit or Fdub.

Description

See MTS Volume 3

Note

This routine is called Emptyf at installations other than UBC.

procedure Error

Purpose

To suspend execution with an error indication.

Description

See MTS Volume 3

global Exit Definitions

Purpose

Defines a number of types that are useful for program, timer and attention interrupt exit routines.

Description

This member contains a number of type definitions that can be used in interfacing between a PLUS program and the MTS timer, program and attention interrupt routines (PGNTTRP, TIMNTRP and ATTNTRP).

A PLUS exit routine is normally called via an interface routine in PLUS:OBJLIB. The exit routine is called with a different stack from the "main program". The stack to use is specified at the time the exit is set up. The exit routine can access global variables as usual. Normally it will eventually restart the interrupted program. The macro Set_Exit can be used to set up a PLUS exit routine. See the description of Set_Exit for further details and an example.

The types defined by Exit_Definitions are as follows:

Stack_Type

defines a stack as a very large doubleword-aligned array. One should not attempt to allocate a variable of type Stack_Type, but should instead allocate the required amount of space, and use equate if necessary to treat it as Stack_Type.

Note that a different stack must be provided for each exit that is to be simultaneously enabled.

Exit_Routine_Type

defines a PLUS exit routine. The exit routine has one parameter (Exit_Area) which is of type pointer to Exit_Area_Type.

Exit_Area_Type

is a control block containing the registers and PSW at the time of the interrupt, as well as the information needed by the interface routine to call the PLUS exit handler.

Mts_Exit_Area_Type

is a region (within Exit_Area_Type) containing the information passed by Mts to an exit routine. Note this varies slightly between timer exits and other exits.

Exit_Save_Type

is a region (within Mts_Exit_Area_Type) containing the registers and PSW at the time of the interrupt.

procedure Fpsect

Purpose

To free psect (dsect) storage allocations.

Parameters

1- value, bit(32)

The psect-id.

Description

See MTS Volume 3

| procedure Fread
|
| Purpose
|
| Fread is an input routine for reading data in a free format.
|
| Parameters
|
| 1- reference, value, unknown
|
| This must be an Mts_Io_Unit_Type or user buffer.
|
| Twenty other reference, optional parameters are defined, all of type unknown.
|
| Description
|
| See MTS Volume 3

procedure Freefd

Purpose

To free a file or device acquired by the Getfd subroutine.

Parameters

1- value, Mts_Fdub_Type

The Fdub to be released.

Description

See MTS Volume 3

procedure Freespac

Purpose

To release storage acquired by the GETSPACE subroutine.

Parameters

1- value, numeric

The length of the region to be freed (or zero).

2- pointer to unknown

Any pointer specifying the locations to be freed.

Description

See MTS Volume 3

procedure Fsize

Purpose

To determine (via repeated calls) the file size required to contain a certain amount of information without actually writing the file.

Parameters

1- reference, value, Integer

Specifies the file organization. The possible values are described by global Mts_File_Organizations.

2- reference, value, Integer

The length of data.

3- reference, Fsize_Workarea_Type

See below for description of Fsize_Workarea_Type.

Description

See MTS Volume 3

Note

Fsize_Workarea_Type is included with member Fsize. It is a record containing the current file size and working space for Fsize. Its definition is as follows

```
record
   Current_Size is Integer,
   Last_Pointer is bit(32),
   Scratch is character(56)
end
```

procedure Fsrf

Purpose

To forward space records (lines) in a line file or sequential file.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the file or device.

2- reference, value, Integer

Specifies the the number of records to skip.

Description

See MTS Volume 3

Note

Constants for the return codes of this procedure are in the global Fsrf_Bsrf_Return_Codes, which is included along with it.

global Fsrf Bsrf Return Codes

Purpose

The constants defined herein may be compared with the return codes from the Bsrf and Fsrf procedures.

Description

The following constants are defined.

Fsrf_End_Of_File (4)
Fsrf_Illegal_Unit (8)
Fsrf_Read_Or_Write_Access_Not_Allowed (12)
Fsrf_Deadlock_Detected (16)
Fsrf_Wait_Cancelled (20)
Fsrf_File_Does_Not_Exist (24)

procedure Gdinf

Purpose

To obtain information returned from the subroutine Gdinfo in a Fortran environment.

Parameters

1- reference, value, Mts_Io_Unit_type

Specifies the file or device for which to obtain information.

2- reference, Gdinfo_Result_Type

Specifies the region in which the information is to be returned. See description of Gdinfo_Result_Type.

Note

The fields Gd_Fdname and Gd_Error_Msg should not be accessed following a call to Gdinf, since the locations pointed to will have been freespaced (a bug in Gdinf).

Description

See MTS Volume 3

procedure Gdinfo

Purpose

To obtain information about a file or device.

Parameters

1- Mts_Io_Unit_Type

Specifies the file or device for which information is requested.

Result

pointer to Gdinfo_Result_Type

(See member Gdinfo_Result_Type.) A region contain the information is returned. It must be explicitly freed by calling Freespac when no longer required.

Description

See MTS Volume 3

procedure Gdinfo2

Purpose

To get information about a file or device.

Parameters

See description of Gdinfo.

Description

See MTS Volume 3

procedure Gdinfo3

Purpose

To get information about a file or device.

Parameters

See description of Gdinfo.

Description

See MTS Volume 3

type Gdinfo Result Type

Purpose

To define the return information for Gdinfo and related procedures.

Description

This member declares a record type corresponding to the Assembler dsect in *GDINFODSECT.

It also declares a number of constants defining the possible values of the fields Gd_Use_Code and Gd_Device_Code.

See *PLUS.SOURCELIB for details.

macro Get Time And Date

Purpose

To set up and call the Time subroutine.

Parameters

1- value, integer

A numeric code for the item to be returned.

2- reference, unknown

The location at which to return the requested item. The type required depends on the item requested.

Description

This macro sets up a call to Time, defaulting the second parameter so that the item is to be returned only (not printed).

procedure Getfd

Purpose

To obtain a file or device.

Parameters

1- reference, value, unknown

Specifies the file or device, with trailing blank. This is normally a fixed-length string type.

Result

Mts_Fdub_Type

The Fdub for the file or device, if return-code is 0.

Description

See MTS Volume 3

procedure Getfst

Purpose

To return the line number associated with the first line in a file.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the logical unit or Fdub.

2- reference, Mts_Line_Number_Type

Specifies the location where the line number is to be returned.

Description

See MTS Volume 3

Note

Constants for the return codes for this procedure are in the global Getfst_Getlst_Return_Codes.

global Getfst Getlst Return Codes

Purpose

To provide constants for return codes from the procedures Getfst and Getlst.

Description

The following constants are defined.

Getfst_Empty_File (4)
Getfst_Hardware_Or_Software_Error (8)
Getfst_No_Access (12)
Getfst_Deadlock_Detected (16)
Getfst_Wait_Cancelled (20)
Getfst_File_Does_Not_Exist (24)

procedure Getime

Purpose

To return the time remaining until a specified timer interrupt will occur without cancelling the interrupt.

Parameters

1- reference, value, bit(32)

The code used to identify the timer when it was set.

2- reference, unknown

The time remaining until the interrupt will occur. The format depends on the value of the first parameter to Setime when the timer was set.

