UOS Utility Library Reference

Table of contents

Title Pasge

UOS Utility Library Reference

October 2023

Preface

Preface

Intended Audience

This document describes utility routines provided by UOS. It is intended for developers writing programs for UOS.

DMG

DMG (Display Management)

The DMG services are used to write to displays in a device-independent way. Each service begins with DMG$.

DMG$Get_Key

DMG$Get_Key

Get next keystroke.

Parse the next keystroke from an input string.

Format

DMG_Get_Key input result length

Parameters

input

A pointer to an SRB that points to the input string to get the next keystroke from.

result

A pointer to an SRB that points to the buffer to receive the keystroke. This should be large enough to contain any key name that the service might return. The length of the SRB indicates the maximum length, in bytes, of the buffer the SRB points to. On return, the length is updated to the number of bytes returned, which will never be larger than the length passed. If the buffer is too small, as much is returned as allowed by the size of the buffer and the SS_BUFFEROVF warning is returned. If 0 is passed, no result is returned.

length

Pointer to a 64-bit integer to receive the length of the escape sequence or UTF-8 characters for the returned result. If 0 is passed, no length is returned.

Description

DMG_Get_Key returns the next key press from the input string. If the key corresponds to a character, that single character is returned. If the key is a non-character key (such as PAGE DOWN) indicated by an escape code, the name of the key is returned. The following standard special keys can be returned by this service:

DELETE
DOWN
END
F0
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
F13
F14
F15
F16
F17
F18
F19
F20
HOME
INSERT
LEFT
PAGEDOWN
PAGEUP
PF1
PF2
PF3
PF4
RIGHT
SELECT
UP

Not all keyboards have all of these keys and some OEM keyboards may return additional key names.

Condition codes returned:

Code	**Meaning**
SS_BUFFEROVF	indicates that the result was larger than the provided buffer
SS_NORMAL	Successful completion.

FS

FS (File System)

The file system services support the creation and access of a UOS file system within a file. All File System services start with FS$.

A file system file is a file store that contains a complete UOS file system. A new file system can be created in a new file and various file system operations can be performed on the file system in the file. The FS$Open service opens an existing file system file, or creates a new one, and returns an integer context which is used by other FS$ service routines. When done with a file system file, the FS$Close routine should be called. For clarity, a reference to the "FS File" indicates the file that contains the file system within a file. "File system file" indicates a file within the FS File's file system.

FS$Close

FS_Close

Close an FS file

FS_Close closes an open FS file.

Format:

FS_Close context

Returns:

Returns a condition code.

Argument:

context

A pointer to the context value returned by FS_Open that indicates the FS file to close.

Description

Closes a FS file opened with FS_Open. This should always be called when you are finished with a FS file.

Condition Codes Returned:

Code	Meaning
FS_INVCONT	Specified context is not valid.
SS_NORMAL	Successful completion.

FS$Create_File

FS$Create_File

Create a file in a FS file

FS$Create_File creates a file in an open FS file.

Format:

FS_Create_File context name info

Returns:

Returns a condition code.

Argument:

context

A pointer to the context value returned by FS_Open that indicates the FS file to affect.

name

The address of a SRB structure that points to the full file specification of the file to create. The name must be a complete valid path and cannot be null.

info

The address of a UOS_File_Info structure that contains information about the file to create. The structure has the following format:

Off set	Size in bytes	Description
0	8	Size of file, in bytes.
8	8	Logical end of file (byte offset)
16	8	reserved. Should be 0.
24	4	Clustersize, in bytes. Must be a power of 2.
28	4	reserved. Should be 0.
32	8	Creation date/timestamp.
40	8	Last modified date/timestamp.
48	4	Creator ID.
52	4	Owner ID.
56	8	Flags. The FAF_* constants can be used for this, but several of them have no meaning in this context.
64	4	Version limit.
68	8	reserved. Should be 0.
76	8	reserved. Should be 0.
84	8	reserved. Should be 0.
92	8	reserved. Should be 0.

The Creator ID, Owner ID, and Version limit are user-defined.

Description

Creates a file in a FS file opened with FS_Open.

Condition Codes Returned:

Code	Meaning
FS_INVCONT	Specified context is not valid.
FS_INVNAME	Specified name is not valid.
SS_NORMAL	Successful completion.

FS$Delete_File

FS$Delete_File

Delete a file in an FS file

FS$Delete_File deletes a file from an open FS file.

Format:

FS$Delete_File context name

Returns:

Returns a condition code.

Arguments:

context

A pointer to the context value returned by FS_Open that indicates the FS file to affect.

name

The address of a SRB structure that points to the full file specification of the file to delete from the file system. The name must be a complete valid path and cannot be null.

Description

Deletes a file from a FS file opened with FS_Open.

Condition Codes Returned:

Code	Meaning
FS_INVCONT	Specified context is not valid.
FS_INVNAME	Specified name is not valid.
SS_NORMAL	Successful completion.

FS$Expand

FS$Expand

Alter FS file size

FS$Expand Alters the size of an open FS file.

Format:

FS$Expand context size

Returns:

Returns a condition code.

Argument:

context

A pointer to the context value returned by FS_Open that indicates the FS file to affect.

size

The address of an 8-byte integer that contains the new size of the FS file, in bytes. This cannot be smaller than the current FS file size.

