PenPoint™ Application Programmatic Interface

Volume II
PenPoint™

PenPoint™
API Reference

VOLUME II
PenPoint Application Writing Guide provides a tutorial on writing PenPoint applications, including many coding samples. This is the first book you should read as a beginning PenPoint applications developer.

PenPoint Architectural Reference Volume I presents the concepts of the fundamental PenPoint classes. Read this book when you need to understand the fundamental PenPoint subsystems, such as the class manager, application framework, windows and graphics, and so on.

PenPoint Architectural Reference Volume II presents the concepts of the supplemental PenPoint classes. You should read this book when you need to understand the supplemental PenPoint subsystems, such as the text subsystem, the file system, connectivity, and so on.

PenPoint API Reference Volume I provides a complete reference to the fundamental PenPoint classes, messages, and data structures.

PenPoint API Reference Volume II provides a complete reference to the supplemental PenPoint classes, messages, and data structures.

PenPoint User Interface Design Reference describes the elements of the PenPoint Notebook User Interface, sets standards for using those elements, and describes how PenPoint uses the elements. Read this book before designing your application's user interface.

PenPoint Development Tools describes the environment for developing, debugging, and testing PenPoint applications. You need this book when you start to implement and test your first PenPoint application.
Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book and Addison-Wesley was aware of a trademark claim, the designations have been printed in initial capital letters.

The authors and publishers have taken care in preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

Copyright ©1991–92 GO Corporation. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada.

The following are trademarks of GO Corporation: GO, PenPoint, the PenPoint logo, the GO logo, ImagePoint, GOWrite, NoteTaker, TableServer, EDA, MiniNote, and MiniText.

Words are checked against the 77,000 word Proximity/Merriam-Webster Linguibase, ©1983 Merriam Webster. ©1983. All rights reserved, Proximity Technology, Inc. The spelling portion of this product is based on spelling and thesaurus technology from Franklin Electronic publishers. All other products or services mentioned in this document are identified by the trademarks or service marks of their respective companies or organizations.

PenTOPS Copyright © 1990-1992, Sitka Corporation. All Rights Reserved.

GO CORPORATION MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT, REGARDING PENPOINT SOFTWARE OR ANYTHING ELSE.

GO Corporation does not warrant, guarantee, or make any representations regarding the use or the results of the use of the PenPoint software, other products, or documentation in terms of its correctness, accuracy, reliability, currentness, or otherwise. The entire risk as to the results and performance of the PenPoint software and documentation is assumed by you. The exclusion of implied warranties is not permitted by some states. The above exclusion may not apply to you.

In no event will GO Corporation, its directors, officers, employees, or agents be liable to you for any consequential, incidental, or indirect damages (including damages for loss of business profits, business interruption, loss of business information, cost of procurement of substitute goods or technology, and the like) arising out of the use or inability to use the documentation or defects therein even if GO Corporation has been advised of the possibility of such damages, whether under theory of contract, tort (including negligence), products liability, or otherwise. Because some states do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitations may not apply to you. GO Corporation’s total liability to you from any cause whatsoever, and regardless of the form of the action (whether in contract, tort [including negligence], product liability or otherwise), will be limited to $50.

The PenPoint documentation is provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in FAR 52.227-19 (Commercial Computer Software—Restricted Rights) and DFAR 252.227-7013 (c) (1) (ii) (Rights in Technical Data and Computer Software), as applicable. Manufacturer is GO Corporation, 919 East Hillsdale Boulevard, Suite 400, Foster City, CA 94404.

ISBN 0-201-60863-4
123456789-AL-9695949392
First Printing, June 1992
Preface

The *PenPoint API Reference* provides reference information on the subsystems of the PenPoint™ operating system. Volume I describes the functions and messages that you use to manipulate classes and describes the fundamental classes used by almost all PenPoint applications. Volume II describes the supplemental classes and functions that provide many different capabilities to PenPoint applications. The text in this volume was generated from the header files in \PENPOINT\SDK\INC.

**Intended Audience**

The *PenPoint API Reference* is written for people who are developing applications and services for the PenPoint operating system. We assume that you are familiar with the C language, understand the basic concepts of object-oriented programming, and have read the *PenPoint Application Writing Guide*.

**What’s Here**

The *PenPoint API Reference* is divided into several parts, which are split across two volumes. Volume I contains these parts:

- **Part 1: Class Manager** describes the PenPoint class manager classes, which supports object-oriented programming in PenPoint.
- **Part 2: PenPoint Application Framework** describes the PenPoint Application Framework classes, which provides you the tools you use to allow your application to run under the notebook metaphor.
- **Part 3: Windows and Graphics** describes ImagePoint classes and how applications can control the screen (or other output devices).
- **Part 4: UI Toolkit** describes the PenPoint classes that implement many of the common features required by the PenPoint user interface.
- **Part 5: Input and Handwriting Translation** describes the PenPoint input system classes and classes that provide programmatic access to the handwriting translation subsystems.

Volume II contains these parts:

- **Part 6: Text Component** describes the PenPoint classes that allow any application to provide text editing and formatting capabilities to its users.
- **Part 7: File System** describes the PenPoint file system classes.
- **Part 8: System Services** describes the function calls that applications can use to access kernel functions, such as memory allocation, timer services, process control, and so on.
• Part 9: Utility Classes describes a wide variety of classes that save application writers from implementing fundamental things such as, list manipulation, data transfer, and so on.

• Part 10: Connectivity describes the classes that applications can use to access remote devices.

• Part 11: Resources describes the classes used to read, write, and create PenPoint resource files.

• Part 12: Installation API describes the PenPoint classes that support installing applications, services, fonts, dictionaries, handwriting prototypes, and so on.

• Part 13: Writing PenPoint Services, describes classes used in writing an installable service.

Other Sources of Information
As mentioned above, the PenPoint Application Writing Guide provides a tutorial on writing PenPoint applications. The tutorial is illustrated with several sample applications.

The PenPoint Development Tools describes how to run PenPoint on a PC, how to debug programs, and how to use a number of tools to enhance or debug your applications. This volume also contains a master index to the five volumes included in the PenPoint SDK.

The PenPoint Architectural Reference groups the PenPoint classes into several functional areas and describes how to use these classes. The PenPoint Architectural Reference is divided into two volumes. The first volume describes the fundamental classes that all application developers will use; the second volume describes supplemental classes that application developers may, or may not, use.

To learn how to use PenPoint, you should refer to the PenPoint user documentation. The user documentation is included with the PenPoint SDK, and is usually packaged with a PenPoint computer. The user documentation consists of these books:

• Getting Started with PenPoint, a primer on how to use PenPoint.

• Using PenPoint, a detailed book on how to use PenPoint to perform tasks and procedures.
**Type Styles in This Book**

To emphasize or distinguish particular words or text, we use different fonts.

**Computerese**

We use fonts to distinguish two different forms of “computerese”:

- C language keywords and preprocessor directives, such as `switch`, `case`, `#define`, `#ifdef`, and so on.
- Functions, macros, class names, message names, constants, variables, and structures defined by PenPoint, such as `msgListAddItem`, `clsList`, `stsBadParam`, `P_LIST_NEW`, and so on.

Although all these PenPoint terms use the same font, you should note that PenPoint has some fixed rules on the capitalization and spelling of messages, functions, constants, and types. By the spelling and capitalization, you can quickly identify the use of a PenPoint term.

- Classes begin with the letters “cls”; for example, `clsList`.
- Messages begin with the letters “msg”; for example, `msgNew`.
- Status values begin with the letters “sts”; for example, `stsOK`.
- Functions are mixed case with an initial upper case letter and trailing parentheses; for example, `OSMemAvailable()`.
- Constants are mixed case with an initial lower case letter; for example, `wsClipChildren`.
- Structures and types are all upper case (with underscores, when needed, to increase comprehension); for example, `U32` or `LIST_NEW_ONLY`.

**Placeholders**

Anything you do not have to type in exactly as printed is generally formatted in italics. This includes C variables, suggested filenames in dialogs, and pseudocode in file listings.

**Other Text**

The documentation uses italics for emphasis. When a Part uses a significant term, it is usually emphasized the first time. If you aren’t familiar with the term, you can look it up in the glossary in the *PenPoint Application Writing Guide* or the index of the book.

DOS filenames such as `\BOOT\PENPOINT\APP` are in small capitals. PenPoint file names can be upper and lower case, such as `\My Disk\Package Design Letter`.

Book names such as *PenPoint Application Writing Guide* are in italics.
### Part 12 / Installation API

- APPIMGR.H 511
- AUXNBMR.H 513
- CODEMGR.H 517
- DYNTABLE.H 525
- FONTMGR.H 529
- HWXMG.R.H 533
- INIFILE.H 537
- INSTALL.H 541
- INSTLMGR.H 543
- INSTLSHT.H 545
- PDICTMGR.H 563
- SERVIMGR.H 567
- SYSTEM.H 569

### Part 13 / Writing PenPoint Services

- HWXSERV.H 579
- MILSERV.H 581
- SERVCONEH 583
- SERVICE.H 589
- SERVMGR.H 593
- SERVMISC.H 609
- SVCTYPES.H 623

### Part 14 / Miscellaneous

- BATTERY.H 637
- DYNARRAY.H 639
- GOSEARCH.H 641
- PDICT.H 647
- POWER.H 649
- POWERMGR.H 653

### Index

- 655
Part 6 /
Text
This file contains the byte encodings used by clsText and clsTextView.

The byte encoding employed by the Text subsystem is based on the IBM-PC code page 850. However, there are differences as noted by the constants below; the differences are peculiar to Text’s interpretation of bytes, they are not part of the interpretation used by the Imaging subsystem. This byte encoding causes Text to use the font encoding sysDcEncodeHWX850 defined by sysfont.h.

In addition to the constants that define the byte encoding, classifications and routines that map from a byte to a class are defined, similar to those classification routines provided by crype.h. Use of these routines should be carefully isolated as they will be replaced by a different package in the "internationalized" version of PenPoint.