3- reference, unknown

The exit area specified when the timer was set.

Description

See MTS Volume 3

procedure Getlst

Purpose

To return the line number associated with the last line in a file.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the logical unit or Fdub.

2- reference, Mts_Line_Number_Type

Specifies the location where the line number is to be returned.

Description

See MTS Volume 3

Note

| Constants for the return codes for this procedure are in
| the global Getfst_Getlst_Return_Codes.

procedure Getspace

Purpose

To acquire storage.

Parameters

1- value, bit(32)

A word of switches affecting the allocation.

2- value, Integer

The number of bytes to be allocated.

3- optional, value, Integer

A storage index number to be used if the flag Storage_Index_Number_Given is specified in parameter 1.

Result

pointer to unknown

An address is returned, which may be assigned to an appropriate pointer variable.

Description

See MTS Volume 3

Note

Member Getspace also includes the following constant definitions which may be used (possibly in combination) as the value of the first parameter:

Dont_Return_If_Not_Available ('00000001')
Current_Link_Level ('00000002')
System_Storage ('00000004')
Storage_Index_Number_Given ('00000008')

Example

The following segment indicates how a record could be dynamically allocated in PLUS.

```
%Include(Getspace);
   ...
type List_Element is
      record
         ...
      end;
variable Next is pointer to List_Element;
   ...
```

```
Next := Getspace(Dont_Return_If_Not_Available
   | Current_Link_Level, Byte_Size(List_Element));
```

procedure Gfinfo

Purpose

To obtain information about a particular file or all of the files in a particular catalog.

Parameters

1- reference, value, unknown

Specifies what file or catalog. The type depends on the value of parameter 3. Normally it will be either an Integer of a fixed-length string type.

2- reference, Returned_File_Type

See below.

3- reference, value, bit(32)

A flag specifying what kind of parameter is passed as parameter 1. A number of constant definitions which may be assigned to a variable to be used for this parameter are included with the definition of Gfinfo.

4- reference, Catalog_Info_Type

Returns requested catalog information. See below.

5- reference, File_Info_Type

Returns requested file information. See below.

6- reference, Sharing_Info_Type

Returns sharing information. See below.

7- optional, reference, Integer

Returns error code.

8- optional, reference, character(80)

Returns an error message corresponding to the error code.

Description

This library member defines procedure Gfinfo, and several associated PLUS types and constants.

See MTS Volume 3 for a description of the operation of

Gfinfo; see *PLUS.SOURCELIB for details of the types defined.

The other definitions included with Gfinfo are:

Returned_File_Type

This is record consisting of a character string (Rtn_File_Name) in which a file name may be returned, and a scratch area required by Gfinfo. Note that this record must be set to zero before the first call to Gfinfo.

constants for parameter 3

The identifiers Gf_Release_Storage, Gf_File_Name, Gf_Fdub, Gf_Catalog_Name and Gf_No_Expensive_Info are defined for use as parameter 3 when calling Gfinfo.

Catalog_Info_Type

This is a record type defining the "catalog information dsect". Note that the first element, Ci_Array_Length must be set before calling Gfinfo.

constants for device type code

Constants Device_2311, Device_2314, Device_3221, Device_3330 are defined. These are the possible values of field Ci_Device_Type_Code of Catalog_Info_Type.

File_Info_Type

This defines the "file information dsect". Note that field Fi_Array_Length must be set before calling Gfinfo.

Sharing_Info_Type

Defines the "sharing information dsect". Note that field Si_Array_Length must be set before calling Gfinfo.

Sharing_List_Type

This defines the overall form of the sharing list pointed at by field Si_Sharing_List of Sharing_Info_Type. It is necessary to use type cheating to define and access the individual sharing list elements.

Sharing_List_Element_Type

Defines an element of the sharing list.

constants for accessor specification

The constants Project_Accessor, Userid_Accessor, Pkey_Accessor, Project_And_Pkey_Accessor and Userid_And_Pkey_Accessor are defined as the possible values of field Sl_Accessor in a sharing list element.

| constants for the possible return codes

procedure Gpsect

Purpose

To acquire psect (dsect) storage allocations.

Parameters

1- value, bit(32)

The psect-id to be used.

2- value, integer

The number of bytes to be allocated.

Result

pointer to unknown

The address of the allocated area.

Note that a return code of 0 indicates the psect was previously allocated; a return code of 4 indicates it was allocated by this call.

Description

See MTS Volume 3

procedure Grgjuldt

Purpose

To convert the Gregorian date to the corresponding Julian date.

Parameters

1- character(8)

The Gregorian date (mm/dd/yy).

Result

Integer

The corresponding Julian date (days since March 1, 1900).

Description

See MTS Volume 3

procedure Grgjultm

Purpose

To convert the Gregorian date and time to the corresponding Julian time.

Parameters

1- character(16)

The Gregorian date and time (mm/dd/yyhh:mm:ss).

Result

Integer

The corresponding Julian time in minutes from March 1, 1900.

Description

See MTS Volume 3

procedure Grjldt

Purpose

Provides an S-type call to Grgjuldt.

Parameters

1- reference, value, unknown

The Gregorian date. This will usually be character(8).

Result

Integer

The corresponding Julian day.

Description

See MTS Volume 3

procedure Grjlsec

Purpose

To convert the Gregorian date and time to Julian seconds.

Parameters

1- character(16)

The Gregorian date and time (mm/dd/yyhh:mm:ss).

Result

Integer

The corresponding Julian time in seconds from March 1 1900.

Description

See MTS Volume 3

procedure Grjltm

Purpose

Provides an S-type call to Grgjultm.

Parameters

1- reference, value, unknown

The Gregorian date and time. This will usually be character(16).

Result

Integer

The corresponding Julian time in minutes from March 1 1900.

Description

See MTS Volume 3

procedure Grosdt

Purpose

To convert the Gregorian date to the corresponding OS-format date.

Parameters

1- reference, value, character(8)

The Gregorian date to be converted (mm/dd/yy).

2- reference, character(8)

Location where the date is stored in the OS form yyddd, with three leading blanks.

Description

See MTS Volume 3

procedure Gtdjms

Purpose

S-type interface for Gtdjmsr.

Parameters

1- reference, value, character(16)

The Gregorian time and date.

2- reference, bit(64)

A doubleword integer in which the Julian microseconds is stored.

Description

See MTS Volume 3

procedure Gtdjmsr

Purpose

To convert the Gregorian time and date into Julian microseconds.

Parameters

1- character(16)

The Gregorian time and date in the format hh:mm:ssdd/yy/mm.

Result

bit(64)

A doubleword integer specifying the number of microseconds from March 1 1900.

Description

See MTS Volume 3

procedure Guinfo

Purpose

To allow the user to obtain information about her status and her task.

Parameters

1- reference, value, unknown

Key specifying what item is to be returned. This may be an integer or a character(8).

2- reference, unknown

The location at which the information is to be returned. The type required depends in the value of the first parameter.

Description

See MTS Volume 3

Note

| Constants for information returned from Guinfo can be had by %Include-ing Guinfo_Cuinfo_Constants. Return code constants are in the global Guinfo_Cuinfo_Return_Codes.

global Guinfo Cuinfo Constants

Purpose

To provide constants corresponding to values passed to/returned from the Cuinfo/Guinfo subroutines.