Description

Expands the size of a FS file opened with FS_Open. This cannot be used to reduce the size of the file. The actual amount of expansion will depend upon the FS File system's clustersize and the result may be larger than requested. If the FS file size cannot be changed, an error code indicating the problem will be returned. For instance, if the calling process lacks necessary privileges or quotas, or if there isn't enough room on the store for the additional FS file space requirements.

Condition Codes Returned:

Code	**Meaning**
FS_INVCONT	Specified context is not valid.
FS_BADSIZE	The specified size is not supplied or is smaller than the current FS file size.
FS_NOCHNG	The FS file size was not changed.
SS_NORMAL	Successful completion.

Other errors can be returned if the FS file resize has an error.

FS$Open

FS_Open

Open FS file

FS_Open creates a new FS file or opens an existing one.

Format:

FS_Open context name flags allocation label password clustersize fclustersize

Returns:

Returns a condition code.

Arguments:

context

A pointer to the context value to be returned by FS_Open. The context is not set if the service fails.

name

The address of a SRB structure that points to the full file specification of the FS file to create or open. The name cannot be null. If the file doesn't exist, it is created and a new file system is initialized in it.

flags

The address of an 8-byte integer containing the open flags to use for the FS file.

allocation

The address of an 8-byte integer containing the size of the new file, in bytes, if a new file is being created. Otherwise, it is ignored.

label

The address of an SRB structure that points to a label for the file system. Only used when a new file system is created. It is ignored otherwise.

password

The address of an SRB structure that points to a password for the file system. On creation, it is applied to the file system. For existing file systems, this must match the file system's existing password.

clustersizs

The address of an 8-byte integer containing the default clustersize for files created in the file system, in bytes. It must be a power of 2. Only used when creating a new file system. Ignored otherwise.

fclustersizs

The address of an 8-byte integer containing the default clustersize for folders created in the file system, in bytes. It must be a power of 2. Only used when creating a new file system. Ignored otherwise.

Description

FS_Open can open an existing file containing a file system, or create a new one. When creating a new one, the arguments can be used to define the characteristics of the file system, including default clustersizes, passwords, and so forth. When the caller is finished with the FS file, FS_Close should be called to release the FS file and associated resources.

Condition Codes Returned:

Code	Meaning
FS_INVCONT	Specified context is not valid.
FS_INVFLAG	One or more invalid flags were specified.
FS_INVNAME	Specified name is not valid.
SS_NORMAL	Successful completion.

Also, any errors that can result from opening or creating the FS file may be returned.

FS$Open_File

FS$Open_File

Open a file in a FS file.

FS$Open_File opens a file within the file system in an open FS file.

Format:

FS$Open_File context name result

Returns:

Returns a condition code.

Argument:

context

A pointer to the context value returned by FS_Open that indicates the FS file to affect.

name

The address of an SRB structure that points to the fully-qualified file name within the file system. This must be provided and cannot be null.

result

The address of an 8-byte integer which will receive the address of a UOS_File instance.

Description

Opens the specified file within the file system of the FS file associated with the context. The file is opened in read/write mode. The file must exist and an error is returned if it does not. No UOS_File instance is returned if there is an error. The returned file instance is an object instance with a CORBA-compliant VMT. The methods of this instance are as follows:

Prototype	Description

TUnified_Exception* Initialize()	Initializes the object.
TUnified_Exception* Terminate()	Terminates the object.
void Attach()	Increments the reference count.
bool Detach()	Decrements the reference count. Returns true if the object is completed dereferenced.
bool Is_Class(char* Name)	Returns true if the passed name matches the name of the class or one of its ancestors.
int32 Interface_Version()	Version of interface.
int32 Facility_ID()	Facility ID for object.
int32 Facility_Version()	Facility version for the object.
TDebug_Interface* Debugger()	Returns nil.
TUnified_Exception* Last_Error()	Returns last exception.
void* Extension()	Returns nil.
constructor Create()	Constructor for the object.
destructor Destroy()	Destructor for the object.
TUnified_Exception* Set_Last_Error(TUnified_Exception* E)	Sets an exception.
int32 Create_Stream(char* Name)	Creates an ancillary data stream with the specified name. Returns the stream index.
void Delete_Stream(char* Name, int32 Index)	Deletes an ancillary data stream with either the specified name or index.
char* Stream_Name(int43 Index)	Returns the name of the specified ancillary data stream.
int32 Max_Stream()	Returns the highest ancillary data stream index.
int64 Read(int32 Stream, int64 Position, int64 Length, void* Buff, int43 Flags)	Reads the specified stream at the specified position for the specified length into the provided buffer. Returns the number of bytes actually read.
int64 Write(int32 Stream, int64 Position, int64 Length, void* Buff, int43 Flags)	Writes the specified stream to the specified position for the specified length from the provided buffer. Returns the number of bytes actually written.
int64 Get_Stream_Size(int32 Stream)	Returns the size, in bytes, of the ancillary data stream with the passed index.
void Set_Stream_Size(int32 Stream, int64 Value)	Sets the specified ancillary data stream to the specified size, in bytes.
bool Get_Contiguous()	Returns true if the file is contiguous.
void Set_Contiguous(bool Value)	Sets the file to contiguous/non-contiguous.
int64 Get_File_Size()	Returns the file's data size, in bytes.
void Set_File_Size(int64 Value)	Sets the file's data size.
bool Read_Only()	Returns true if the file is set to read-only.
bool Write_Only()	Returns true if the file is set to write-only.

int64 XSpaceAvail()	Returns the maximum possible space available to the file, in bytes. This includes space on the store that could be used to extend the file.
bool Is_Store()	Returns true if the file is a store. This will always return true for a file in a FS file.
char* Name()	Returns the name of the file.
int64 ID()	Returns the internal File ID.
int64 Last_Error_Code()	Returns the error code for the last operation (0 = no error).
int64 Owner()	Returns the Owner ID.
int64 ACL()	Returns the ACL pointer.
int64 Flags()	Returns the file flags.