The functions described in this file are contained in TEXT.LIB.

```c
#ifndef TENCODE_INCLUDED
#define TENCODE_INCLUDED $Revision: 1.205$
#endif

ifndef GO_INCLUDED
#include <go.h>
#endif

Types and Constants

"Text encoding" abbreviates to "te".

Format effectors: recognized

#define teEmbeddedObject 0x13  // ASCII’s DC3, 850’s !!
#define teSpace 0x20
#define teTab 0x09
#define teNewLine 0x0D  // ASCII’s CR, 850’s music glyph
#define teNewPage 0x0C  // ASCII’s FF, 850’s female glyph
#define teNewParagraph 0x14  // ASCII’s DC4, 850’s para glyph
#define teUnrecognized 0x15  // ASCII’s NAK, 850’s sect glyph

Format effectors: unrecognized

#define teBackSpace 0x08
#define teLineFeed 0x0A
#define teVerticalTab 0x0B

The classification package is designed to support multiple classification schemes. The type TEXT_CHAR_TABLE represents the abstraction of a classification scheme; as such, a parameter of this type is required by each of the classification routines. TXTCTYPE_DEF represents the default classification scheme used by the Text subsystem. Thus, to see if a particular byte encodes a sentence ending character in the default classification scheme, the client would call:

TEIsSentenceEnd(TXTCTYPE_DEF, aByte)

typedef U16 TEXT_CTYPE_FLAG, *P_TEXT_CTYPE_FLAG;
typedef P_TEXT_CTYPE_FLAG TEXT_CHAR_TABLE;
#define TXTCTYPE_DEF ((TEXT_CHAR_TABLE)(-1L))
### Exported Functions and Macros

**TEIsSentenceEnd**
Determines if 'c' is a sentence-ending character.
Returns BOOLEAN.
BOOLEAN EXPORTED

Function Prototype
```c
TEIsSentenceEnd(
    TEXT_CHAR_TABLE table,
    CHAR c);
```

Comments
Returns true if and only if 'c' is a sentence-ending character.

**TEIsLineBreak**
Determines if 'c' forces a line-break.
Returns BOOLEAN.
BOOLEAN EXPORTED

Function Prototype
```c
TEIsLineBreak(
    TEXT_CHAR_TABLE table,
    CHAR c);
```

Comments
Returns true if and only if 'c' forces a line-break.

**TEIsBlank**
Determines if 'c' acts as a blank/space character.
Returns BOOLEAN.
BOOLEAN EXPORTED

Function Prototype
```c
TEIsBlank(
    TEXT_CHAR_TABLE table,
    CHAR c);
```

Comments
More than one character may act as a blank/space for some purposes. For example, a non-breaking blank/space; none is defined for the PenPoint Developers Release. Returns true if and only if 'c' acts as a blank/space character.

**TEIsSpecialPunct**
Determines if 'c' is a "special" punctuation character.
Returns BOOLEAN.
BOOLEAN EXPORTED

Function Prototype
```c
TEIsSpecialPunct(
    TEXT_CHAR_TABLE table,
    CHAR c);
```

Comments
Such characters end a word or sentence unless surrounded by alphanumerics. The period and commas in numbers are the most obvious case. Special punctuation might also include the periods in something like "Section II.A.i: The Rise and Fall of Punctuation". Since the surrounding context is not available to this function, it simply indicates whether the character can function as special punctuation; the caller must then examine the context to decide whether the character is actually special punctuation.

Returns true if and only if 'c' is a "special" punctuation character.
**TEIsWord**
Determines if 'c' is part of a "normal" word.

Returns BOOLEAN.

```c
BOOLEAN EXPORTED

Function Prototype
TEIsWord(
    TEXT_CHAR_TABLE table,
    CHAR c);
```

**Comments**
Returns true if and only if 'c' is part of a "normal" word.
This file contains clsTextView's well-known TAGs and associated constants.

The usage of well-known TAGS by clsTextView falls into these categories:

1) Quick Help identifiers
2) Option Sheet card and item (i.e., window) tags
3) Option Sheet card labels
4) User note identifiers

Most of clsTextView's Option Sheet components use the same tag for both the window tag and the quick help tag. This causes category 1 above to be almost identical to category 2.

All of the Quick Help resources for clsTextView can be enumerated by finding all resources whose .wkn.admin == resForQuickHelp (see qHelp.h) and Cls(.wkn.id) == cls(clsTextView).

```c
ifndef TV_TAGS_INCLUDED
#define TV_TAGS_INCLUDED

#include <go.h>

ifndef GO_INCLUDED
endif

#include <uid.h>

ifndef UID_INCLUDED
endif

// Allocated clsTextView TAGs: 1-54, 94-95
```

**Tags for Option Sheet**

```c
typedef enum TV_CARD_INDEX {     // TVMakeCardTag(TV_CARD_INDEX) => tag
tvCardChar = 0,
tvCardPara,
tvCardTabs,
tvCardView,
tvCardLength,       // Pseudo-card index which gives # cards
} TV_CARD_INDEX,

Labels for Option sheet & cards. All Card Label strings are in a single resource: a string array with ResId = tagTVOptResAdmin and indexed via TV_CARD_INDEX.

#define tagTVOptResAdmin MakeTag(clsTextView, 95)

typedef enum TV_CHAR_OPTION {     // TVMakeCharTag(TV_CHAR_OPTION) => tag
tvCharOptBold = 0,
tvCharOptFont,
tvCharOptItalic,
tvCharOptSize,
tvCharOptSizeOther,
tvCharOptSizeOtherVal,
tvCharOptSmallCaps,
tvCharOptStrike,
tvCharOptStyle,
tvCharOptUnderlineNormal,
tvCharOptUnderlineHeavy,
tvCharOptLength,       // Pseudo item which gives # char options
} TV_CHAR_OPTION;
```
```c
typedef enum TV_PARA_OPTION {
    tvParaOptAfterSpacing = 0,
    tvParaOptBeforeSpacing,
    tvParaOptFirstLineOffset,
    tvParaOptInterLineHeight,
    tvParaOptJustification,
    tvParaOptLeftMargin,
    tvParaOptLineHeight,
    tvParaOptRightMargin,
    tvParaOptLength // Pseudo item which gives # para options
} TV_PARA_OPTION;

typedef enum TV_VIEW_OPTION {
    tvViewOptSpecial = 0,
    tvViewOptMagnification,
    tvViewOptLength // Pseudo item which gives # show options
} TV_VIEW_OPTION;
```

The following macros combine all of the sub-ranges into a universal name space, suitable for both win.tag and gwin.helpld. Note that the labels of options are not tagged, only the value fields; if the labels must be tagged, use a new administered range so that it does not conflict with these helpIds.

```c
#define TVMakeTag(tag) MakeTag(clsTextView, (tag))
#define tagTextView TVMakeTag(1)
#define tagTextViewOption TVMakeTag(2)
#define TVMakeCardTag(i) TVMakeTag(3+i)
#define TVMakeCharOptTag(i) TVMakeTag(10+i)
#define TVMakeParaOptTag(i) TVMakeTag(30+i)
#define tagQHTabStop TVMakeTag(42)
#define TVMakeViewOptTag(i) TVMakeTag(45+i)
#define TVMakeXXXTag(i) TVMakeTag(55+i)
```

### Tags for Notes

A Note is a string displayed to the user when a Text View encounters difficulties processing a user action. All of the Note strings are in a single resource: a string array with ResId = resForStdMsgDialog(clsTextView) and indexed via the following ids.

```c
#define tagTVNoteResAdmin MakeTag(clsTextView, 94)
```

"text view note" abbreviates to "tvn".