Description

The following constants are defined.

```
Guinfo_Switch_On (1)
Guinfo_Switch_Off (0)

Guinfo_Errordump_Off (0)
Guinfo_Errordump_On (1)
Guinfo_Errordump_Full (2)

Guinfo_Loader_Suppress_Prmap ('80')
Guinfo_Loader_Suppress_Pdmap ('40')
Guinfo_Loader_Print_Usmsg ('20')
Guinfo_Loader_Print_Uxref ('10')
Guinfo_Loader_Print_Xref ('08')
Guinfo_Loader_Print_Mapdots ('04')
Guinfo_Loader_Print_Map ('02')
Guinfo_Loader_Print_Warnings ('01')

Guinfo_Signoff_Long (0)
Guinfo_Signoff_Short (1)
Guinfo_Signoff_$ (2)

Guinfo_Endfile_Never (0)
Guinfo_Endfile_Off (1)
Guinfo_Endfile_On (2)

Guinfo_Task_Is_Terminal (0)
Guinfo_Task_Is_Local_Batch (1)
Guinfo_Task_Is_Remote_Batch (2)
Guinfo_Task_Is_Normal_Batch (3)
Guinfo_Task_Is_Asterisk_File (4)
Guinfo_Task_Is_Operator (5)

Guinfo_Spellcor_Off (0)
Guinfo_Spellcor_On (1)
Guinfo_Spellcor_Prompt (3)

Guinfo_Rcprint_Never (0)
Guinfo_Rcprint_Positive (1)
Guinfo_Rcprint_Non_Negative (2)
Guinfo_Rcprint_Always (3)

Guinfo_Overloaded_Processor ('80')
Guinfo_Overloaded_Paging ('40')
Guinfo_Overloaded_Disk_Io ('20')
Guinfo_Overloaded_Io_Activity ('10')
Guinfo_Overloaded_Drum_Space ('08')
```

Guinfo_Low_Priority (0)
Guinfo_Normal_Priority (1)
Guinfo_High_Priority (2)

global Guinfo Cuinfo Return Codes

Purpose

To provide constants for comparison with return codes from Guinfo and Cuinfo.

Description

The following constants are defined.

Guinfo_Invalid_Item_Number (4)
Guinfo_Item_Name_Not_In_List (8)
Guinfo_Illegal_To_Change_Item (12)
Guinfo_Illegal_Parameter_Address (16)

procedure Guinfupd

Purpose

To update certain items obtainable via the GUINFO subroutine.

Parameters

This routine has no parameters and no return value.

Description

See MTS Volume 3

procedure Guser

Purpose

To read an input record from the logical I/O unit GUSER.

Parameters

See description of Scards_Procedure_type.

Description

See MTS Volume 3

procedure Guserid

Purpose

To obtain the current 4-character CCID.

Result

character(4)

The current user's CCID.

Description

See MTS Volume 3

global Io Subroutine Return Codes

Purpose

To provide constants for comparison against return codes from input-output subroutines.

Description

See the library member for the constants defined.

procedure Jlgrdt

Purpose

S-type interface to Julgrgdt.

Parameters

1- reference, value, Integer

The Julian date.

2- reference, unknown

Location where Gregorian date is to be stored. This will usually be a character(8).

Description

See MTS Volume 3

procedure Jlgrsec

Purpose

To convert the number of seconds from March 1, 1900 to Gregorian date and time.

Parameters

1- Integer

The Julian time in seconds since March 1, 1900.

Result

character(16)

The corresponding Gregorian date and time in form dd/mm/yyhh:mm:ss.

Description

See MTS Volume 3

procedure Jlgrtm

Purpose

S-type interface to Julgrgtm.

Parameters

1- reference, value, Integer

The Julian time in minutes.

2- reference, unknown

Location where Gregorian date and time is to be stored. This will usually be a character(16).

Description

See MTS Volume 3

procedure Jmsgtd

Purpose

S-type interface for Jmsgtdr.

Parameters

1- reference, value,, bit(64)

A doubleword containing the Julian microseconds.

2- reference, unknown

Location where the corresponding Gregorian time and date will be returned. This will usually be a character(16) variable.

Description

See MTS Volume 3

procedure Jmsgtdr

Purpose

To convert the Julian time in microseconds to the corresponding Gregorian time and date.

Parameters

1- bit(64)

The Julian time in microseconds since March 1, 1900.

Result

character(16)

The Gregorian time and date in form hh:mm:ssmm/dd/yy.

Description

See MTS Volume 3

procedure Jtugtd

Purpose

S-type interface for Jtugtdr.

Parameters

1- reference, value,, bit(64)

A doubleword containing the Julian timer units.

2- reference, unknown

Location where the corresponding Gregorian time and date will be returned. This will usually be a character(16) variable.

Description

See MTS Volume 3

procedure Jtugtdr

Purpose

To convert the Julian time in timer units to the corresponding Gregorian time and date.

Parameters

1- bit(64)

The Julian time in timer units from March 1, 1900.

Result

character(16)

The Gregorian time and date (hh:mm:ssmm/dd/yy).

Description

See MTS Volume 3

procedure Julgrgdt

Purpose

To convert the Julian date to the corresponding Gregorian date.

Parameters

1- value, Integer

The Julian date (days from March 1, 1900).

Result

character(8)

The corresponding Gregorian date (mm/dd/yy).

Description

See MTS Volume 3

procedure Julgrgtm

Purpose

To convert the Julian time in minutes to the corresponding Gregorian date and time.

Parameters

1- value, Integer

The Julian time (minutes from Match 1, 1900).

Result

character(16)

The corresponding Gregorian date and time (mm/dd/yyhh:mm:ss).

Description

See MTS Volume 3

procedure Kwscan

Purpose

To scan a list of keywords and perform specified action(s) for each keyword assignment.

Parameters

1- reference, value, Short_Integer Length of left-hand table.

2- reference, unknown

The left-hand table.

3- reference, unknown

The execute table.

4- reference, value, unknown

The text to be parsed.

5- reference, unknown

The right-hand table.

6- reference, value, Short_Integer

The length of the text.

7- reference, value, bit (32)

A fullword of switches.

8- reference, unknown

The return vector.

9- reference, optional, unknown

The delimiter list.

10-
reference, optional, unknown

The separator list.

Description

See MTS Volume 3

Note

This routine is not very useful for PLUS programs.

global Lcs_Types

Purpose

To define types for low-core symbol tables and LCS PRV's.

Description

This member contains the following types and macros. Some of these also appear under Loader_Definitions, as that member %Includes Lcs_Types.

type Esd_List_Entry_Type

Defines an element of a loader external symbol list.

type Low_Core_Symbol_Type

Defines an element of a low core symbol table.

|
|
| type Lcspr_Entry_Type
|
| Defines an element of a low-core pseudo-register vector.
|
| type Lcspr_Type
|
| Defines a low-core pseudo-register vector.
|
| macro Lcs_Entry
|
| Used to generate a constant entry in an LCS.
|
| macro Null_Lcs_Entry
|
| Used to generate a null entry in an LCS.
|
| macro Lcspr_Entry
|
| Used to generate a constant entry in an LCSPR.

procedure Letgo

Purpose

To periodically unlock and then relock a file.