These methods all use the C calling standard (stdcall). When finished with this object instance, be sure to call Detach(), which will free the instance when all other references to the object are released. Note that if the FS file is closed with FS$Close, the object is no longer valid.

Condition Codes Returned:

Code	Meaning
FS_INVCONT	Specified context is not valid.
FS_INVNAME	Specified name is not valid.
SS_NORMAL	Successful completion.

Other errors can be returned.

HashLib

Hashlib

Hashlib provides routines for hashing and encryption algorithms. It also supports dynamically adding new algorithms to the available library.

Hash$Count

HASH$Count

Returns the highest hash index of the installed hash algorithms.

Format

HASH$Count

Description

This service returns the highest index of the installed hash algorithms as a 64-bit integer.

Condition Values Returned

No code returned; the result value is the highest hash algorithm index.

Hash$Default

HASH$Default

Returns the index of the default hash algorithm to use for secure hash operations.

Format

HASH$Default

Description

This service returns the index of the default hash algorithm used for security purposes.

Condition Values Returned

No code returned; the result value is the index of the default hash algorithm.

Hash$Hash

HASH$Hash

Returns the hash of a string, according to the specified hash algorithm.

Format

HASH$Hash index, source, target, key, resultlength

Arguments

index

Pointer to a 32-bit integer value that indicates the hash algorithm to use. This must match an installed algorithm. The following hash algorithms are provided with UOS by default.

Mnuemonic	**Description**
Hash_Plaintext	No hashing is done. The service returns the passed string, unaltered.

Secure hashes

Hash_MD5	MD5 hash
Hash_SHA1	Original SHA hash
Hash_SHA224	224 bit SHA hash
Hash_SHA256	256 bit SHA hash
Hash_SHA384	384 bit SHA hash
Hash_SHA512	512 bit SHA hash
Hash_HMAC_MD5	MD5 keyed hash
Hash_HMAC_SHA1	SHA keyed hash
Hash_HMAC_SHA256	256 bit SHA keyed hash
Hash_HMAC_SHA512	512 bit SHA keyed hash

Error checking hashes

Hash_Checksum	32 bit checksum
Hash_XOR8	8 bit XOR hash
Hash_XOR16	16 bit XOR hash
Hash_XOR32	32 bit XOR hash
Hash_CRC16	16 bit Cyclic Redundancy Check
Hash_CRC32	32 bit Cyclic Redundancy Check
Hash_Adler32	32 bit Adler hash

Hashtable hashes

Hash_ELF	ELF hash
Hash_Knuth	Donald Knuth hash

Note: when security is needed, only Hash_SHA384, Hash_SHA512, or Hash_HMAC_SHA512 should be used. Other "secure" hashes are not considered to be strong enough to secure passwords.

source

A pointer to an SRB which points to the data string to hash.

target

A pointer to an SRB which points to the buffer to receive the hashed data. The buffer should be large enough to contain the resulting hash.

key

A pointer to an SRB which contains a key to use for keyed hashes. This is ignored if a non-keyed hash is specified.

resultlength

A pointer to where a 32-bit integer value can be written. The value written indicates the number of bytes written to the target buffer. If the data is truncated due to the buffer being too small to receive all of the data, UOSErr_Buffer_Overflow is returned by the service and the value written to this location will be the size of the data actually written (the receiving buffer size).

Description

This service returns a hashed value of the passed data. In the case of Hash_Plaintext, the service returns a copy of the passed string. If the calling code does not know the result size of the hash, it should check the resulting length; if the result length is equal to the receiving buffer size, the receiving buffer should be increased and the hash attempted again. This should be repeated until the result length is less than the receiving buffer size, which will guarantee that the entire hash is returned. Note that the above hash sizes indicate the number of bits in the result.

Condition Values Returned

Value	**Meaning**
SS_NORMAL	Normal completion
HASHLIB_Invalid_Hash	The specified hash index did not match an installed hash algorithm.
HASHLIB_No_Data	Either or both of the source and target buffers was not provided.
UOSErr_Buffer_Overflow	The receive buffer was too small to contain the entire hash value.

Hash$Index

HASH$Index

Returns the hash algorithm index corresponding to the specified name.

Format

HASH$Index name result

Arguments

name

A pointer to an SRB which points to the hash algorithm name to look up. The comparison to installed algorithm names will be done case-insensitively. This is always an ASCII string and not interpreted as UTF-8.

result

A pointer to a 32-bit buffer where the index will be written.

Description

This service returns the index of the hash algorithm corresponding to the passed name.

Condition Values Returned

Value	Meaning
SS_NORMAL	Successful completion.
HASHLIB_Invalid_Hash	The specified hash name did not match an installed hash algorithm.