```c
#define tvnHazardousSetting 1L // margins may overlap
#define tvnInvalidFieldValue 2L
#define tvnTranslateOutOfMem 3L
#define tvnTabsOverlap 4L
#define tvnReadOnlyChars 5L
#define tvnReadOnlyAttrs 6L
#define tvnNotAnIP 7L
#define tvnNotAComponent 8L
#define tvnApplyWithoutSel 9L // a negative number entered for an
#define tvnNegForUnsignedField 10L // unsigned field in an option sheet
#define tvnNewParasAdded 11L
```
This file contains the API definition for clsText.

clsText inherits from clsObject.

clsText is the Data Object for the Text subsystem. These objects hold characters, their attributes and embedded objects.

The functions described in this file are contained in TEXT.LIB.

Road Map

Clients manipulating the character contents of the textData might use:
- msgTextView
- msgViewTextBuffer
- msgViewTextModify

Clients manipulating the attributes stored in textData might use:
- msgTextViewChangeAttrs
- msgTextViewClearAttrs
- msgTextViewGetAttrs
- msgTextViewInitAttrs
- msgTextViewPrintAttrs
- TextViewCharAttrs() 
- TextViewCharMask() 
- TextViewParaAttrs() 
- TextViewParaMask() 
- TextViewDeleteMany() 
- TextViewInsertOne() 

Clients manipulating a textData’s embedded objects might use:
- msgTextEmbedObject
- msgTextExtractObject
- msgTextEnumEmbeddedObjects

Clients needing to work with words, sentences or paragraphs might use:
- msgTextSpan
- msgTextSpanType
Clients needing to import or export text might use:
• msgTextRead
• msgTextWrite

Clients observing a textData might want to handle:
• msgTextChanged
• msgTextReplaced

Characters and Encodings
Text data objects hold bytes representing characters using the encoding specified in tencode.h. In PenPoint 1.0, this encoding is derived from the IBM-PC's code page 850, and uses one byte per character. There are characters representing line, paragraph, and page breaks.

Characters are indexed starting from zero.

Formatting Information
Text data objects also hold "formatting" or "attribute" information. The types of attributes stored are:
• character attributes such as font face, size and weight
• paragraph attributes such margins, first line offset, first line offset
• tab attributes for a paragraph
• embedded object info (specifically the embedded object's uid)
• link termination (specifically the destination information for marks)

Attributes "tile" ranges of characters. In other words, no character can have two different sets of character attributes associated with it, although it can have both character and paragraph attributes. This tiling is enforced by the textData.

Any character that does not have explicit character or paragraph attributes takes on the "default" character or paragraph attributes of the data object. There are messages to inspect, enumerate, and modify all the attributes, including the defaults.

Relation to UI Classes
A textData only provides storage for characters and attributes. It does not provide any user interface (UI). The UI is provided by an instance of clsTextView.

To assist the class providing the UI, the textData provides notifications whenever either the characters or the attributes are modified.

Implementation Note
clsText is actually composed of three layers of classes. Clients need not be concerned by these layers, and should not rely on their existence as they may disappear in future releases.

clsTextBlock (usually referred to as clsText) is a descendant of clsTextMarkStore, which in turn is a descendant of clsTextChar.
Types and Constants: Atoms

Atoms are used as parameters to many of textData messages. All valid atoms are defined below.

typedef U16 ATOM;
#define atomChar ((ATOM) 1)
#define atomWord ((ATOM) 2)
#define atomLine ((ATOM) 3)
#define atomSentence ((ATOM) 4)
#define atomPara ((ATOM) 5)
#define atomDivision ((ATOM) 6)
#define atomDoc ((ATOM) 7)
#define atomMisc ((ATOM) 8)
#define atomEmbedded ((ATOM) 9)
#define atomParaTabs ((ATOM)10)
#define atomLink ((ATOM) 11)
#define atomWSDelimit ((ATOM)12)
#define atomClient1 ((ATOM)28)
#define atomClient2 ((ATOM) 29)
#define atomClient3 ((ATOM)30)
#define atomClient4 ((ATOM)31)
#define minValidAtom atomChar
#define maxValidAtom atomClient4

AtomGetName

Passes back a pointer to the string value of the atom.

Returns STATUS.

STATUS EXPORTED

Function Prototype
AtomGetName ( ATOM atom, PP_STRING ppString);

Comments
Most clients and subclasses do not use this function. It is occasionally useful for debugging.

Return Value
stsBadParam atom is out of the range of valid atoms
stsOK atom is within the valid range. *ppString may still be NULL if the atom falls into one of the gaps.
Types and Constants: Character Indices

Character Indices

typedef U32 TEXT_INDEX;
typedef TEXT_INDEX * P_TEXT_INDEX;
#define maxTEXT_INDEX maxU32;

Some messages and functions which take a TEXT_INDEX as a parameter may use special values to achieve certain effects. Each message and function description indicates which special values can be used.

#define lpoTEXT_INDEX (maxTEXT_INDEX-maxU16)
#define lastTEXT_INDEX (lpoTEXT_INDEX-1)
#define infTEXT_INDEX (maxTEXT_INDEX-1)
#define mInfTEXT_INDEX maxTEXT_INDEX

"Magic" value for msgTextChangeAttrs, msgTextGetAttrs and msgTextInitAttrs.

#define textDefaultAttrs infTEXT_INDEX

Types and Constants: Character Attributes

The prefixes "TA_" and "ta" indicate that an identifier is related to "text attributes."

Use these in the alignBase field of a TA_CHAR_ATTRS.

typedef enum {
    taNormalLineBase = 0,
    } TA_ALIGN_BASE;

Character Attributes

typedef struct TA_CHAR_ATTRS {
    U16 size;            // Font size in twips. Not all values are available -- some are rounded down. Max of 160*20 twips.
    U16 tacSpare : 8,    // Reserved.
    highlight : 1,
    smallCaps : 1,
    upperCase : 1,
    strikeout : 1,
    underlines : 2,     // As defined in syslog.h. Must be 0, 1, or 2.
    alignBase : 2;      // Use a TA_ALIGN_BASE value. Only taNormalLineBase is implemented.

    SYSDC_FONT_SPEC font;
} TA_CHAR_ATTRS, *P_TA_CHAR_ATTRS;

Character Attributes Mask.

The highlight and encoding fields contain extra bits. These bits are automatically zero-ed by assigning a legitimate values to the field.

typedef struct {
    U16 tacSpare : 8, // Must fit in 32 bits
    highlight : 2,   // true or false (and 1 spare bit)
    size : 1,
    smallCaps : 1,
    upperCase : 1,
    strikeout : 1,
    underlines : 1,
    alignBase : 1;
    U16 id : 1,      // mask bit for attrs.font.id
    group : 1,       // mask bit for attrs.font.attr.group
Types and Constants: Tab Attributes

Each paragraph can have up to TA_MAX_TABS tab stops. A paragraph without its own explicit tab stops "inherits" the document's "default" tab stops.

Paragraphs that desire uniformly spaced tab stops can compactly define the stops by setting at least two explicit stops and then setting repeatAtEnd to true. This has the effect of defining an unlimited number of implicit stops, each of which follows the prior stop by the distance between the last two explicit stops.

NOTE: Even though each tab store has a type and leader, only the type taTabLeft and the leader taLeadSpace are implemented.

typedef enum {                               // Must fit in 2 bits
    taTabLeft = 0,                          // Not Implemented
    taTabCenter = 1,                        // Not Implemented
    taTabRight = 2,                         // Not Implemented
    taTabDecimal = 3                        // Not Implemented
} TA_TAB_TYPE;

typedef enum {                               // Must fit in 2 bits
    taLeadSpace = 0,                        // Not Implemented
    taLeadDot = 1,                          // Not Implemented
    taLeadDash = 2,                         // Not Implemented
    taLeadUnderline = 3                     // Not Implemented
} TA_TAB_LEADER;

Tab Stop.

The type and leader fields contain extra bits. These bits are automatically zero-ed by assigning a legitimate values to the field.

typedef struct TA_TAB_STOP {               // In twips
    U16 x;                                  // TA_TAB_TYPE (and 6 spare bits)
    U8 type;                                // TA_TAB_LEADER (and 6 spare bits)
    U8 leader;
} TA_TAB_STOP, *P_TA_TAB_STOP;

The maximum number of tab stops for a paragraph.

#define TA_MAX_TABS 31

Tab Stops.

The count and repeatAtEnd fields contain extra bits. These bits are automatically zero-ed by assigning a legitimate values to the field.

typedef struct TA_TABS {                   // Number of tab stops, in the range
    U16 count : 8,                         // 0..TA_MAX_TABS. (plus 3
                                            // spare bits.)
    repeatAtEnd : 8;                      // true or false (and 7 spare bits)
    TA_TAB_STOP tabs[1];                  // Actually variable size array
} TA_TABS, *P_TA_TABS;
Another representation of tab stops.

typedef struct TA_MANY_TABS {
    U16 count : 8,  // Number of tab stops, in the range 0..TA_MAX_TABS. (plus 3
                     // spare bits.)
    repeatAtEnd : 8; // true or false (and 7 spare bits)
    TA_TAB_STOP tabs[TA_MAX_TABS];
} TA_MANY_TABS, *P_TA_MANY_TABS;

#define textNoTabs ((P_TA_MANY_TABS)1)  // Not Implemented

## Types and Constants: Paragraph Attributes

Use these in the alignment field of a TA_PARA_ATTRS.

typedef enum {
    taParaLeft = 0,
    taParaCenter = 1,
    taParaRight = 2,
    taParaSpare = 3  // Reserved
} TA_PARA_ALIGN;

Paragraph Attributes.

All of the fields in TA_PARA_ATTRS that are linear measurements are in twips.

The alignment and justify fields contain extra bits. These bits are automatically zero-ed by assigning a legitimate values to the field.

typedef struct TA_PARA_ATTRS {
    U16 alignment : 8,  // TA_PARA_ALIGN (and 6 spare bits)
        justify : 8;  // 0 or 1. (0x80 is used internally,
                        // so there are 6 spare bits.)
    U16 lineHeight;  // The special value textUseMaxHeightOnLine
                     // causes the line height to be as high
                     // as the highest thing in the line.
                     // Don't use zero!
    U16 interLineHeight;
    U16 beforeSpacing;  // Adds to previous paragraphs's
                        // afterSpacing
    S16 afterSpacing;
    S16 firstLineOffset;  // Add to leftMargin to get the effective
                           // left margin for the first line of the
                           // paragraph.
    U16 leftMargin;
    U16 rightMargin;
} TA_PARA_ATTRS, *P_TA_PARA_ATTRS;

Special lineHeight value

#define textUseMaxHeightOnLine maxU16  // 0 or 1 (2 spare bits)

Paragraph Attribute Mask

The lineHeight, interLineHeight, beforeSpacing and afterSpacing fields contain extra bits. These bits are automatically zero-ed by assigning a legitimate values to the field.

typedef struct {
    U16 alignment : 1,  // Must fit in 32 bits
        justify : 1,
        firstLineOffset : 1,
        leftMargin : 1,
        rightMargin : 1,
        lineHeight : 3,  // 0 or 1 (2 spare bits)
        interLineHeight : 8;  // 0 or 1 (7 spare bits)
    U16 beforeSpacing : 8;  // 0 or 1 (7 spare bits)
    afterSpacing : 8;  // 0 or 1 (7 spare bits)
} TA_PARA_MASK, *P_TA_PARA_MASK;
Types and Constants: Embedding

typedef struct TEXT_EMBED_OBJECT {
    TEXT_INDEX first;
    OBJECT toEmbed;
    U8 clientFlags;
    U8 action;  // One of the values below (6 spare bits)
} TEXT_EMBED_OBJECT, *P_TEXT_EMBED_OBJECT;

Use these in the action field of a TEXT_EMBED_OBJECT.

#define textEmbedCopy 0  // For internal use only.
#define textEmbedFree 1  // For internal use only.
#define textEmbedInsert 2
#define textEmbedMove 3  // For internal use only.

The fields of this structure are described in the comments for msgTextEnumEmbeddedObjects.

typedef struct TEXT_ENUM_EMBEDDED {
    TEXT_INDEX first;
    TEXT_INDEX length;
    U16 flags;  // One of the values below
    U16 max;
    U16 count;
    P_TEXT_EMBED_OBJECT pItems;
} TEXT_ENUM_EMBEDDED, *P_TEXT_ENUM_EMBEDDED;

The prefix "tee" indicates that an identifier is related to "TEXT_ENUM_EMBEDDED."

Use these in the flags field of a TEXT_ENUM_EMBEDDED.

#define teeFloat flag0  // Include floating embedded objects. (These will be children of theRootWindow.)
#define teeInline flag1  // Include embedded objects
#define teeDefault (teeFloat | teeInline)

Types and Constants: Import/Export

More information about the fields of this structure is in the comments for for msgTextRead.

The freeAfter and inputIsObject fields contain extra bits. These bits are automatically zero-ed by assigning a legitimate values to the field.

typedef struct TEXT_READ {
    TEXT_INDEX first;
    P_UNKNOWN input;
    U16 embeddedAction: 2,
        freeAfter: 6,  // true or false (and 5 spare bits)
        inputIsObject: 8;  // true or false (and 7 spare bits)
    TAG format;
} TEXT_READ, *P_TEXT_READ;

More information about the fields of this structure is in the comments for for msgTextWrite.

The flags and outputIsObject fields contain extra bits. These bits are automatically zero-ed by assigning legitimate values to the fields.

typedef struct TEXT_WRITE {
    TEXT_INDEX first;
    TEXT_INDEX length;
    P_UNKNOWN output;
    U16 flags;  // One of the values below (and 13
        // spare bits)
    TAG format;
    U8 outputIsObject;
} TEXT_WRITE, *P_TEXT_WRITE;
The prefix "tw" indicates that an identifier is related to "text write."

Use these in the flags field of a TEXT_WRITE. They are described in the comments for msgTextWrite.

#define twExtractEmbedded flag0
#define twTempFile flag1
#define twForUndo flag3

### Other Types and Constants

typedef OBJECT TEXT_DATA;

Resource ids

#define textResDefaultCharAttrs MakeWknResId(clsText, 1)
#define textResDefaultParaAttrs MakeWknResId(clsText, 2) // Not Impl.
#define textResDefaultParaTabs MakeWknResId(clsText, 3) // Not Impl.

### Public Functions and Macros

#### Utility Functions

**TextDeleteMany**

Deletes characters from a textData.

Returns STATUS.

**Function Prototype**

```
TextDeleteMany(
    const OBJECT dataObj,
    const TEXT_INDEX pos,
    const TEXT_INDEX length);
```

**Comments**

The return values are the same as those for msgTextModify.

**TextInsertOne**

Inserts one character into a textData.

Returns STATUS.

**Function Prototype**

```
TextInsertOne(
    const OBJECT dataObj,
    const TEXT_INDEX pos,
    const CHAR toInsert);
```

**Comments**

The return values are the same as those for msgTextModify.

**TextFindNextParaTab**

Passes back the next tab stop to the right of the passed-in stop.

Returns STATUS.

**Function Prototype**

```
TextFindNextParaTab(
    const P_TA_TABS p,
    const P_TA_TAB_STOP pTab,
    const P_U16 pIndex);
```
Note that if p->repeatAtEnd is true, there are effectively an infinite number of tab stops.

stsNoMatch - no tabs, or this is the last tab.

**Attribute and Mask Initialization Routines**

**TextlnitCharAttrs**
Initializes a character attribute structure.
Returns nothing.