Parameters

1- reference, value, Mts_Fdub_Type

A Fdub for the file to be unlocked.

2- reference, value, Integer

A code specifying how the file is to be relocked. Constants defining possible values for this variable are included with the definition of the Lock subroutine.

3- reference, value, Integer

Specifies the interval at which the file is to be unlocked.

Description

See MTS Volume 3

procedure Link

Purpose

To effect the dynamic loading and execution of a program.

Parameters

1- reference, value, unknown

The input specifier. The type depends on parameter 2; it will normally be an Mts_Io_Unit_Type, a fixed-length string type (containing Fdname with trailing blank), or a system procedure.

2- reference, value, unknown

This may be either an Integer containing 0, or a Loader_Info_Parameter_Type (see Loader_Definitions).

3- reference, value, unknown

Specifies the parameter list to be passed to the loaded program. Note that when the parameter list contains only one entry, this parameter to Link will be a pointer. When there is more than one parameter, Link will be passed an array of pointers. If a variable-length parameter list is expected, the caller of Link must ensure the parameter list has the appropriate high-order bit set.

4- optional, reference, unknown

Normally a system procedure, called if an error occurs.

5- optional, reference, unknown

Normally a system procedure, specifies a routine to be called for output from the loader.

6- optional, reference, bit(32)

Specifies a "loader status word" of flags. Constants which may be assigned to a variable for use as this parameter are defined in Loader_Definitions.

7- optional, reference, unknown

Normally a system procedure; specifies a subroutine to be used in place of Getspace.

8- optional, reference, unknown

Normally a system procedure; specifies a subroutine to be used in place of Freespace.

9- optional, reference, unknown

Normally a system procedure; specifies a subroutine to be used in place of Point.

Description

See MTS Volume 3

procedure Load

Purpose

To effect the dynamic loading of a program.

Parameters

1- reference, value, unknown

The input specifier. The type depends on parameter 2; it will normally be an Mts_Io_Unit_Type, a fixed-length string type (containing Fdname with trailing blank), or a system procedure.

2- reference, value, unknown

This may be either an Integer containing 0, or a Loader_Info_Parameter_Type (see Loader_Definitions).

3- reference, value, bit(32)

Specifies a location containing Load control switches.

4- pointer to unknown

Specifies an area in which an ESD list is to be returned.

5- optional, reference, unknown

Normally a system procedure, called if an error occurs.

6- optional, reference, unknown

Normally a system procedure, specifies a routine to be called for output from the loader.

7- optional, reference, bit(32)

Specifies a "loader status word" of flags. Constants which may be assigned to a variable for use as this parameter are defined in Loader_Definitions.

8- optional, reference, unknown

Normally a system procedure; specifies a subroutine to be used in place of Getspace.

9- optional, reference, unknown

Normally a system procedure; specifies a subroutine to be used in place of Freespace.

10-
optional, reference, unknown

Normally a system procedure; specifies a subroutine to be used in place of Point.

Result

Integer

The storage index number used.

Note that if loading was successful, the "return code" specifies the entry point.

Description

See MTS Volume 3

global Loader Definitions

Purpose

Defines several constants and record types which are useful in calling the Link, Load and Xctl subroutines.

Description

This member contains the following:

type Esd_List_Entry_Type

Defines an element of a loader external symbol list.

type Loader_Info_Parameter_Type

Defines the "info" parameter used by the Link, Load and Xctl procedures to specify loader switches and initial ESD list.

constants for loader switches

A number of constants are defines that may be used in creating the value of a loader "info" parameter.

constants for loader status word

A number of constants are defined that may be assigned to a variable as the value of the loader status word parameter.

See the contents of *PLUS.SOURCELIB, and MTS Volume 3 for details.

Note

The PLUS declarations of the dynamic loading facilities of MTS do not currently attempt to define all the possible variations. It is expected that most uses of these definitions will require some type-cheating.

procedure Loadinfo

Purpose

To return information about an external symbol or a virtual memory address.

Parameters

1- reference, value, Integer

A code indicating what kind of parameter has been passed as parameter 2.

2- reference, value, unknown

The item for which information is to be returned. Member Loadinfo also defines constants that may be used for this parameter.

3- reference, Loadinfo_Return_Bits_Type

A record containing a series of switches which are set to indicate which items have been returned.

4- reference, Loadinfo_Return_Vector_Type

A record in which information is returned.

Description

See MTS Volume 3

Note

This member also defines the record types Loadinfo_Return_Bits_Type and Loadinfo_Return_Vector_Type. See *PLUS.SOURCELIB for details.

procedure Lock

Purpose

To request that a file be locked in the indicated manner.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the logical unit or Fdub.

2- reference, value, Integer

A flag specifying how the file is to be locked. See below.

3- reference, value, Integer

Specifies how long to wait if the file cannot be locked currently.

Description

See MTS Volume 3

Note

Member Lock also defines constants which can be used for the second parameter. The constants are:

```
Lock_Read (1)
Lock_Modify (0)
Lock_Destroy (-1)
```

procedure Lodmap

Purpose

To produce a loader map from the current contents of the loader tables.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies where the loader map is to be written.

2- reference, value, bit(32)

"Loader status word" switches controlling what is printed. Constants that may be assigned to a variable for use as this parameter are defined in member Loader_Definitions.

Description

See MTS Volume 3

macro Model_Number

Purpose

To convert the compilation date to the MTS "model number" format.

Result

Varying_String

The converted date.

Description

The compilation date of the program being run is converted to MTS "model number" format, for example "January 05, 1983" becomes "AN053".

procedure Mount

Purpose

To mount magnetic and paper tapes, network connections, etc.

Parameters

1- reference, value, unknown

Location containing the mount command(s). Should be either a fixed-length string type or character(0 to 256).

2- optional, reference, value, Short_Integer

Halfword length of first parameter is it does not include length.

Description

See MTS Volume 3

procedure Mts

Purpose

To suspend execution of a program and return to MTS command mode.

Description

See MTS Volume 3

procedure Mtscmd

Purpose

To suspend execution of a program, return to MTS command mode, and feed a character string to the MTS command

Parameters

1- reference, value, unknown

The command to be interpreted. This will usually be a PLUS fixed-length string type.

2- reference, value, Integer

The length of the command.

Description

See MTS Volume 3

global Mts_File_Access_Codes

Purpose

To define the codes used for file-access.

Description

This global just contains a number of PLUS constant declarations giving the codes used by various system subroutines (e.g., PERMIT) for specifying access.

The constants defined are:

```
Read_Access ('01')
Write_Expand_Access ('02')
Write_Change_Access ('04')
Empty_Access ('04')
Truncate_Access ('08')
Renumber_Access ('08')
Destroy_Access ('10')
Rename_Access ('10')
Permit_Access ('20')
```

```
Default_Access ('80') --  used by Permit only
Write_Access (Write_Expand_Access |
                  Write_Change_Access)
Read_Write_Access (Read_Access | Write_Access)
Unlim_Access ('3f')
```

global Mts File Organizations

Purpose

To define the codes used to define MTS file types.