Hash$Name

HASH$Name

Returns the hash algorithm name of the specified index.

Format

HASH$Name index result

Arguments

index

A pointer to a 32-bit integer containing the hash algorithm index. The lowest valid hash algorithm index is 0, and HASH_Count returns the total count of algorithms.

result

A pointer to an SRB which will receive the name of the algorithm corresponding to the specified index. This buffer should be 64 bytes long. The returned name will terminate with an ASCII 0 (NUL). The name will never exceed 64 bytes in length, including the trailing NUL. If the buffer is too small, a UOSErr_Buffer_Overflow error occurs, and the returned name will not terminate with a NUL.

Description

This service returns the name of the hash algorithm corresponding to the passed index value.

Condition Values Returned

Value	Meaning
SS_NORMAL	Successful completion.
HASHLIB_Invalid_	The specified hash index did not match an installed hash algorithm.

Hash	
UOSErr_Buffer_Overflow	The receive buffer was too small to contain the entire hash algorithm name.

LBR

LBR

All librarian service routine names are prefixed with "LBR$". To create a new library or use an existing library, the LBR$INI_CONTROL routine must be called to obtain a context number. This is passed to the other LBR$ routines and allows multiple libraries to be accessed without having to close one to open a new one. LBR$Open is then used to open a library file. LBR$Close is used to close a library file when you are finished with it. LBR$OUTPUT_HELP can be used to display help to the user and query him for more information, without the need to use LBR$INI_CONTROL, LBR$OPEN, or LBR$CLOSE.

LBR$Close

LBR$Close
Close library

This service closes an open library file.

Format:
LBR$Close context

Returns:
Returns a condition code.

Argument:
context
The context value returned by LB$INI_CONTROL that indicates the library to close.

Description
Closes a library file opened with LBR$OPEN. This should always be called when you are finished with a library file.

Condition Codes Returned:

Code	Meaning
LBR_ILLCTL	Specified context is not valid.
LBR_LIBNOTOPN	Specified library is not open.
SS_NORMAL	Successful completion.

LBR$Close_Control

LBR$Close_Control

Close a library context

LBR$Close_Control closes a context.

Format:

LBR$Close_Control context

Returns:

Returns a condition code.

Arguments:

context

The address of a 64-bit context value returned by LBR_INI_CONTROL that indicates the library to access.

Description

The specified context is closed and any allocated memory is released.

Condition Codes Returned:

Code	Meaning
LBR_ILLCTL	Specified context is not valid.
SS_NORMAL	Successful completion.

LBR$Delete

LBR$Delete

Delete a module

LBR$Delete deletes a module from an open library file.

Format:

LBR$Delete context name

Returns:

Returns a condition code.

Arguments:

context

The address of a 64-bit context value returned by LBR$INI_CONTROL that indicates the library to close.

name

The name of the module to delete. This is the address of a TSRB structure that points to the name of the module.

Description

Deletes the specified module from the library.

Condition Codes Returned:

Code	Meaning
LBR_ILLCTL	Specified context is not valid.
LBR_LIBNOTOPN	Specified library is not open.
LBR_NOTFO	Specified module does not exist.

UND	
SS_NORMAL	Successful completion.

LBR$Get_Help

LBR$Get_Help

Retrieve help text

LBR$Get_Help retrieves help text.

Format:

LBR$Get_Help context {line_width} {routine} {data} {key}

Returns:

Returns a condition code.

Arguments:

context

The address of a 64-bit context value returned by LB_INI_CONTROL that indicates the library to flush.

line_width

The address of a 64-bit value indicating the maximum line width for displayed text. The data passed to the routine will be broken into lines of this length using CRLF characters. Note that any single word that exceeds the line width will be on a line by itself and will exceed the line-length (it is not truncated to fit). A "word" is delimited by whitespace (one or more spaces and/or tabs). If a value of 0 is used, the text is returned as-is, with no inserted line breaks. However, line breaks within the module are not removed.

routine

The address of a callback routine. This routine is called each time help text is to be displayed. Two parameters are passed to the routine. The first is the value passed in the data argument. The second is the address of a TSRB structure that points to the complete help text. This data is only valid until the callback routine returns.

data

The address of a 64-bit integer value to be passed to the callback routine.

key

The address of a TSRB structure that points to the name of the help module to be returned. If the name is "*", the module named "Index" and the modules referenced by it will be returned. If the name is "*...", all modules are returned, in no particular order. If the key is 0 or name is null, the root module is returned. By convention, the module named "Index" is considered to be the root of the help library. If no key is given and no module named "Index" is found, an error is returned.

Description

Returns the specified help module text. Note that most application programs use LBR_OUTPUT_HELP to display help.

Condition Codes Returned:

Code	**Meaning**
LBR_ILLCTL	Specified context is not valid.
LBR_LIBNOTOPN	Specified library is not open.
LBR_NOTHLPLIB	Specified library contains no help modules.

LBR_NOTFOUND	Specified module is not found in the library.
SS_NORMAL	Successful completion.

LBR$Get_Module

LBR$Get_Module
Get module from library

LBR$Get_Module retrieves a module from a library.

Format:

LBR$Get_Module context {routine} {data} {key}

Returns:

Returns a condition code.