```c
void EXPORTED TextlnitCharAttrs( P_TA_CHAR_ATTRS p);
```

Comments
This function reads the default character attributes from the process's resource list (using the resource id textResDefaultCharAttrs), or sets all values to 0 if the resource cannot be found.

See Also
msgTextChangeAttrs

**TextlnitCharMask**
Initializes a character attribute mask to all zeros.
Returns nothing.

```c
void EXPORTED TextlnitCharMask( P_TA_CHAR_MASK p);
```

See Also
msgTextChangeAttrs

**TextlnitParaAttrs**
Initializes a paragraph attribute structure to all zeros.
Returns nothing.

```c
void EXPORTED TextlnitParaAttrs( P_TA_PARA_ATTRS p);
```

See Also
msgTextChangeAttrs

**TextlnitParaMask**
Initializes a paragraph attribute mask to all zeros.
Returns nothing.

```c
void EXPORTED TextlnitParaMask( P_TA_PARA_MASK p);
```

See Also
msgTextChangeAttrs
Message Arguments

The prefix "TD_" indicates that an identifier is related to "text data."

The prefix "tdm" indicates that an identifier is related to "text data metrics."

typedef struct TD_METRICS {
    U16 flags; // One of the values below
    U16 spareBits; // Reserved.
    P_UNKNOWN spares[2]; // Reserved.
} TD_METRICS, *P_TD_METRICS;

Use these in the flags field of a TD_METRICS.

#define tdmCanUndo     flag8 // if on, textData supports undo
#define tdmFileCharsOnOwn flag1 // Not Implemented
#define tdmReadOnly    flag0  // characters cannot be modified

expectedSize is a hint about the expected number of characters in a textData. An accurate hint can improve performance.

typedef struct TD_NEW_ONLY {
    TD_METRICS metrics;
    TEXT_INDEX expectedSize;
    U16 expectedTagCount; // Private. For internal use only.
} TD_NEW_ONLY, *P_TD_NEW_ONLY;

typedef struct TD_NEW {
    OBJECT_NEW_ONLY object;
    TD_NEW_ONLY text;
} TD_NEW, *P_TD_NEW;

typedef struct TEXT_BUFFER {
    TEXT_INDEX first; // In
    TEXT_INDEX length; // In
    TEXT_INDEX bufLen; // In
    P_CHAR buf; // In:Out via *buf
    TEXT_INDEX bufUsed; // Out
} TEXT_BUFFER, *P_TEXT_BUFFER;

typedef enum {
    tdForward = 1,
    tdBackward = 2
} TEXT_DIRECTION;

typedef struct TEXT_SPAN {
    TEXT_INDEX first; // In:Out
    TEXT_INDEX length; // In:Out
    ATOM type; // In:Out (for msgTextSpanType)
    TEXT_DIRECTION direction; // In
    BOOLEAN needPrefix; // In
    BOOLEAN needSuffix; // In
    U16 prefixLength; // Out: valid if and only if needPrefix is true
    U16 suffixLength; // Out: valid if and only if needSuffix is true
    U8 firstNormal; // Out: 0 or 1 (7 spare bits)
    U8 lastNormal; // Out: 0 or 1 (7 spare bits)
    U32 spares[4]; // Reserved
} TEXT_SPAN, *P_TEXT_SPAN;

typedef struct TEXT_SPAN_AFFECTED {
    OBJECT sender;
    U32 changeCount;
    TEXT_INDEX first;
    TEXT_INDEX length;
} TEXT_SPAN_AFFECTED, *P_TEXT_SPAN_AFFECTED;
typedef struct TEXT_REPLACED {
    TEXT_SPAN_AFFECTED span;
    TEXT_INDEX bytesTakenFromBuf;
} TEXT_REPLACED, *P_TEXT_REPLACED;

typedef struct TEXT_AFFECTED {
    TEXT_SPAN_AFFECTED span;
    U16 remeasurei;
    P.UNKNOWN spare;
} TEXT_AFFECTED, *P_TEXT_AFFECTED;

typedef struct TEXT_COUNTER_CHANGED {
    OBJECT sender;
    U32 changeCount;
    U32 oldCount;
} TEXT_COUNTER_CHANGED, *P_TEXT_COUNTER_CHANGED;

typedef struct TEXT_CHANGE_ATTRS {
    ATOM tag;
    TEXT_INDEX first;
    TEXT_INDEX length;
    P.UNKNOWN pNewMask;
    P.UNKNOWN pNewValues;
} TEXT_CHANGE_ATTRS, *P_TEXT_CHANGE_ATTRS;

typedef struct TEXT_GET_ATTRS {
    ATOM tag;
    TEXT_INDEX first;
    TEXT_INDEX length;
    P.UNKNOWN pValues;
} TEXT_GET_ATTRS, *P_TEXT_GET_ATTRS;

Messages Defined by Other Classes

msgNewDefaults
Initializes the NEW struct.
Takes P_TD_NEW, returns STATUS. Category: class message.

Message Arguments
typedef struct TD_NEW {
    OBJECT_NEW ONLY object;
    TD_NEW ONLY text;
} TD_NEW, *P_TD_NEW;

Comments
In response to this message, dsText does the following:

pNew->object.cap = objCapCreate;
memset(&(pNew->text), 0, sizeof(pNew->text));
pNew->text.expectedSize = 5;
pNew->text.expectedTagCount = 5;

msgNew
Creates a new instance of dsText.
Takes P_TD_NEW, returns STATUS. Category: class message.

Message Arguments
typedef struct TD_NEW {
    OBJECT_NEW ONLY object;
    TD_NEW ONLY text;
} TD_NEW, *P_TD_NEW;
**msgTextChangeCount**

Passes back (and optionally sets) the `textData`'s `changeCount`.

Takes S32, returns S32.

```c
#define msgTextChangeCount TCMakeMsg(0)
```

**Comments**

Each instance of `dsText` keeps a monotonically increasing count of the number of changes that have been made to it (via `msgTextModify`). In response to this message, a `textData` passes back that count. The counter's value is always greater than or equal to 0.

If the value of `pArgs` is:

- `< 0` the counter's current value is returned and the counter is unchanged.
- `maxS32` the counter is incremented by one, and the new value returned.
- `>= 0` the counter is set to `pArgs`, and its previous value is returned.

In general, clients should only increment the counter, not decrement it.

**msgTextGet**

Returns the character in a `textData` at the specified position.

Takes `TEXT_INDEX`, returns STATUS.

```c
#define msgTextGet TCMakeMsg(1)
```

**Return Value**

- `stsEndOfData` `pArgs->first` is too large
- `>= 0` the 8 bit character is returned as the low byte of the returned STATUS; the high 3 bytes are zero.

**msgTextGetBuffer**

Passes back a contiguous range of characters from a `textData`.

Takes `P_TEXT_BUFFER`, returns STATUS.

```c
#define msgTextGetBuffer TCMakeMsg(5)
```

**Message Arguments**

```c
typedef struct TEXT_BUFFER {
    TEXT_INDEX first; // In
    TEXT_INDEX length; // In
    TEXT_INDEX,bufLen; // In
    P_CHAR buf; // In:Out via *buf
    TEXT_INDEX bufUsed; // Out
} TEXT_BUFFER, *P_TEXT_BUFFER;
```

**Comments**

Use this message to get the values of several characters at a time. This message is a high-performance alternative to `msgTextGet`.

If `pArgs->length` > `pArgs->bufLen`, then up to `bufLen` characters are placed into `pArgs->buf`.

Upon return, `pArgs->bufUsed` is set to the count of characters read, even if there was a problem with the request.

**Return Value**

- `stsBadParam` `pArgs->length` was 0 or `pArgs->bufLen` was 0 or `pArgs->buf` was `pNull`
- `stsEndOfData` `pArgs->first` is too large
- `< stsOK` some other error occurred.
msgTextGetMetrics

Passes back the textData's metrics.

Takes P_TD_METRICS, returns STATUS.

```c
#define msgTextGetMetrics TCMakeMsg(2)

typedef struct TD_METRICS {
    U16 flags;
    U16 spareBits;
    P_UNKNOWN spares[2];
} TD_METRICS, *P_TD_METRICS;
```

msgTextLength

Returns the number of characters stored in the textData.

Takes nothing, returns TEXT_INDEX.

```c
#define msgTextLength TCMakeMsg(3)

< stsOK some error occurred.

>= stsOK Cast the returned value to a TEXT_INDEX; that's the number of characters.
```

msgTextModify

Modifies the characters stored in the textData.

Takes P_TEXT_BUFFER, returns STATUS.

```c
#define msgTextModify TCMakeMsg(4)

typedef struct TEXT_BUFFER {
    TEXT_INDEX first;
    TEXT_INDEX length;
    TEXT_INDEX bufLen;
    P_CHAR buf;
    TEXT_INDEX bufUsed;
} TEXT_BUFFER, *P_TEXT_BUFFER;
```

Use this message to insert, delete or replace characters in a textData.

In response to this message, the textData replaces the characters in the range [pArgs->first .. pArgs->first+pArgs->length) with the characters from pArgs->buf. If pArgs->buf is pNull, the effect is a deletion. If pArgs->length is 0, the effect is an insertion. Otherwise the effect is a replacement. If pArgs->first is in TEXT_INDEX, the current length minus pArgs->length is substituted. If pArgs->length is max TEXT_INDEX, strlen(pArgs->buf) is substituted.

```c
stsReadOnly request refused because object is read only.

stsOK modification successful.
```

msgTextSetMetrics

Sets a textData's metrics.

Takes P_TD_METRICS, returns STATUS.