Description

This global just contains PLUS constant declarations giving the codes used by Gfinfo, Create, etc. for file organization.

The constants defined are:

```
Line_File (0)
Sequential_File (1)
Seqwl_File (2)
```

global Mts Io Modifiers

Purpose

Defines the values of I/O modifiers

Description

This global just contains a number of constant declarations defining bitstring constants for the modifier values.

The names of the constants are of the form Mts_Io_x or Mts_Io_Not_x, where x is the name of the modifier.

See *PLUS.SOURCELIB for details.

global Mts Io Types

Purpose

Defines some types that are useful in interfacing to I/O subroutines under MTS.

Description

This global defines several types that are used in the definitions of MTS I/O subroutines.

The types defined are

Mts_Fdub_Type

This is just defined as bit(32).

Mts_Io_Modifiers_Type

Also defined as bit(32). Note that member Mts_Io_Modifiers defines the various modifiers by means of constant declarations.

Mts_Line_Number_Type

This is also defined as bit(32), rather than using a numeric range.

Mts_Io_Length_type

Defines the form of the length parameter used with the @MAXLEN modifier. That is, it is a record consisting of three halfword-integer fields, called Transmitted_Length, Maximum_Length and Actual_Length.

Mts_Io_Unit_Type

Defines a record corresponding to the I/O unit-or-fdname parameter allowed by many I/O subroutines. It is a PLUS variant record which may contain a Fdub (field Fdub), a logical unit name (field Liounit - character(8)) or a logical unit number (field Lio_Number, type Integer).

macro Mvcl Instruction

Purpose

To move difficult-to-type data quickly from one location to another.

Parameters

1- pointer to unknown

Pointer to the target storage location.

|
|
| 2- numeric
|
| Length of the target storage area.
|
| 3- pointer to unknown
|
| Pointer to the source storage location.
|
| 4- numeric
|
| Length of the source storage area.
|
| 5- bit(8)
|
| Character to be used to pad the data.
|
| Description
|
| This macro uses Inline to issue the appropriate MVCL
| instruction to move the data.
|
| Note
|
| This macro should not be used as a replacement for normal
| assignation -- no type or run-time value checking is done.
|
| In general, it is possible via PLUS type cheating to
| persuade the compiler to perform assignments and should be
| unnecessary to resort to this macro.

procedure Note

Purpose

To "remember" the values of the logical pointers for a sequential file. This information is used by the POINT subroutine to change the values of the logical pointers.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the logical unit or Fdub.

2- reference, Note_Point_Info_Type

Location in which the pointers are to be returned. See member Note_Point_Info_Type.

Description

See MTS Volume 3

type Note_Point_Info_Type

Purpose

To define the note-point information used by the subroutines Note and Point.

Description

This member defines a record type containing fields Read_Pointer, Write_Pointer and Last_Pointer (all bit(32)) and Last_Line_Number (Mts_Line_Number_Type), corresponding to the structure returned by Note and passed to Point.

procedure Osgrdt

Purpose

To convert the OS-format date to the corresponding Gregorian date.

Parameters

1- reference, value, character(8)

The date in OS format (yyddd), padded on the left to eight characters.

2- reference, character(8)

Location to return the corresponding Gregorian date (mm/dd/yy).

Description

See MTS Volume 3

procedure Permit

Purpose

To permit a file so that it can be shared by other users.

Parameters

1- reference, value, unknown

Specifies what is to be permitted. Normally, this is either of type Mts_Io_Unit_Type, or a fixed-length string type containing a file-name with trailing blank.

2- reference, value, bit(32)

A location specifying the access to be allowed. Member Mts_File_Access_Codes defines some constants that may be used for this parameter.

3- reference, value, Integer

A location specifying what type of accessor is represented by parameter 5 (or parameters 5 and 8). Member Permit also defines some constants that may be used as for this parameter.

4- reference, value, Integer

Specifies the length of the next parameter.

5- reference, value, unknown

Normally a fixed-length string type. Specifies the "accessor".

6- reference, value, Integer

Determines whether the first parameter is treated as an Mts_Io_Unit_Type (value 1) or a file name (value 0).

7- optional, reference, Integer

If parameter 3 indicates that a CCID or department and a program key are specified, then this is the length of the program key.

Otherwise, this is used to return an error code.

Note that since it may return a value under some conditions, a constant cannot be used.

8- optional, reference, unknown

If parameter 3 indicates that a CCID or department and a program key are specified, then this is the program key. It may be any fixed-length string type.

Otherwise, this is used to return an error message, and should be of type character(80).

Note that since it may return a value under some conditions, a constant cannot be used.

9- optional, reference, Integer

If parameter 3 indicates that a CCID or department and a program key are specified, then this parameter may be used to return an error code instead of parameter 7.

10-
optional, reference, character(80)

If parameter 3 indicates that a CCID or department and a program key are specified, then this parameter may be used to return an error message instead of parameter 8.

Description

See MTS Volume 3

Note

This member also defines the following constants, which may be assigned to a variable to be passed as parameter 3:

```
Who_Is_Id (0)
Who_Is_Project (1)
Who_Is_Others (2)
Who_Is_All (3)
Who_Is_Me (4)
Who_Is_Owner (5)
Who_Is_Program_Key (6)
Who_Is_Id_And_Key (7)
Who_Is_Project_And_Key (8)
```

| Constants corresponding to the possible values returned in
| the Ercode parameter are also defined.

procedure Pgnttrp

Purpose

To allow control to be returned to the user on a program interrupt.

Parameters

1- system procedure

A procedure to be called when a program interrupt occurs, or Null.

2- reference, unknown

An area to save the registers and PSW when the interrupt occurs. This will normally be a variable of type Mts_Exit_Area_Type (see Exit_Definitions).

Description

See MTS Volume 3

Note

The macro Set_Exit may be used to set up a PLUS procedure as an program interrupt exit routine.

| procedure Pkey

| Purpose

| To push and pop program keys.

| Parameters

| 1- reference, value, unknown

| Action to be taken by the Pkey procedure, terminated by a blank. The action must be one of "PUSH ", "POP ", "SET " or "RESET ".

| 2- optional, reference, value, unknown

| New pkey, terminated by a blank.

| Description

| See MTS Volume 3

procedure Point

Purpose

To alter the values of any or all of the logical pointers for a sequential file.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the logical unit or Fdub.

2- reference, Note_Point_Info_Type

A record containing the new value or values. See member Note_Point_Info_Type.

3- reference, value, bit(32)

Code specifying which of the pointers are to be reset. See below.

Description

See MTS Volume 3

Note

Member Point also defines the following constants:

```
Set_Read_Pointer ('00000001')
Set_Write_Pointer ('00000002')
Set_Last_Pointer ('00000004')
Set_Last_Line_Number ('00000008')
```

which may be assigned used (possibly in combination) for the third parameter.

procedure Qpsect

Purpose

To retrieve psect (dsect) storage allocations.

Parameters

1- value, bit(32)

The psect-id to be used.

Result

pointer to unknown

The address of the allocated area, or Null if it has not been allocated.

Note that a return code of 0 indicates the psect was found, a return code of 4 indicates it was not.

Description

See MTS Volume 3

procedure Quit

Purpose

To cause the user to be signed off when the next MTS command is encountered.