Arguments:

context

The address of a 64-bit context value returned by LBR_INI_CONTROL that indicates the library to access.

routine

The address of a callback routine. This routine is called to return the requested data. Two parameters are passed to the routine. The first is the value passed in the data argument (0 if not passed). The second is the address of a TSRB structure that points to the module data. This data is only valid until the callback routine returns.

data

The address of a 64-bit integer value to be passed to the callback routine.

key

The address of a TSRB structure that points to the name of the module to be returned. If the key is 0 or name is null, an error is returned.

Description

Returns the specified module data.

Condition Codes Returned:

Code	**Meaning**
LBR_ILLCTL	Specified context is not valid.
LBR_LIBNOTOPN	Specified library is not open.
LBR_NOTFOUND	Specified module is not found in the library.
SS_NORMAL	Successful completion.

LBR$Ini_Control

LBR$Ini_Control

Initialize library access

LBR$Ini_Control initializes access to a library.

Format:

LBR$Ini_Control context function {type} {name} {default} {result} {length}

Returns:

Returns a condition code.

Arguments:

context

The address of a 64-bit integer to receive the context created by this call.

function

The address of a 64-bit value indicating the access type. The values are:

Mnuemonic	Meaning
LBR_C_Create	Create a new library file.
LBR_C_Read	Open library for reads only.
LBR_C_Update	Open library for reads and writes.

type

The address of a 64-bit value indicating the type of library. The value must be one of the following:

Mneumonic	Description
LBR_C_TYP_OBJ	Objects
LBR_C_TYP_SHSTB	Sharable image
LBR_C_TYP_MLB	Macros
LBR_C_TYP_HLP	Help
LBR_C_TYP_TXT	Text
LBR_C_TYP_UNK	Unknown
LBR_C_TYP_NCS	NCS

A negative integer indicates a user-developed library.

name

The address of a TSRB structure that points to the name of the library.

default

The address of a TSRB structure that points to the default name of the library. Any missing fields in the name are substituted from this specification.

result

The address of a TSRB structure that points to the a buffer to receive the full filename of the library file.

length

The address of a 64-bit integer to receive the length of the result filename. Note that this will not exceed the length specified in the TSRB pointed to by the result argument.

Description

LBR_Ini_Control must be called before any other LBR_ service, except for LBR_OUTPUT_HELP. After calling, the LBR_OPEN service must be used to open or create the library. When done, the library should be closed with LBR_Close. If the call fails, the returned context is 0 and the service returns a condition code indicating the problem. Otherwise, the returned context is the value that must be used for other LBR_ calls.

Condition Codes Returned:

Code	Meaning
LBR_ILLFUNC	Requested function is not valid.
LBR_ILLTYP	Specified library type is not valid.
LBR_NOFILNAM	Error. No filename was specified.
SS_NORMAL	Successful completion.

LBR$Module_Exists

LBR$Module_Exists

Check if module exists

LBR$Module_Exists indicates if a module exists in a library.

Format:

LBR$Module_Exists context name {result}

Returns:

Returns a condition code.

Arguments:

context

The address of a 64-bit context value returned by LBR_INI_CONTROL that indicates the library to access.

name

The address of a TSRB structure that points to the name of the module. If the value is 0 or points to a null name, an error is returned.

result

The address of a 64-bit integer value to be set.

Description

The integer at the specified address is set indicating whether or not the module exists. If the module exists the value is set to 1, otherwise it is set to 0.

Condition Codes Returned:

Code	Meaning
LBR_ILLCTL	Specified context is not valid.

LBR_LIBNOTOPN	Specified library is not open.
SS_NORMAL	Successful completion.

LBR$Open

LBR$Open
Open or create a library

LBR$Open is used to open an existing library or to create a new library.

Format:

LBR$Open context {options} {name}

Returns:

Returns a condition code.

Arguments:

context

The address of a 64-bit context returned by LIB_Ini_Control.

options

The address of a 64-bit value indicating the creation options. These options are a contiguous array of 64-bit integers containing the following items:

Offset	Description
0	Allocation size, in bytes, for the library file.
1	Internal clustersize, in bytes.
2	Reserved for furture use. Should be 0.

name

The address of a TSRB structure that points to the name of the library. If null, the name passed to LBR_INI_Control is used.

Description

The LBR_OPEN service must be used to open or create a library. When done, the library should be closed with LBR_Close.

Condition Codes Returned:

Code	Meaning
LBR_ILLCTL	Specified library control index is not valid.
LBR_ILLFMT	Specified library format is not valid.
LBR_LIBOPN	Specified library is already open.
SS_NORMAL	Successful completion.

LBR$Output_Help

LBR$Output_Help
Display help text

LBR$Output_Help is used to display help.

Format:

LBR$Output_Help routine {width} {path} {library} {flags} {input}

Returns:

Returns a condition code.