```c
#define msgTextSetMetrics TCMakeMsg(6)

typedef struct TD_METRICS {
    U16 flags;
    U16 spareBits;
    P_UNKNOWN spares[2];
} TD_METRICS, *P_TD_METRICS;
```
**msgTextSpan**

Determines the range corresponding to the requested span.

Takes P_TEXT_SPAN, returns STATUS..

```c
#define msgTextSpan TCMakeMsg(9)
```

```c
typedef struct TEXT_SPAN {
    TEXT_INDEX first; // In:Out
    TEXT_INDEX length; // In:Out
    ATOM type; // In:Out (for msgTextSpanType)
    TEXT_DIRECTION direction; // In
    BOOLEAN needPrefix; // In
    BOOLEAN needSuffix; // In
    U16 prefixLength; // Out: valid if and only if needPrefix is true
    U16 suffixLength; // Out: valid if and only if needSuffix is true
    U8 firstNormal; // Out: 0 or 1 (7 spare bits)
    U8 lastNormal; // Out: 0 or 1 (7 spare bits)
    U32 spares[4]; // Reserved
} TEXT_SPAN, *P_TEXT_SPAN;
```

**Comments**

A span is a consecutive range of characters that share some common trait. Given a position and the desired span type, this message returns the range of the span. For instance, a client can use this message to ask a textData to find the bounds of the word containing a position.

Actually, this message can be used to find the start of one span and the end of another. If pArgs->length is 1, then the start and end of the same span is returned.

If the client only needs only the beginning or the end of the span, then pArgs->direction should be set to the needed end. This substantially improves performance.

Using this message, a textData can find the range of the following types of spans:

- **atomWSDelimit**: passes back a white-space delimited span
- **atomWord**: passes back a word span using the definitions in tencode.h

pArgs->type specifies the desired span's type.

pArgs->direction indicates whether the span should be searched for in preceding characters, succeeding characters, or both.

It is often useful to know something about the characters immediately preceding or succeeding the span. This information is returned if pArgs->needPrefix or pArgs->needSuffix (or both) are true. Upon return, pArgs->prefixLength and/or pArgs->suffixLength identifies the appropriate characters.

pArgs->firstNormal and pArgs->lastNormal indicate whether the corresponding portions of the span are normal or abnormal characters for the span. For instance, for **atomWord**, an "a" is a normal character, but an "!'" is abnormal.

**Return Value**

- **stsBadParam**: Neither the two directions in pArgs->direction was on.
**msgTextSpanType**

Determines the span type of the specified range.

Takes **P_TEXT_SPAN**, returns **STATUS**.

```c
#define msgTextSpanType TCMakeMsg(10)
```

```c
typedef struct TEXT_SPAN {
    TEXT_INDEX first;       // In:Out
    TEXT_INDEX length;      // In:Out
    ATOM type;              // In:Out (for msgTextSpanType)
    TEXT_DIRECTION direction; // In
    BOOLEAN needPrefix;     // In
    BOOLEAN needSuffix;     // In
    U16 prefixLength;       // Out: valid if and only if needPrefix is true
    U16 suffixLength;       // Out: valid if and only if needSuffix is true
    U8 firstNormal;         // Out: 0 or 1 (7 spare bits)
    U8 lastNormal;          // Out: 0 or 1 (7 spare bits)
    U32 spares[4];          // Reserved
} TEXT_SPAN, *P_TEXT_SPAN;
```

**Comments**

In response to this message, a **textData** passes back the span type that corresponds to the range.

The same range often has several span types. For instance, all ranges have the span type **atomChar**. All ranges that include a complete paragraph also have the span types **atomChar**, **atomWord** and **atomSentence**. When the passed-in range has multiple span types, the largest span type is returned.

The span type ordering from smallest to largest is as follows. This is also the complete list of span types returned in response to this message.

- **atomChar**
- **atomWord**
- **atomSentence**
- **atomPara**
- **atomDoc**

**msgTextChangeAttrs**

Changes the attributes of the specified range.

Takes **P_TEXT_CHANGE_ATTRS**, returns **STATUS**.

```c
#define msgTextChangeAttrs TAMakeMsg(taVersion, 1)
```

```c
typedef struct TEXT_CHANGE_ATTRS {
    ATOM tag;
    TEXT_INDEX first;
    TEXT_INDEX length;
    P_UNKNOWN pNewMask;
    P_UNKNOWN pNewValues;
} TEXT_CHANGE_ATTRS, *P_TEXT_CHANGE_ATTRS;
```

**Comments**

Clients use this message to change the formatting attributes of characters in a **textData**. They can manipulate three types of attributes:

- character attributes (indicated by **atomChar**)
- paragraph attributes (indicated by **atomPara**)
• tab attributes (indicated by atomParaTabs)

The pArgs type for this message is P_TEXT_CHANGE_ATTRS. This structure has a tag, which must be one of the three atoms mentioned above. The structure also has two P_UNKNOWN fields: pNewMask and pNewValues. The true type of these two fields depends on the value of the tag.

<table>
<thead>
<tr>
<th>tag</th>
<th>pNewValues type</th>
<th>pNewMask type</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomChar</td>
<td>P_TA_CHAR_ATTRS</td>
<td>P_TA_CHAR_MASK</td>
</tr>
<tr>
<td>atomPara</td>
<td>P_TA_PARA_ATTRS</td>
<td>P_TA_PARA_MASK</td>
</tr>
<tr>
<td>atomParaTabs</td>
<td>P_TA_MANY_TABS</td>
<td>none; always null</td>
</tr>
</tbody>
</table>

The mask field allows the client to change only some of the attributes. If the appropriate bit in the mask is off, then the value of the attribute is not changed. To simplify initializing attribute and mask structures, textData has a few utility messages and functions:

msgTextInitAttrs The client must set the tag pArgs->first. In response to this message, a textData initializes pNewValues to the values in effect at pArgs->first and sets all of the bits in the mask to zero.

TextInitCharAttrs reads the default character attributes from the process's resource list (using the resource id textResDefaultCharAttrs), or sets all values to 0 if the resource cannot be found.

TextInitParaAttrs Sets all values to 0.

TextInitParaMask Turns off all bits in the mask

If pArgs->first is the "magic value" textDefaultAttrs, the textData's default attributes are modified.

If pArgs->tag is atomPara or atomParaTabs, then the passed-in range is automatically extended to complete paragraph boundaries. (The resulting range is passed back in pArgs->first and pArgs->length updated.)

Return Value

stsBadParam Either pArgs->tag or the range was invalid. No attributes have changed.

< stsOK Some other error occurred. No attributes have changed.

msgTextClearAttrs

Clears all attributes of the specified type to the default values.

Takes ATOM, returns STATUS.

#define msgTextClearAttrs TBMakeMsg(5)

Comments

In response to this message, a textData clears all formatting for the specified type. This message is "all or nothing" -- no mask or range can be specified.

The attributes have not changed the return value is < stsOK:

Return Value

stsBadParam pArgs was invalid. No attributes have changed.

< stsOK Some other error occurred. No attributes have changed.

msgTextEmbedObject

Embeds an object at a specified position.

Takes P_TEXT_EMBED_OBJECT, returns STATUS.

#define msgTextEmbedObject TBMakeMsg(2)
typedef struct TEXT_EMBED_OBJECT {
    TEXT_INDEX first;
    OBJECT toEmbed;
    U8 clientFlags;
    U8 action; // One of the values below (6 spare bits)
} TEXT_EMBED_OBJECT, *P_TEXT_EMBED_OBJECT;

Each embedded object is represented by a character with the encoding value teEmbeddedObject. (See tencode.h.)

In response to this message, the textData inserts the embedded object anchor character and "remembers" the embedded object's id.

msgTextExtractObject

Extracts the specified embedded object.
Takes OBJECT, returns STATUS.
#define msgTextExtractObject TBMakeMsg(4)

In response to this message, the textData "forgets" the specified embedded object. It also deletes the associated embedded object anchor character.

Nothing is done to the object itself. In particular, the client should probably msgWinExtract the object.

msgTextGetAttrs

Gets the attributes of the specified type.
Takes P_TEXT_GET_ATTRS, returns STATUS.
#define msgTextGetAttrs TAMakeMsg(taVersion, 2)

Clients can retrieve the attributes of a character in the textData using msgTextGetAttrs.

The client specifies the type of attributes it is interested in by filling in pArgs->tag. The client must set pArgs->pValues to point to a structure with the "real" type of the attributes corresponding to the tag. This "real" type is described in the comments for msgTextChangeAttrs.

The client also specifies the character whose attributes the client wants by specifying pArgs->first. If pArgs->first is textDefaultAttrs then the default attribute values are returned.

Return Value
stsBadParam  pArgs->tag is not valid
stsEndOfData  pArgs->first is too large
stsOK        the attribute values have been copied into pArgs->pValues

msgTextInitAttrs

Initialize the attributes and mask before a msgTextChangeAttrs.
Takes P_TEXT_CHANGE_ATTRS, returns STATUS.
#define msgTextInitAttrs TAMakeMsg(taVersion 3)
typedef struct TEXT_CHANGE_ATTRS {
    ATOM tag;
    TEXT_INDEX first;
    TEXT_INDEX length;
    P_UNKNOWN pNewMask;
    P_UNKNOWN pNewValues;
} TEXT_CHANGE_ATTRS, *P_TEXT_CHANGE_ATTRS;

The type of attributes is specified by pArgs->tag. pArgs->pNewValues and pArgs->pNewMask must be set as appropriate to an invocation of msgTextChangeAttrs.

If pArgs->first is textDefaultAttrs, the default attributes are used to initialize pArgs->pNewValues. Otherwise the attributes in effect at pArgs->first are used. All bits of pArgs->pNewMask are set to 0.

stsBadParam Either pArgs->tag or the range was invalid.
< stsOK Some other error occurred. No change has been made to the attributes and mask.

See Also
msgTextChangeAttrs

msgTextPrintAttrs
Prints the values of an attribute set and a mask.
Takes P_TEXT_CHANGE_ATTRS, returns stsOK.

#define msgTextPrintAttrs TAMakeMsg(taVersion, 4)

typedef struct TEXT_CHANGE_ATTRS {
    ATOM tag;
    TEXT_INDEX first;
    TEXT_INDEX length;
    P_UNKNOWN pNewMask;
    P_UNKNOWN pNewValues;
} TEXT_CHANGE_ATTRS, *P_TEXT_CHANGE_ATTRS;

This message takes the same parameters as msgTextChangeAttrs and the pArgs must be filled in the same way. In response to this message, a textData prints out a useful dump of the contents of pArgs.

Internal Use Only: If pArgs->first is txtPrvAttrs, then pArgs->pNewValues must be in the internal format.

See Also
msgTextChangeAttrs

msgTextRead
Inserts Ascii, RTF, etc. at the specified location.
Takes P_TEXT_READ, returns STATUS.
#define msgTextRead TAMakeMsg(0)

typedef struct TEXT_READ {
    TEXT_INDEX first;
    P_UNKNOWN input;
    UI6 embeddedAction: 2,
    freeAfter: 6, // true or false (and 5 spare bits)
    inputIsObject: 8; // true or false (and 7 spare bits)
    TAG format;
} TEXT_READ, *P_TEXT_READ;

The textData reads data and inserts the data into itself.
The fields of pArgs are:

- **first**: the read text is inserted into the textData starting at this position. After a successful return, pArgs->first is position immediately after the inserted text.

- **input**: the input source. If pArgs->inputIsObject is true, this field must hold a FILE_HANDLE object. If pArgs->inputIsObject is false, then this field must hold a P_FILE.

- **embeddedAction**: Client must set this to TEXTEmbedInsert. (Other values are for internal use only.)

- **freeAfter**: If true, then pArgs->input is freed after reading successfully.

- **inputIsObject**: describes the type of pArgs->input.

- **format**: one of the file types defined in filetype.h, or fileTypeUndefined. If the latter, the textData object attempts to deduce the form at from the contents of the data found in pArgs->input.

The textData reads pArgs->input using the functions defined in stdio.h. Thus, if pArgs->inputIsObject is true, pArgs->input must be an object which supports the stream protocol as used by stdio.

- **stsReadOnly**: request refused because object is read only.

- **stsNoMatch**: RTF error: first character of input is not "{|" or format version > 1 or unrecognized font name.

- **stsFailed**: StdioStreamBind() or fseek() failed.

- **stsBadParam**: pArgs->format is invalid.

- **stsFS...**: see <fs.h>.

- **stsOK**: request completed successfully; pArgs->first updated.

### msgTextWrite

Outputs the specified span as one of Ascii, RTF, etc.

Takes P_TEXT_WRITE, returns STATUS.