Description

See MTS Volume 3

procedure Read

Purpose

To read an input record from a specified logical I/O unit.

Parameters

1- reference, unknown

Specifies the starting location of the buffer.

2- reference, unknown

This will normally be either a Short_Integer, or a record of type Mts_Io_Length_Type. It returns the number of bytes transmitted.

3- reference, value, Mts_Io_Modifiers_Type

The modifiers to be used.

4- reference, Mts_Line_Number_Type

The line number to use (if indexed), or returned line number of line read.

5- reference, value, Mts_Io_Unit_Type

Specifies the unit or Fdub to use.

Result

optional, numeric

The return value if notification or noprompt is requested.

Description

See MTS Volume 3

procedure Rename

Purpose

To change the name of a file.

Parameters

1- reference, value, unknown

A fixed-length string type, specifying the old file name (with trailing blank).

2- reference, value, unknown

A fixed-length string type, specifying the new file name (with trailing blank).

Description

See MTS Volume 3

procedure Renumb

Purpose

To renumber all or a subset of the lines in a line file.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the logical unit or Fdub.

2- reference, value, Mts_Line_Number_Type

The starting line number of the portion to be renumbered.

3- reference, value, Mts_Line_Number_Type

The ending line number of the portion to be renumbered.

4- reference, value, Mts_Line_Number_Type

The new beginning line number for the portion to be renumbered.

5- reference, value, Mts_Line_Number_Type

The increment to use for renumbering.

Description

See MTS Volume 3

procedure Retlnr

Purpose

To return all or a subset of the line numbers in a line file.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the logical unit or Fdub.

2- reference, value, Mts_Line_Number_Type

Specifies the first line number to be returned.

3- reference, value, Mts_Line_Number_Type

Specifies the last line number to be returned.

4- reference, Integer

A location is which the count of the number of lines in the specified range will be returned.

5- reference, Retlnr_Buffer_Type

A buffer or list of buffers in which the line numbers can be returned. See below.

Description

See MTS Volume 3

Note

Member Retlnr also includes the definition of type Retlnr_Buffer_Type, which is a record defining the format of the line-number buffers used by Retlnr and Setlnr.

The definition is as follows:

```
record
   Next_Buffer is pointer to Retlnr_Buffer_Type,
   Buffer_Length is Integer,
   Line_Numbers is array (1 to 32767) of
                Mts_Line_Number_Type
end;
```

Note that generally type cheating will be used to pass a storage area containing a smaller array than this. Also, the fields Next_Buffer and Buffer_Length must be preset before calling Retlnr.

procedure Rewind

Purpose

To rewind a logical I/O unit in FORTRAN.

Parameters

1- reference, value, Integer

Specifies the logical unit number (0 through 19) to be rewound.

Description

See MTS Volume 3

procedure Rewind#

Purpose

To reset a magnetic tape or a file to be read from the beginning.

Parameters

1- Mts_Io_Unit_Type

Specifies the file or device to be rewound.

Description

See MTS Volume 3

procedure Rstime

Purpose

To cancel timer interrupts set up by the Setime subroutine and return the time remaining until the interrupt would have occurred.

Parameters

1- reference, value, bit(32)

The code used to identify the timer in the call to Setime.

2- reference, unknown

Used to return the time remaining until the timer would have occurred. The format depends on the value of parameter 1.

3- reference, unknown

The exit area specified for this timer.

Description

See MTS Volume 3

| procedure Rssas

| Purpose

| To reset *SOURCE* to *MSOURCE* and *SINK* to *MSINK*.

| Parameters

| 1- optional, reference, value, Integer

| A set of switches to control what is reset. Constants Reset_Source_And_Sink, Reset_Source and Reset_Sink are defined in the library member.

| Description

| See MTS Volume 3

| procedure Rtwait

| Purpose

| To wait for a given amount of real time at a terminal task.

| Parameters

| 1- reference, value, Integer

| Amount of real time to wait, in (seconds x 300), between 0 and 60 seconds (0 and 18000 units).

| Description

| This procedure causes a real-time wait for the indicated length of time. The wait is only performed for terminal tasks. If invoked in batch mode, this procedure will return without effect.

procedure Scanstor

Purpose

To "scan" storage blocks. For each block of allocated storage in the range specified, SCANSTOR will call a subroutine specified, giving it the location and length of that block.

Parameters

1- value, (-1 to 1)

A switch indicating the range of storage index numbers to be scanned.

2- value, Integer

The storage index number or limit of storage index number range.

3- system procedure

A system procedure to be called for each storage block. Note this cannot currently be written in PLUS.

Description

See MTS Volume 3

procedure Scards

Purpose

To read an input record from the logical I/O unit SCARDS.

Parameters

See description of Scards_Procedure_Type.

Description

See MTS Volume 3

type Scards_Procedure_Type

Purpose

To define the type of Scards and similar procedures.

Parameters

1- reference, unknown

Specifies the starting location of the buffer.

2- reference, unknown

This will normally be either a Short_Integer, or a record of type Mts_Io_Length_Type. It returns the number of bytes transmitted.

3- reference, value, Mts_Io_Modifiers_Type

The modifiers to be used.

4- reference, Mts_Line_Number_Type

The line number of the line read, or line to be read (if indexed).

Result

optional, numeric

The return value if notification or noprompt is requested.

Description

This type declaration is used to define Scards and Guser.

procedure Sdump

Purpose

To produce a dump of general registers, floating point registers, and/or a region of virtual memory.

Parameters

1- reference, value, Sdump_Bits_Type

A word of switches specifying the format and contents of the dump. See below.

2- Sprint_Procedure_Type

A procedure to be called for each line of the dump. If the procedure is written in PLUS, it requires special linkage to retrieve the PLUS environment. Linkage "QSACHAIN" can be used to set the environment up efficiently.

3- reference, Sdump_Workarea_Type

A location to be used as scratch space by Sdump.

4- bit(24)

The address of the first location to be dumped. Normally type-cheating will be needed to specify this parameter.

5- bit(24)

The address of the last location to be dumped. Normally type-cheating will be needed to specify this parameter.

Description

See MTS Volume 3

Note

| Member Storage_Dump defines a macro that may be used to
| interface to this routine.

Member Sdump also defines the following types:

Sdump_Workarea_Type

specifies the working storage required by SDUMP.

Sdump_Bits_Type

is a record containing a number of Boolean switches that are set to determine the output from SDUMP.

type Sense_Data_Type

Description

This library member contains a record-type defining the format of the data returned by the CONTROL subroutine for the SNS control operation.

It is the PLUS equivalent of *SNSDSECT.

See *PLUS.SOURCELIB for details.

procedure Sercom

Purpose

To write an output record on the logical I/O unit SERCOM.

Parameters

See description of Sprint_Procedure_Type.

Description

See MTS Volume 3

macro Set_Exit

Purpose

To set up a PLUS routine as a program, attention or timer exit routine.

Parameters

1- system procedure

One of the procedures ATTNTRP, PGNTTRP or TIMNTRP.

2- Exit_Routine_Type

A PLUS routine to call when an exit occurs.

3- name, Exit_Area_Type

An area in which the exit routine name, etc. are saved when the exit is set up. Contains the registers and PSW when the exit is taken.