Arguments:

routine

The address of a routine to handle the output of help text. If 0, the service writes the data to sys$output. The routine takes one 64-bit address of a TSRB that contains a pointer to the article text.

width

The address of a 64-bit integer indicating the maximum line width of the output text. If not specified, or 0, the lines are limited based on the output device and the source text. If written to a target with no inherent width (such as a store file) the lines are limited to 80 characters.

path

The address of a TSRB that points to a help path. Generally this is simply the module name of the help text to display, but it can be used to provide a navigation path, with each item in the path indicating a module name, delimited by backslashes (\), which will allow the user to "back up".
If the value begins with an at-sign (@), the text following that is treated as a help library filename that will be used instead of the library argument. If Index or HELP is specified, the index module is used as the initial help text. If there is no Index module, the module displayed may be indeterminate.

library

The address of a TSRB structure that points to the name of the main help library.

flags

The address of a 64-bit integer containing processing flags:

Mnuemonic	**Description**
HLP_M_PROMPT	If not specified, an input routine must be specified to handle prompting the user.
HLP_M_PROCESS	The process symbol name table is searched for default help.
HLP_M_GROUP	The group symbol name table is searched for default help.
HLP_M_SYSTEM	The system symbol name table is searched for default help.
HLP_M_LIBLIST	A list of default libraries available is displayed for the user to choose from.

Multiple help libraries in addition to the main library can be used by defining HLP$LIBRARY, HLP$LIBRARY_1, HLP$LIBRARY_2, and so forth in the symbol tables, each symbol indicating a help library file name. These are used to resolve any anchor tags that refer to a module that is not in the main help library. If more than one of the HLP_M_PROCESS, HLP_M_GROUP, and/or HLP_M_SYSTEM flags are specified, searches are done in the order: process, group, system.

input

The address of a routine that is called when user input is needed. A 64-bit address is returned by the routine of a TSRB structure that points to the response text. If this is not specified or 0, HLP_M_PROMPT is required. If HLP_M_PROMPT is specified, this argument is ignored.

Description

LBR_Output_Help is used to display help text. A program can leave the entire processing of help to the service by providing no routine or input arguments and specifying HLP_M_PROMPT in the flags. LBR_INI_CONTROL, LBR_OPEN, and LBR_CLOSE are all handled internally.
In non-GUI settings, the default input will prompt the user differently depending on the situation. The default prompt for the user is "Topic?", though this can be changed by the input routine.

The user response to the prompt is handled this way:

User response	Behavior of LBR$OUTPUT_HELP
?	Displays an alphabetic list of all topics (module names) in the help library.
Ctrl-Z	Immediately exit interactive help session.
Enter	Return to previously viewed help module. If there are no previously viewed modules, exit the interactive help session.
name	Any other response is interpreted as a module name to display. If the specified module isn't found, an error is shown and the user is prompted again. The main library is searched first, then the other libraries are searched if the topic isn't found.

Condition Codes Returned:

Code	Meaning
LBR_ILLINROU	Input routine improperly specified or omitted.
LBR_NOHLPLIS	Error. No default help libraries can be opened.
LBR_INVPATH	Part of the path contains invalid characters.
SS_NORMAL	Successful completion.

LBR$Put_Module

LBR$Put_Module

Create new library module

LBR$Put_Module creates a new module in the library or overwrites an existing one with the same name.

Format:

LBR_Put_Module context name module

Returns:

Returns a condition code.

Arguments:

context

The context value returned by LBR$INI_CONTROL that indicates the library to modify.

name

The address of a TSRB structure which points to the name of the module to write. This must not be null and must be a valid UOS filename (without wildcards).

module
The address of a TSRB structure which points to the data to be written to the module.

Description
LBR_Put_Module writes a memory image of a module to the library.

Condition Codes Returned:

Code	Meaning
LBR_ILLCTL	Specified library control index is not valid.
LBR_LIBNOTOPN	Specified library is not open.
LBR_INVNAME	The name is null or does not conform to a valid UOS filename. SS_NORMAL Normal completion.

Appendix A: UHTML

Appendix A

UHTML

This appendix describes the subset of HTML that UOS supports natively for application programs. This is referred to as UHTML.

HTML stands for HyperText Markup Language and is a world standard that is widely used. UHTML follows the formatting rules of HTML with the only differences being which tags and entities are recognized by UOS.

Non UHTML tags are ignored by UOS HTML processing. All UHTML text is assumed to he UTF-8, with less than (<) and ampersand (&) being reserved characters. Multiple contiguous spaces are reduced to single spaces on output. Carriage Return and Linefeeds are ignored. Forcing a new line is done by using the <br> tag. Otherwise, text is automatically wrapped.

UHTML text is designed to be used to output formatted text to printers, terminals, and displays rather than be used on websites. As a consequence, there is no need for many of the tags used on web pages, including <body>, <script>, <meta>, <html>, and <doc>, among others. It also does not support CSS, although some tags support the STYLE attribute for some features.

Entities

Entities

Special characters are specified using an entity code which starts with an ampersand (&) and ends with a semicolon (;). To include a literal ampersand, the &amp; entity must be used. The following character entities are recognized by UHTML:

Entity code	Meaning
#RE	ASCII 13: carriage return
#RS	ASCII 10: linefeed
#TAB	ASCII 9: Horizontal tab
Aacute	A with acute accent (À)
aacute	a with acute accent (á)