```c
#define msgTextWrite TBMakeMsg(1)
```

```c
typedef struct TEXT_WRITE
{
  TEXT_INDEX first;
  TEXT_INDEX length;
  P_UNKNOWN output;
  U16 flags; // One of the values below (and 13 spare bits)
  TAG format;
  U8 outputIsObject;
} TEXT_WRITE, *P_TEXT_WRITE;
```

The fields of pArgs are:

- **first**: first character of range to be written

- **length**: length of range to be written

- **output**: if null, the textData creates a P_FILE and returns that handle. If non-null, then this field is either an object or a P_FILE, depending on the value of outputIsObject.

- **flags**: described below

- **format**: one of the file types defined in filetype.h.

- **outputIsObject**: If output is non-null and outputIsObject is true, then output is an object. If output is non-null and outputIsObject is false, then output is a P_FILE.
Possible values for the flags field of a TEXT_WRITE are:

twExtractEmbedded embedded objects in the specified span are extracted from their parent window.

twTempFile if output is null, then a temporary file is created. (Developer's Note: If you're debugging the behavior of msgTextWrite, you probably don't want to turn this flag on as your file will be deleted before msgTextWrite returns.)

twForUndo add additional information needed for supporting UNDO.

**Return Value**

stsBadParam pArgs->format is invalid.

stsFailed StdioStreamBind() failed.

stsFS... see <fs.h>.

stsOK request completed successfully.

---

**msgTextEnumEmbeddedObjects**

Enumerates the textData's embedded objects.

Takes P_TEXT_ENUM_EMBEDDED, returns STATUS.

```c
#define msgTextEnumEmbeddedObjects TMMakeMsg(9)
```

**Message Arguments**

```c
typedef struct TEXT_ENUM_EMBEDDED {
    TEXT_INDEX first;
    TEXT_INDEX length;
    U16 flags; /* One of the values below
    U16 max;
    U16 count;
    P_TEXT_EMBED_OBJECT pItems;
} TEXT_ENUM_EMBEDDED, *P_TEXT_ENUM_EMBEDDED;
```

**Comments**

There are two ways of enumerating the embedded objects:

1) Get all the objects in one send. The textData allocates an array of TEXT_EMBED_OBJECT elements and passes it back in pArgs->pItems. You must OSHapBlockFree() the array when you are done with it.

TEXT_ENUM_EMBEDDED is used as follows:

**first** position at which you want to start the enumeration. Use 0 to start at the beginning of the data.

**length** length of the range you want the enumeration to include. Use infTEXT_INDEX to go to the end of the data.

**flags** Usually teeDefault. Use teeFloat to get only floating embedded objects. Use teeInline to get only in-line embedded objects.

**max** Pass in 0. The object passes back the number of items in the allocated block

**count** Pass in maxU16. The object passes back the number of items returned (same as max).

**pItems** Pass in pNull. The object passes back a pointer to the allocated block

2) Get the objects a few at a time. You repeatedly send msgTextEnumEmbeddedObjects re-using the same TEXT_ENUM_EMBEDDED structure. When the message returns stsEndOfData, there are no more objects in the enumeration. You should set the fields of TEXT_ENUM_EMBEDDED only before the first call. For successive calls you must not modify the fields.

**first** Same as Case 1.

**length** Same as Case 1.

**flags** Same as Case 1.
max  number of objects the pItems block can hold.
count  Pass in the same value as max. textData passes back the number of objects returned in block.  
       May be less than max for the last chunk, and is 0 when no further objects are left to enumerate.

pItems  pointer to a block that can hold at least max objects.

Return Value

stsOK  next chunk of objects has been enumerated

stsEndOfData  no more objects to enumerate. Passed back count is be zero. If pItems was nil and max was 0, then no block has been allocated.

Notifications

msgTextAffected
Notifies observers that a range of text has been affected.
Takes P_TEXT_AFFECTED, returns STATUS.

#define msgTextAffected                 MsgNoError(TCMakeMsg(7))

typedef struct TEXT_AFFECTED {
    TEXT_SPAN_AFFECTED      span;
    U16                     remeasure;
    P_UNKNOWN              spare;
} TEXT_AFFECTED, *P_TEXT_AFFECTED;

This message informs observers that the attributes of the range have been modified.

msgTextCounterChanged
Notifies observers that textData's changeCount has been modified.
Takes P_TEXT_COUNTER_CHANGED, returns STATUS.

#define msgTextCounterChanged              MsgNoError(TCMakeMsg(11))

typedef struct TEXT_COUNTER_CHANGED {
    OBJECT            sender;
    U32                changeCount;
    U32                oldCount;
} TEXT_COUNTER_CHANGED, *P_TEXT_COUNTER_CHANGED;

The changeCount is normally incremented by 1 as a result of handling msgTextModify. Observers here about these changes via msgTextReplaced and msgTextAffected notification messages.

However, the changeCount can change in other ways. For instance, the changeCount is rolled back as part of undoing certain operations. Also, clients and/or subclasses can explicitly set the changeCount via msgTextChangeEvent.

Whenever the changeCount changes in some way OTHER than a single increment by 1,
msgTextCounterChanged is sent to the observers to allow them to synchronize any caches they keep based on the changeCount.

msgTextReplaced
Notifies observers that a range of text was replaced via msgTextModify.
Takes P_TEXT_REPLACED, returns STATUS.

#define msgTextReplaced                 MsgNoError(TCMakeMsg(8))


typedef struct TEXT_REPLACED {
    TEXT_SPAN_AFFECTED span;
    TEXT_INDEX bytesTakenFromBuf;
} TEXT_REPLACED, *P_TEXT_REPLACED;
This file contains the API definition for `clsTextView` and `clsTextIP`. `clsTextView` inherits from `clsView`. `clsTextView` implements the user interface of a text editor. It uses an instance of `clsText` (or one of its subclasses) to hold its data. `clsTextIP` inherits from `clsIP`. `clsTextIP` is a specialization of `clsIP` used by a Text Views. The functions described in this file are contained in `TEXT.LIB`.

**Introduction**

An instance of `clsTextView` (or `textView`) provides a user interface which presents text data to the user and lets the user edit that data. Every `textView` has an associated data object of `clsText` (or a subclass of `clsText`). This object is referred to as `textData`.

**Painting Model**

A `textView` displays the `textData` as a series of non-overlapping, exhaustively tiling, horizontal display lines. With the possible exception of space below the last line, there is no area between lines that does not belong to any line. Characters are laid out left to right with lines running from top to bottom. When first created, the `textView` positions the first line of `textData` at the top of itself. Subsequent user or client actions (e.g. scrolling) can position some other line to the top of the window. However, the top line is always completely visible unless the view is too small to allow this. The last visible line, in contrast, may be clipped at the bottom.

Even though a `textView` is a descendant subclass of `clsBorder`, `clsTextView` ignores all `clsBorder` functionality relating to display of the view's background and border.

**Deferred Repaint**

A `textView` uses a "delayed repair" model in which several changes to the `textData` may be made before the visible display lines are repainted. For certain operations (e.g. selection change), such a delay can be misleading to the user and the individual operations provide a way to override the normal delay. If no override is available within a message's arguments, `msgTextViewRepair` can be used.

**Word Wrap**

By default, a `textView` displays each line beginning at the left edge of its window and "word wraps" at the right edge. That is, if a word would be clipped by the right edge of the window, it is instead placed at the beginning of the next line. By modifying paragraph margin attributes the line can be adjusted to have uninked margins in which no character is displayed.
Word wrap can be turned off by setting the `textView`'s style (see `msgTextViewSetStyle`). When off, a line breaks only when a "hard break" character (such as `teNewLine` or `teNewParagraph`) is encountered. As a result, a significant portion of many lines may be invisible to the user.

**Embedded Objects**

Other objects can be embedded within a `textView` (see `msgTextViewAddIP` and `msgTextViewEmbed`). (All embedded instances of some subclasses of `clsEmbeddedWin`.)

A `textView` handles an embedded object as if it is a "very large" character.

The `textView`'s displayed lines are always as tall as the tallest character or embedded object in the line. Therefore the presence of a large embedded object causes the containing line to be quite tall. (Not all embedded objects are large. For instance, closed application icons and reference buttons are only slightly larger than typical text.)

The baseline of the line containing embedded objects is determined, in part, by the embedded object's response to `msgWinGetBaseline`. (See `win.h`.)

**Text IPs**

An instance of `clsTextIP` (or `textIP`) implements two special features that are useful to `textView`s.

The first is size management. An embedded `textIP` tracks the width of its parent window. When the parent's width changes, an embedded `textIP` modifies its own width so that it fits within and completely fills the parent window (in the horizontal direction).

The second is special filtering of text going from the IP into a `textView`. A `textIP` filters translated data from its superclass (`clsIP`) before passing its data onto its client (typically a `textView`). Two kinds of filtering are performed: paragraph break insertion and space correction. A `textIP` inserts paragraph breaks based on how many blank lines there are between scribbles on an IP. `textIP` also filters out unnecessary spaces between words and adds spaces after a sentence-ending character such as a period or question-mark.

**Limitations**

`textView` is not WYSIWYG: although it will closely match font sizes and line breaks and spacing on a printer, it is based on a "make the printer match the screen" model that has enough variability that clients requiring WYSIWYG will find unacceptable (e.g., an overlaying mark-up layer).

`textView`s do not support multiple views of a single data object. Thus each `textView` is the unique view for its `textData` object. This restriction is not checked by `clsTextView`.

Although `TV_NEW_ONLY` has a "dc" field, there are so many restrictions on its use in PenPoint 1.0 that the field should always be left at the default value of `Nil(OBJECT)`. In addition, changing the units or scale used by the view-allocated "dc" is forbidden. This prevents "magnifying glass" and "pan in or out" effects from being used with a `textView`.
## Types and Constants

```c
typedef OBJECT TEXT_VIEW;
```

## Message Arguments

### TextView Style

The prefix "TV" indicates that an identifier is related to "TextView."

The prefix "tvs" indicates that an identifier is related to "text view style."