Note this space must not be changed or released until the exit is cancelled.

4- name, Exit_Stack_Type

A region to use as a stack when calling the exit routine.

5- Boolean

A flag indicating whether this setup call is to return (False), or restart (True) from a previous exit whose status was saved in the Exit_Area.

Description

This macro sets a timer, program or attention interrupt exit. The actual exit routine is set to be an assembler interface routine from PLUS:OBJLIB. This interface routine then calls the specified PLUS routine, with the specified stack, when the exit occurs. Generally, the exit routine will eventually restart the interrupted routine. It may also use the Return_From macro to abort part of the interrupted environment.

Note

See also Exit_Definitions.

Example

The following indicates how an attention handling routine might be written in PLUS.

```
%Include(Exit_Definitions, Set_Exit, Attntrp);
   ...
variable Attn_Stack is pointer to Stack_Type,
   Attn_Area is Exit_Area_Type;
procedure Attn_Routine is Exit_Routine_Type;
   ...
definition Setup
   Attn_Stack := Getspace(3,2048); /* half page
                                      stack */
      ...
   /* Set up the exit... */
   Set_Exit(Attntrp, Attn_Routine, Attn_Area,
      Attn_Stack@,  False);
      ...
end Setup;
   ...
definition Attn_Routine
   /* do something about Attn... */
   ...
   /* Reset the exit and restart */
   Set_Exit(Attntrp, Attn_Routine, Exit_Area@,
      Attn_Stack@, True);
end Attn_Routine;
```

procedure Setime

Purpose

To set up a timer interrupt to occur after a specified time interval.

Parameters

1- reference, value, Integer

A code indicating what kind of timer value has been given. See below.

2- reference, value, bit(32)

A code used to identify this timer in subsequent calls.

3- reference, value, unknown

The time when the interrupt is to occur. The format depends on the value of parameter 1.

4- reference, unknown

An exit area to be used for this timer. Normally this is of type Exit_Area_Type.

Description

See MTS Volume 3

Note

The member Setime also defines the following constants which may used for the first parameter.

```
Task_Microseconds_From_Call (0)
Real_Microseconds_From_Call (1)
Task_Microseconds_From_Signon (2)
Real_Microseconds_From_Signon (3)
Task_Timer_Units_From_Call (4)
Absolute_Time_And_Date (5)
```

procedure Setioerr

Purpose

To allow users to regain control when I/O transmission errors that would otherwise be fatal occur during execution.

Parameters

1- Io_Error_Routine_Type

Specifies the routine to be called for subsequent I/O errors.

Description

See MTS Volume 3

Note

The definition of Setioerr includes the definition of Io_Error_Routine_Type, as a system procedure.

There is currently no provision for setting up a PLUS routine as an I/O exit routine.

procedure Setkey

Purpose

To set the program key associated with a file.

Parameters

1- reference, value, unknown

This will normally be either a variable of type Mts_Io_Unit_Type, or a fixed length string-type containing a file name with trailing blank.

2- reference, value, unknown

The new program key, with a trailing blank. This will normally be a fixed length string-type.

3- reference, value, Integer

Specifies what kind of parameter is passed for parameter 1. (0 indicates a file name, 1 indicates Mts_Io_Unit_Type.)

4- optional, reference, Integer

Returns an error code if the return-code is non-zero.

5- optional, reference, character(80)

Returns an error message if return-code is non-zero.

Description

See MTS Volume 3

Note

| Constants corresponding to the possible values returned in
| the Ercode parameter are also provided.

procedure Setlcl

Purpose

To set a local time limit for the executing program.

Parameters

1- reference, value, Integer

The time limit, in timer units.

Result

optional, Integer

The timing remaining in the previously set time limit.

Description

See MTS Volume 3

procedure Setlio

Purpose

To assign a file or device to a logical I/O unit.

Parameters

1- reference, value, character(8)

Specifies the logical unit to be assigned.

2- reference, value, unknown

Specifies the Fdname (with trailing blank) to be assigned to it. This will usually be a PLUS fixed-length string type.

Description

See MTS Volume 3

procedure Setlnr

Purpose

To set all or a subset of the line numbers in a line file.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the logical unit or Fdub.

2- reference, value, Mts_Line_Number_Type

Specifies the first line number of the region to be reset.

3- reference, value, Mts_Line_Number_Type

Specifies the last line number to be reset.

4- reference, value, Integer

Specifies the number of lines to be reset.

5- reference, Retlnr_Buffer_Type

A buffer or list of buffers containing the linenumbers to be set. See description of Retlnr for details.

Description

See MTS Volume 3

procedure Setpfx

Purpose

To set a single character input/output prefix character for the program currently executing.

Parameters

1- reference, value, character(1)

The new prefix character.

Result

character(1)

The previous prefix character.

Note that return code 4 indicates that the previous prefix was more than one character.

Description

See MTS Volume 3

procedure Sioc

Purpose

To perform floating-point, integer, logical and hexadecimal input/output conversions.

Note

This is a complex routine. Routines in PLUS:OBJLIB (described in section F, previously) will provide many similar conversion services, and are much easier to invoke.

Parameters

1- reference, unknown

The buffer containing characters to be converted, or result of conversion.

2- reference, Sioc_Data_Area_Type

A control block containing parameters indicating the type of conversion and containing internal forms of data and a work-area for use by SIOC.

Description

See MTS Volume 3 for description of the use of Sioc. Note that the caller must provide and initialize many of the fields of the control block used for the second parameter.

Library member Sioc also includes the following types:

Sioc_Control_Type

A field within the Sioc data area consisting of a number of control flags.

Sioc_Picture_Type

A field occurring twice within the Sioc data area to describe the fields to be converted, or the fields found.

Sioc_Data_Area_Type

A record describing the format of the Sioc work area passed as parameter two.

See *PLUS.SOURCELIB for details of any of the above types.

procedure Skip

Purpose

To space a magnetic tape or file either forward or backward a specified number of records or files.

Parameters

1- reference, value, Integer

The number of files to skip.

2- reference, value, Integer

The number of records to skip.

3- reference, value, Mts_Io_Unit_Type

The unit to be repositioned.

Description

See MTS Volume 3

procedure Spellchk

Purpose

To determine if a string is a possible misspelling of another string.

Parameters

1- reference, value, unknown

Any fixed-length string type; this is a word that is known to be correctly spelled.

2- reference, value, unknown

Any fixed-length string type; the word that is to be compared with parameter 1.

3- reference, value, Integer

The length of the string specified for the first parameter (must be a number between 1 and 32).

4- reference, value, Integer

The length of the string specified for the second parameter (must be a number between 1 and 32).

Result

(-1 to 1)

Code indicating the strings are identical, possible misspelling or otherwise.

The constants Spellchk_Words_Identical, Spellchk_Possibly_Misspelled and Spellchk_Otherwise are defined.

Description

See MTS Volume 3

procedure Sprint

Purpose

To write an output record on the logical I/O unit SPRINT.

Parameters

See description of Sprint_Procedure_Type.

Description

See MTS Volume 3

type Sprint Procedure Type

Purpose

To define the type of Sprint and similar procedures.

Parameters

1- reference, value, unknown

Specifies the starting location of the buffer.