Acirc	A with circumflex (Â)
acirc	a with circumflex (â)
acute	Acute accent (´)
aelig	Lowercase ae ligature (æ)
AElig	Capital AE ligature (Æ)
agrave	a with grave accent (à)
Agrave	A with grave accent (À)
amp	Ampersand (&)
Aring	A with ring (Å)
aring	a with ring (å)
atilde	a with tilde (ã)
Atilde	A with tilde (Ã)
auml	A with umlat (Ä)
auml	a with umlat (ä)
brvbar	Broken vertical bar (¦)
bull	Bullet (•)
Ccedil	Big C-cedilla (Ç)
ccedil	Small c-cedil (ç)
cedil	Cedilla (¸)
cent	Cent (¢)
circ	Cicumflex (^)
copy	Copyright (©)
curren	Currency (¤)
Dagger	Uppercase dagger (‡)
dagger	Lowercase dagger (†)
dbquo	Double subscript quote („)
deg	Degree (°)
divide	Divide (÷)
Eacute	E with acute accent (É)
eacute	e with acute accent (é)
ecirc	e with circumflex (ê)
Ecirc	E with circumflex (Ê)
Egrave	E with grave accent (È)
egrave	e with grave accent (è)
emul	e with umlat (ë)
ETH	Big Eth (Ð)
eth	Small eth
Euml	E with umlat (Ë)
euro	Euro (€)
fnof	Function of (ƒ)
frac12	1/2
frac14	1/4
frac34	3/4
gt	Greater than (>)
hellip	ellipse (…)
Iacute	I with acute accent (Í)
iacute	i with acute accent (í)
Icirc	I with circumflex (Î)
icric	i with circumflex (î)
iexcl	Inverted exlamation (¡)

Igrave	I with grave accent (Ì)
igrave	i with grave accent (ì)
iquest	Inverted question mark (¿)
Iuml	I with umlat (Ï)
iuml	i with umlat (ï)
laquo	Left angle quote
ldquot	Left double quote (“)
lsaquo	Left single angle quote (‹)
lsquo	Left single quote (‘)
lt	Less than (<)
macr	Macron (¯)
mdash	M-dash (—)
micro	Micro (µ)
middot	Middle dot (·)
mu	Lowercase mu (µ)
nbsp	Nonbreaking space
ndash	N-dash (–)
not	Not (¬)
ntilde	Lowercase n with tilde (ñ)
Ntilde	Capital N with tilde (Ñ)
oacute	o with acute accent (ó)
Oacute	O with acute accent (Ó)
Ocirc	O with circumflex (Ô)
ocirc	o with circumflex (ô)
oelig	Lower case oe ligature (œ)
OElig	Uppercase OE ligature (Œ)
Ograve	O with grave accent (Ò)
ograve	o with grave accent (ò')
ordf	Feminine ordinal indicator (ª)
ordm	Masculine ordinal indicator (º)
oslash	Lowercase o with slash (ø)
Oslash	O with slash (Ø)
Otilde	O with tilde (Õ)
otilde	o with tilde (õ)
Ouml	O with umlat (Ö)
ouml	o with umlat (ö)
para	Paragraph (¶)
permil	Parts per million (‰)
plusmn	Plus/minus (±)
pound	English pound (£)
raquo	Right angle quote
rdquot	Right double quote (”)
reg	Registered trademark (®)
rsaquo	Right single angle quote (›)
rsquo	Right single quote (')
sbquo	Subscript quote (‚)
scaron	Lowercase scaron (š)
Scaron	Uppercase Scaron
sect	Section (§)
shy	Soft hyphen

sup1	Superscript 1
sup2	Superscript 2
sup3	Superscript 3
szlig	sz ligature
thorn	thorn (þ)
THORN	Big thorn (Þ)
tilde	Tilde (~)
times	Multiplication (×)
trade	Trademark (™)
uacute	u with acute accent (ú)
Uacute	U with acute accent (Ú)
Ucirc	U with circumflex (Û)
ucirc	u with circumflex (û)
ugrave	u with grave accent (ù)
Ugrave	U with grave accent (Ù)
uml	Diaresis (umlat) (¨)
Uuml	U with umlat (Ü)
uuml	u with umlat (ü)
Yacute	Y with acute accent (Ý)
yacute	y with acute accent (ý)
yen	Japanese yen (¥)
Yuml	Y with umlat (Ÿ)
yuml	y wuth umlat (ÿ)

Tags

Tags

UHTML tags are special characters embedded in text that indicate how the text is to be rendered. The general format of a tag is:

< name {attribute{=value} ... } >

where "name" is the tag name. The optional "attribute" indicates some modifier to the behavior of the tag. Some tags do not allow attributes, while others may allow multiple attributes. Attributes have optional values that further modify the tab's behavior. Tags are of two types: single and paired. Paired tags have a starting tag and an ending tag and affect the contents of the text between the tags. The ending tag has the format:

< / name >

where "name" matches the name of the opening tag.

Paired tags must be properly nested. That is, when a starting tag appears between the starting and ending tags of another tag, it must end before the outermore tag ends. Examples:

<font size=+1> <b>Warning</b> </font>

Properly nested tags.

<font size=+1> <b>Warning </font> </b>

Improperly nested tag.

Bold

Bold text

Format: <b>...</b>

Alternate format: <strong>...</strong>

Description

These tags bold the text between them.

Examples:

<b>This is bold text</b>

<strong>Description</strong>

Big Text

Big Text

Format: <big>...</big>

Description

These tags increase the font size of the text between them. This is an alternative for using the <font> tag with a size attribute.

Examples:

<big>This is big text</big>

Blink

Blink

Format: <blink>...</blink>

Description:

The text between the tags is shown as blinking.

Example:

<blink>SECURITY ALERT!</blink>

Break

br

Format: <br>

Description:
Breaks the text at the position of the tag, starting a new line.