```c
typedef struct TV_STYLE {
    U16 flags;          // One of the values below
    S8 magnification;   // when tvsFormatForPrint is not on, this
                        // value (in points) is added to the
                        // character font sizes.
    U8 showSpecial;     // 0: show no special characters.
                        // 1: undefined -- do not use.
                        // 2: undefined -- do not use.
                        // 3: show all special characters.
                        // (6 spare bits)
                        // Not implemented. Should be null.
    OBJECT printer;
} TV_STYLE, *P_TV_STYLE;
```

Use these flags in the flags field of TV_STYLE:

- **tvsEmbedOnlyComponents** can only embed components. Cannot embed apps
- **tvsEmbedOnlyIPs** can only embed subclasses of clsIP. Can embed no other objects.
- **tvsFormatForPrinter** printer preview. style.magnification is ignored.
- **tvsQuietWarning** don’t display warning notes to user
- **tvsQuietError** don’t display error notes to user
- **tvsQuiet** both tvsQuietWarning and tvsQuietError
- **tvsReadOnlyChars** characters are read-only; user cannot add, remove or replace characters.
- **tvsReadOnlyAttrs** attributes are read-only; user cannot change any attribute information.
- **tvsReadOnly** both tvsReadOnlyChars and tvsReadOnlyAttrs
- **tvsWordWrap** break display line by wrapping words that don’t fit at the right edge of the view.

```
#define tvsEmbedOnlyComponents flag0
#define tvsEmbedOnlyIPs (tvsEmbedOnlyComponents|flag1)
#define tvsFormatForPrinter flag2
#define tvsQuietWarning flag3
#define tvsQuietError flag4
#define tvsQuiet (tvsQuietWarning|tvsQuietError)
#define tvsReadOnlyChars flag5
#define tvsReadOnlyAttrs flag6
#define tvsReadOnlyChars (tvsReadOnlyChars|tvsReadOnlyAttrs)
#define tvsWordWrap flag7
#define tvsSpare1 flag8 // Reserved
#define tvsSpare2 flag9 // Reserved
#define tvsSpare3 (flag10|flag11|flag12|flag13) // Reserved
#define tvsSpare4 flag14 // Reserved
#define tvsSpare5 flag15 // Reserved
```
Embedding

TV_EMBED_METRICS describes where and how to embed an object. The client either specifies the object to embed, or sets the embedded field to Nil and lets the text view create a new object based on the flags field. In the latter case, the UID of the newly created object is passed back in the embedded field.

typedef struct TV_EMBED_METRICS {
    TEXT_INDEX pos;        // In: embedded object is inserted just before this position.
    U16 flags;             // One of the values below
    OBJECT embedded;       // In-Out: the UID of the embedded object
} TV_EMBED_METRICS, *P_TV_EMBED_METRICS;

Use these in the flags field of a TV_EMBED_METRICS.

#define tvEmbedAnnotate flag0 // Not implemented
#define tvEmbedFloat flag1    // Make the embeddee floating
#define tvEmbedReplace flag2  // The IP's contents replace the character following the IP.

Use this in the flags field of a TV_EMBED_METRICS.

#define tvEmbedAddMargin flag5 // Leave small between previous line and the IP.

Use these in the flags field of a TV_EMBED_METRICS when using the struct as the pArgs to msgTextViewAddIP.

#define tvEmbedAtEnd flag8    // IP should be last char of data.
#define tvEmbedPara flag9     // IP is a paragraph pad
#define tvEmbedOneChar flag10 // IP is only 1-char
#define tvEmbedPreload flag11 // preload the selection into the IP
#define tvEmbedDisplayType (flag13|flag14|flag15) // Obsolete.

Resolution

The prefix "tvr" indicates that an identifier is related to "text view resolve."

The values for the xRegion and yRegion fields of a TV_RESOLVE struct are illustrated here. The values are of the form (xRegion, yRegion).

```
(-1,1) | (0,1) | (1,1)

-------------

| Line's ink |
(-1,0) | (0,0) | (1,0)

-------------

|       |
(-1,-1) | (0,-1) | (1,-1)
```
The fields of this structure are described in more detail in the comments for msgTextviewResolveXY.

typedef struct TV_RESOLVE {
    XY32 xy; // In:Out: Units are LWC
    U16 flags; // One of the values below
    TEXT_INDEX pos; // Out: Pos of char containing xy, or
    // maxTEXT_INDEX if no such char
    TEXT_INDEX lineStart; // Out: Pos of first char on line
    // containing xy, or maxTEXT_INDEX
    // if no line contains xy.
    S8 xRegion; // Out: Region x was in. See diagram.
    S8 yRegion; // Out: Region y was in. See diagram.
    TEXT_INDEX selects; // Out: Pos of char "selected" by xy
    XY32 offset; // Out: Offset to prev/next char's ink
    P_UNKNOWN spares[4]; // Reserved.
} TV_RESOLVE, *P_TV_RESOLVE;

Use these flags in the flags field of TV_RESOLVE. Note that they are not completely orthogonal; in particular, only one of [tvrSelFirst, tvrSelLPO and tvrBalance] should be enabled at once, similarly for [tvrPrevChar and tvrNextChar].

tvrSelFirst causes TV_RESOLVE.selects to be <= TV_RESOLVE.pos (i.e., the "selected" character is at or before the character "hit" by TV_RESOLVE.xy.)

tvrSelLPO causes TV_RESOLVE.selects to be >= TV_RESOLVE.pos (i.e., the "selected" character is after the character "hit" by TV_RESOLVE.xy, unless the line contains only one character in which case TV_RESOLVE.selects == TV_RESOLVE.pos.)

tvrBalance has the effect of tvrSelFirst or tvrSelLPO, depending on which edge of the character "hit" by TV_RESOLVE.xy is closest to TV_RESOLVE.xy.x.

tvrSelWord causes the "selection" behavior specified by any of the previous three flags to occur for the "word" containing the character "hit" by TV_RESOLVE.xy.x.

tvrPrevChar normally TV_RESOLVE.offset.x is 0 upon return. Enabling tvrPrevChar causes TV_RESOLVE.offset.x to contain the amount that TV_RESOLVE.xy.x exceeds the x coordinate of the lower-left corner of the character specified by TV_RESOLVE.pos (i.e., the distance past the previous character's right edge).

tvrNextChar normally TV_RESOLVE.offset.x is 0 upon return. Enabling tvrNextChar causes TV_RESOLVE.offset.x to contain the amount that TV_RESOLVE.xy.x falls short of the x coordinate of the lower-right corner of the character specified by TV_RESOLVE.pos (i.e., the distance before the next character's left edge).

tvrPastEOL normally a line contains only those character positions for the characters displayed on the line. tvrPastEOL permits TV_RESOLVE.selects to return with the TEXT_INDEX of the first character of the following line if the specified TV_RESOLVE.xy.x is to the right of the last character in the line.

tvrNLIfPastEOL when disabled, if TV_RESOLVE.xy.x is to the right of the last character in a line with a hard line break (e.g., teNewLine or teNewParagraph) and at least one other character, TV_RESOLVE.selects specifies the character immediately before the hard line break. When enabled, if tvrPastEOL is also enabled and would have caused TV_RESOLVE.selects to be after the hard line break, tvrNLIfPastEOL will override and cause TV_RESOLVE.selects to specify the break character instead.

#define tvrSelFirst flag0
#define tvrSelLPO flag1
#define tvrSelWord flag2
#define tvrPrevChar flag3
#define tvrNextChar flag4
#define tvrBalance flag5
#define tvrPastEOL flag6
#define tvrNLIfPastEOL flag7
Selection

The prefix "tvs" indicates that an identifier is related to "text view select."

The fields of this structure are described in more detail in the comments for msgTextViewSetSelection.

```c
typedef struct TV_SELECT {
    TEXT_INDEX first; // lpoTEXT_INDEX means "clear selection"
    TEXT_INDEX length; // 0 results in an 0 length selection
    U16 flags; // either 0 or wsSynchRepaint (see win.h)
    ATOM level; // Obsolete. Don’t use.
} TV_SELECT, *P_TV_SELECT;
```

Scrolling

The prefix "ts" indicates that an identifier is related to "text view scroll."

```c
typedef struct TV_SCROLL {
    TEXT_INDEX pos; // Position to scroll to
    U32 flags; // One of the values below
} TV_SCROLL, *P_TV_SCROLL;
```

Use these in the flags field of a TV_SCROLL.

- tsAlignAtTop: scroll so that pArgs->pos is "near the top." See tsAlignEdge.
- tsAlignAtBottom: scroll so that pArgs->pos is "near the bottom." See tsAlignEdge.
- tsAlignAtCenter: scroll so that pArgs->pos is in the center displayed line.
- tsAlignEdge: If set, and tsAlignAtTop or tsAlignAtBottom is set, this flag forces the line containing pArgs->pos to be the exact edge. If this flag is off, and tsAlignAtTop tsAlignAtBottom is set, the textView tries to leave an extra line or two between the line containing pArgs->pos and the view’s edge.
- tsIffInvisible: If set, the textView scrolls only if pArgs->pos is not already visible. If not set, the textView scrolls even if pArgs->pos is visible.
- textNoScrollNotify: By default, the scrollbar(s) for the view are notified (via a msgWinSend of msgScrollbarUpdate) that they should update after a msgTextViewScroll. If this flag is set, the notification is not sent.