2- reference, value, Short_Integer

Specifies the number of bytes to be transmitted.

3- reference, value, Mts_Io_Modifiers_Type

The modifiers to be used.

4- optional, reference, Mts_Line_Number_Type

The line number to use (if indexed), or line written (if Getlinenumber modifier).

Result

optional, numeric

The return value if notification or noprompt is requested.

Description

This type declaration is used to define Sprint, Spunch and Sercom, and those routines which require a "Sprint-like" procedure as a parameter.

procedure Spunch

Purpose

To write an output record on the logical I/O unit SPUNCH.

Parameters

See description of Sprint_Procedure_Type.

Description

See MTS Volume 3

macro Standard Dump

Purpose

| To interface to the STDDMP routine.

Parameters

1- reference, value, bit(32)

Contains the storage index number and switches.

2- Sprint_Procedure_Type

Specifies a procedure to be called for each line of the
| output. It must be declared as Sprint_Procedure_Type with
| linkage "QSACHAIN".

3- name, unknown

Location containing the address of the first location to be dumped. This may be a variable of a pointer type.

4- name, unknown

Location containing the address of the last location to be dumped. This may be a variable of a pointer type.

Description

| This macro just type cheats to allow use with arbitrary
| parameters, and provides the workarea required by STDDMP.

procedure Startf

Purpose

To execute a program dynamically loaded by the subroutine LOADF.

Parameters

1- reference,, unknown

Specifies the location to start. This is normally either of type Integer (specifying the storage index number returned by Loadf) or character(8) (specifying an entry-point name).

2...7- optional, reference, unknown

Up to six optional parameters to be passed to the loaded program as a parameter list.

Description

See MTS Volume 3

procedure Stddmp

Purpose

To dump a region of the user's virtual memory in the MTS standard format.

Parameters

1- reference, value, bit(32)

A fullword specifying a storage index number and switches. Constants Stddmp_Nolib and Stddmp_Doublespace are defined for use in these switches.

2- Sprint_Procedure_Type

A procedure to be called for each line of the dump. If the procedure is written in PLUS, it requires special linkage to retrieve the PLUS environment. Linkage "QSACHAIN" can be used to set the environment up efficiently.

3- reference, character(400)

A location to be used as scratch space by Sdump.

4- bit(24)

The address of the first location to be dumped. Normally type-cheating will be needed to specify this parameter.

5- bit(24)

The address of the last location to be dumped. Normally type-cheating will be needed to specify this parameter.

Description

See MTS Volume 3

Note

| Member Standard_Dump defines a macro that may be useful to
| interface to this routine.
|
| Constants are defined for return codes and the switches
| parameter.

macro Storage_Dump

Purpose

| To interface to the SDUMP subroutine.

Parameters

1- reference, value, Sdump_Bits_Type

Contains the switches.

2- Sprint_Procedure_Type

Specifies procedure to be called for each line of the
| output. It must be declared as Sprint_Procedure_Type with
| linkage "QSACHAIN".

3- name, unknown

Location containing the address of the first location to be dumped.

4- name, unknown

Location containing the address of the last location to be dumped.

Description

This macro just type cheats to allow use with arbitrary parameters, and provides the workarea required by SDUMP.

procedure System#

Purpose

To terminate execution successfully.

Description

See MTS Volume 3

procedure Time

Purpose

To obtain the elapsed time, CPU time used, time of day, and the date in various formats.

Parameters

1- reference, value, Integer

A key specifying the item to be returned.

2- reference, value, Integer

A switch indicating whether the item is to be returned and/or printed on SPRINT.

3- reference, unknown

The location at which the requested item is to be returned. The type required depends on the item requested.

Description

See MTS Volume 3

Note

The macro Get_Time_And_Date may be useful in interfacing to this routine.

procedure Timntrp

Purpose

To enable, disable, or return from timer interrupts set by the Setime subroutine.

Parameters

1- system procedure

A procedure to be called when a timer interrupt occurs, or Null.

2- reference, unknown

An area to save the registers and PSW when the interrupt occurs. This will normally be a variable of type Mts_Exit_Area_Type (see Exit_Definitions).

Description

See MTS Volume 3

Note

The macro Set_Exit may be used to set up a PLUS procedure as a timer interrupt exit routine.

procedure Trmtyp

Purpose

To return the type and, optionally, the name associated with the terminal attached to the current task.

Parameters

1- reference, character(8)

The "remote type" of the terminal. For network-attached devices, this is the type as specified to the network.

2- optional, reference, character(24)

The name of the terminal, or network address for network devices.

3- optional, reference, character(4)

The type of the device as known to the host (MTS). For network devices, this is the type of network connection.

|
|
| 4- optional, reference, character(4)
|
| The name of the device as known to the host. For network-
| attached devices, this is a network-connection name.
|
| Description
|
| This routine is used to determine the name and type of
| terminal at which the user is signed on. Generally only
| the first two parameters are of any interest.

procedure Trunc

Purpose

To deallocate unused space at the end of a file previously allocated to the file.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the logical unit or Fdub.

Description

See MTS Volume 3

procedure Twait

Purpose

To wait for a specified real time interval.

Parameters

1- reference, value, Integer

A code specifying the meaning of parameter 2. See below.

2- reference, value, unknown

Indicates how long to wait. Depending on the value of parameter 1, this may be bit(64) or character(16).

Description

See MTS Volume 3

Note

Member Twait also defines the following constants, which may be used for as parameter 1:

Microsec_From_Call (0)
Julian_Microsec (1)
Ebcdic_Time_And_Date (2)

procedure Unlk

Purpose

To request that a file be unlocked.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the logical unit or Fdub.

Description

See MTS Volume 3

procedure Unload

Purpose

To unload what was loaded on some previous call to the Load subroutine.

Parameters

1- reference, value, unknown

The type of this parameter is determined by parameter 3. It will normally be a fullword or a fixed-length string type.

2- reference, value, Integer

Specifies a storage index number (used only if parameter 1 is 0).

3- reference, value, Integer

Specifies the nature of the first parameter. See below.

Description

See MTS Volume 3

Note

Member Unload also defines the following constants that may be used for the third parameter:

Unload_Fdname (0)
Unload_Symbol (1)
Unload_Address (2)

procedure Write

Purpose

To write an output record on a specified logical I/O unit.

Parameters

1- reference, value, unknown

Specifies the starting location of the buffer.

2- reference, value, Short_Integer

Specifies the number of bytes to transmitted.

3- reference, value, Mts_Io_Modifiers_Type

The modifiers to be used.

4- reference, Mts_Line_Number_Type

The line number to use (if indexed), or line written (if returnlinenumber modifier).

5- reference, value, Mts_Io_Unit_Type

Specifies the unit or Fdub to use.

Result

optional, numeric

The return value if notification or noprompt is requested.

Description

See MTS Volume 3

procedure Writebuf

Purpose

To write out all changed file buffers.

Parameters

1- reference, value, Mts_Io_Unit_Type

Specifies the logical unit or Fdub.

Description

See MTS Volume 3

procedure Xctl

Purpose

To effect the dynamic loading and execution of a program.

Parameters

The parameters to Xctl are the same as the parameters to Linkf.

Description

See MTS Volume 3

INDEX

INDEX