Example:
First line<br>second line

Blockquote

Blockquote

Format: <blockquote>...</blockquote>

Description:
The text between the tags is indented on the left and right.

Example:

```
<blockquote>
    Hail, foreign wonder!<br>
    Whom certain these rough shades did never breed.<br>
    Milton.
</blockquote>
```

Division

Division

Format: <div {attributes}>,,,</div>

Description:
Marks the section of text between the tags with certain attributes.

Attributes:

BGCOLOR: color
or
BACKGROUND-COLOR: color

Defines the color of the background, using the HTML color specification.

DIRECTION:dir

Defines the direction of the text. Valid values are LTR (left-to-right) or RTL (right-to-left). LTR is the default.

FLOAT:spec

> Allows text outside of the division to float around this division. The valid values are LEFT and RIGHT. FLOAT:LEFT places the division on the left side with the following text showing to the right of the division. FLOAT:RIGHT places the division on the right side with the outside text showing to the left of the division.

Examples:

> <div float:right><img src="image.png"></div>This text wraps to the left of the image.
>
> <div bgcolor:red>Warning</div>

Font

Font

Format: <font {attributes}>...</font>

Description:

> The text between the tags is given the specified font attributes.

Attributes:

COLOR

Format: Color=color

> Specifies the foreground color (that is, the color of the text). The color is an HTML color specification.

Example: <font color=red>Warning!</font>

FACE

Format: Face=name

> Specifies the name of the font to be used for the text.

Example: <font face="courier">monotyped text</font>

SIZE

Format: Size=value

Specifies the size of the text. value is an integer value. If the value is preceded by a plus (+) or minus (-), the font size is increased or decreased by the number of steps indicated by the value.

Example: <font size=+1>Bigger text</font>

Heading

Heading

Format: <hn>..</hn>

where "n" is an integer value from 1 to 3, inclusive. A value of 1 indicates the largest header, and 3 indicates the smallest header.

Examples:
<h1>Main header</h1>
<h2>Sub header</h2>
<h3>Sub-sub header</h3>

Image

Image

Format: <img attributes>

Description:

Includes the specified image at the tag's location.

Attributes:

SRC

Format: SRC="name"

where name is the file name of an image file. Supported image file formats are BMP, PNG, and JPG.

Example: <img src="image.png">

Italic

Italic

Format: <i>...</i>

Alternate format: <italic>...</italic>

Description:

Text between the tags is shown italicized.

Example:

He <i>supposed</i> it was on purpose.

Paragraph

Paragraph

Format: <p {attributes}>...</p>

Description:

The text between these tags is treated as a separate paragraph from the surrounding text.

Attributes:

DIRECTION

Format: direction=value

Indicates text direction for the paragraph where value is RTL for right to left and LTR for left to right. LTR is the default.

Strike

Strike

Format: <s>...</s>

Alternate format: <strike>...</strike>

Description:

The text between the tags is marked as strike-through (or strike-out).

Example:
We were <s>not</s> confined.

Span

Span

Format: <span {attributes}>,,,</span>

Description:
Marks the section of text between the tags with certain attributes.

Attributes:

BGCOLOR: color
or
BACKGROUND-COLOR: color

Defines the color of the background, using the HTML color specification.

DIRECTION:dir

Defines the direction of the text. Valid values are LTR (left-to-right) or RTL (right-to-left). LTR is the default.

Examples:

<span bgcolor:red>Warning</span>

Subscript

Subscript

Format: <sub>...</sub>

Description:

The text between the tags is subscripted.

Example:

Water is the chemical compound: H<sub>2</sub>O

Superscript

Superscript

Format: <sup>...</sup>

Description:

The text between the tags is superscripted.

Example:

The square of x is shown as x<sup>2</sup>.

Table

Table

Format: <table>...</table>

Description:

The text between the tags is displayed as a table. Each row within the table must begin with the <tr>

tag and end with </tr>. Each row has one or more columns that start with <td> and end with </td>. <th> and </th> may be used instead of <td> and </td> to display a column heading. Headings are bolded.

<td> and <th> can use the optional COLSPAN attribute, which indicates that the cell spans multiple columns. Format:
COLSPAN=number-of-columns-to-span.

Example:

```
<table>
<tr><th>Name</th><th>Birthday</th></tr>
<tr><td>Dianne</td><td>February 28</td></tr>
<tr><td>George</td><td>June 9</td></tr>
</table>
```

Underline

Underline

Format: <u>...</u>

Description:

The text between the tags is underlined.

Example:

This event is recorded in <u>The Journal of Entomology</u>, issue 28, page 17.

Color Specification

Color Specification

When specifying colors, the specification can be one of three formats, RGB, Hex, and Name.

RGB

Format: RGB(red, green, blue)

Where red, green, and blue indicate the values for each color component, which can be from 0 to 255.

Example:

RGB(255,0,0)

This indicates red.

Hex

Format: #rrggbb

Where rr is the hexadecimal value for red, gg is the hexadecimal value for green, and bb is the hexadecimal value for blue. Each value can be from 00 to FF.

Example:

#00FF00

This indicates the Green.

Name

Format: name

Where name is one of the following values:

AQUA
BLACK
BLUE
WHITE
FUCHSIA
GREEN
GREY or GRAY
GREYTEXT or GRAYTEXT
LIME
MAROON
NAVY
OLILVE
PURPLE
RED
SILVER
TEAL
YELLOW