```c
#define tsAlignAtTop 0L
#define tsAlignAtBottom 1L
#define tsAlignAtCenter 2L
#define tsAlignEdge (((U32)flag2)
#define tsIffInvisible (((U32)flag3)
#define textNoScrollNotify (((U32)flag15)
```

Messages Defined by Other Classes

- msgNewDefaults
  Initializes the NEW structure.
  Takes p_TV.NEW, returns STATUS. Category: class message.

  Zeros out pNew->tv and sets:

  ```c
tv.style.flags = tvsWordWrap;
tv.flags = tvFillWithIP;
```
### Messages Defined by Other Classes

<table>
<thead>
<tr>
<th>Message</th>
<th>Arguments</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>win.flags.style</code></td>
<td>`</td>
<td>= wsGrowBottom</td>
</tr>
<tr>
<td><code>view.createDataObject</code></td>
<td><code>= true;</code></td>
<td><code>gWin.helpId</code> <code>= tagTextView;</code></td>
</tr>
</tbody>
</table>

#### msgNew

Creates a new instance of `clsTextView`.

Takes `P_TV_NEW`, returns `STATUS`. Category: class message.

**Comments**

If `pArgs->view.createDataObject` is true, then the `textView` creates a Text data object (`clsText`; see `txtdata.h`) and sets the view's data object. If `pArgs->tv.dc` is `NULL`, the `textView` creates a DC for its exclusive use.

#### msgGWinXList

Defined in `gwin.h`.

Takes `P_XLIST`, returns `STATUS`.

**Comments**

In response to this message, a `textView` typically performs some editing operation on its associated data object. A `textView` can process both "vanilla" xlists as described in `xlist.h` or text-specific xlists as `txtxlist.h`.

Here's how a `textView` responds to each xlist element:

- `xtBounds` remembers the bounds of a gesture element
- `xtGesture` processes the gesture
- `xtText` inserts the text
- `xtObject` embeds the object
- `xtCharAttrs` modifies the character attributes of the specified characters
- `xtParaAttrs` modifies the attributes of the specified paragraphs
- `xtTabs` modifies the tabs of the specified paragraphs
- `xtCharPos` sets the insertion point for text to the specified character position

### Messages

#### msgTextViewAddIP

Adds an insertion pad to the `textView`.

Takes `P_TV_EMBED_METRICS`, returns `STATUS`.

```c
#define msgTextViewAddIP TVMakeMsg(0)
```

**Message Arguments**

```c
typedef struct TV_EMBED_METRICS {
    TEXT_INDEX pos; // In: embedded object is inserted just before this position.
    UI6 flags; // One of the values below
    OBJECT embedded; // In-Out: the UID of the embedded object
} TV_EMBED_METRICS, *P_TV_EMBED_METRICS;
```

**Comments**

The client must set all of the fields of `pArgs` as described in the discussion of `TV_EMBED_METRICS`. 
msgTextViewCheck
A textView performs a self-consistency check.
Takes P_UNKNOWN, returns STATUS.

#define msgTextViewCheck TVMakeMsg(5)

Comments
This message is only available in the debugging version of text.dll. The only currently defined value for pArgs is zero.

Return Value
stsOK  no problems detected
< stsOK  problems detected

msgTextViewEmbed
Embeds an object in the textView. Makes associated changes in text data.
Takes P_TV_EMBED_METRICS, returns STATUS.

#define msgTextViewEmbed TVMakeMsg(1)

Message Arguments
typedef struct TV_EMBED_METRICS {
  TEXT_INDEX pos;  // In: embedded object is inserted
  U16 flags;  // just before this position.
  OBJECT embedded;  // One of the values below
} TV_EMBED_METRICS, *P_TV_EMBED_METRICS;

Comments
The client must set all of the fields of pArgs as described in the discussion of TV_EMBED_METRICS.

msgTextViewGetEmbedMetrics
Passes back the textView-specific metrics for an embedded object.
Takes P_TV_EMBED_METRICS, returns STATUS.

#define msgTextViewGetEmbedMetrics TVMakeMsg(2)

Message Arguments
typedef struct TV_EMBED_METRICS {
  TEXT_INDEX pos;  // In: embedded object is inserted
  U16 flags;  // just before this position.
  OBJECT embedded;  // One of the values below
} TV_EMBED_METRICS, *P_TV_EMBED_METRICS;

Comments
The client must only fill in pArgs->embedded.

msgTextViewRepair
Forces a delayed paint operation to take place immediately.
Takes pNull, returns stsOK.

#define msgTextViewRepair TVMakeMsg(3)

Comments
Use with caution, as overuse of this message significantly degrades performance.

msgTextViewResolveXY
Given a point in LWC space, passes back the character at (or near) the point.
Takes P_TV_RESOLVE, returns STATUS.

#define msgTextViewResolveXY TVMakeMsg(4)
typedef struct TV_RESOLVE {
    XY32 xy;       // In/Out: Units are LWC
    UI6 flags;    // One of the values below
    TEXT_INDEX pos;  // Out: Pos of char containing xy, or
                     // maxTEXT_INDEX if no such char
    TEXT_INDEX lineStart;  // Out: Pos of first char on line
                           // containing xy, or maxTEXT_INDEX
                           // if no line contains xy.
    S8 xRegion;    // Out: Region x was in. See diagram.
    S8 yRegion;    // Out: Region y was in. See diagram.
    TEXT_INDEX selects;  // Out: Pos of char "selected" by xy
    XY32 offset;  // Out: Offset to prev/next char's ink
    P UNKNOWN spares[4];  // Reserved.
} TV_RESOLVE, *p_TV_RESOLVE;

Client arguments control exactly which character is "selected", and how much information is provided by the message.

Clients can also use this message to "reverse resolve" as follows. If both pArgs->xy.x and pArgs->xy.y are maxS32, then the textView sets pArgs->xy to the coordinates of the lower left corner of the character at pArgs->pos.

Warning: The response to this message always updates pArgs->xy to reflect information about the line either containing (or near) the original xy (or pos).

"LWC" is short for Local Window Coordinates. See win.h for more information.

Return Value

stsBadParam if no line's y extents include pArgs->xy.y

stsNoMatch if a containing line exists but it has no character under pArgs->xy.x; of if reverse resolve of a character not contained in any display line

msgTextViewScroll

Repositions displayed text within the textView.

Takes P_TV_SCROLL, returns stsOK.

#define msgTextViewScroll TVMakeMsg(6)

typedef struct TV_SCROLL {
    TEXT_INDEX pos;  // Position to scroll to
    U32 flags;  // One of the values below
} TV_SCROLL, *P_TV_SCROLL;

The client must set the fields of pArgs as described in the discussion of TV_SCROLL.

msgTextViewGetStyle

Passes back a textView's style.

Takes P_TV_STYLE, returns stsOK.

#define msgTextViewGetStyle TVMakeMsg(8)

typedef struct TV_STYLE {
    U16 flags;  // One of the values below
    S8 magnification;  // when tvsFormatForPrint is not on, this
                       // value (in points) is added to the
                       // character font sizes.
    U8 showSpecial;  // 0: show no special characters.
                     // 1: undefined -- do not use.
                     // 2: undefined -- do not use.
                     // 3: show all special characters.
                     // (6 spare bits)
    OBJECT printer;  // Not implemented. Should be null.
} TV_STYLE, *P_TV_STYLE;
**msgTextViewSetSelection**

Selects one or more characters displayed by the textView.

Takes P_TV_SELECT, returns stsOK.

```c
#define msgTextViewSetSelection TVMakeMsg(9)
```

**Message**

typedef struct TV_SELECT {
    TEXT_INDEX first;         // lpoTEXT_INDEX means "clear selection"
    TEXT_INDEX length;       // 0 results in an 0 length selection
    U16 flags;               // either 0 or wsSynchRepaint (see win.h)
    ATOM level;              // Obsolete. Don't use.
} TV_SELECT, *P_TV_SELECT;

**Arguments**

- `first`: The first character to select. The value lpoTEXT_INDEX means that cause the selection to be cleared.
- `length`: Number of characters to select. The value 0 results in a zero-length I-Bean selection.
- `flags`: If this field is wsSynchRepaint (defined in win.h) the textView repaint immediately. Otherwise this field must be zero.

While handling this message, the textView becomes the selection owner unless pArgs->first is lpoTEXT_INDEX, in which case the text view ensures that it is NOT the selection owner.

**msgTextViewSetStyle**

Sets a textView's style.

Takes P_TV_STYLE, returns stsOK.

```c
#define msgTextViewSetStyle TVMakeMsg(10)
```

**Message**

typedef struct TV_STYLE {
    U16 flags;                     // One of the values below
    S8 magnification;             // when tvsFormatForPrint is not on, this
    // value (in points) is added to the
    // character font sizes.
    U8 showSpecial;               // 0: show no special characters.
    // 1: undefined -- do not use.
    // 2: undefined -- do not use.
    // 3: show all special characters.
    // (6 spare bits)
    OBJECT printer;              // Not implemented. Should be null.
} TV_STYLE, *P_TV_STYLE;

**Arguments**

- `flags`: If this field is wsSynchRepaint (defined in win.h) the textView repaint immediately. Otherwise this field must be zero.

pArgs->printer should be set to nil(OBJECT).

---

**Definitions for msgNew**

```c
#ifndef NO_NEW
#ifndef txtViewNewFields
#ifndef
#include <view.h>
#endif
#endif
```

See comment with msgNew and msgNewDefaults for more information.

```c
typedef struct TV_NEW_ONLY {
    U16 flags;                     // One of the values below
    OBJECT dc;
    TV_STYLE style;
} TV_NEW_ONLY, *P_TV_NEW_ONLY;
```
Use this in the flags field of a TV_NEW_ONLY.

```c
#define tvFillWithIP flag0
#define tvbViewNewFields \  
    viewNewFields \  
    TV_NEW_ONLY  tv;
typedef struct TV_NEW {  
    tvbViewNewFields  
} TV_NEW, *P_TV_NEW;
```

**Utility Functions**

**TextCreateTextScrollWin**

Utility function that creates a `textView` (with a data object) placed inside a scroll window. (See swin.h.)

Returns STATUS.

STATUS EXPORTED

Function Prototype

```c
TextCreateTextScrollWin(
    P_TV_NEW pNew,
    P_OBJECT scrollWin);
```

Comments

Clients often need a "vanilla" `textView` inside a vanilla `scrollWin`. This function does just that. Clients can modify the created objects after the creation if this function doesn't do quite the right thing. Client who need more control over the creation should probably create the objects manually.

The `pNew` parameter should be null or should point at an already initialized NEW struct. If it is null, then the function creates a default instance of clsTextView.

Because the view is created with `formatForPrinter` FALSE, the `scrollWin`'s `expandChildWidth` is set to true. This causes the `scrollWin` to manage the width of the `textView`.

Here's a simplified indication of how the `scrollWin` is created:

```c
ObjectCall(msgNewDefaults, clsScrollWin, &sn)  
sn.scrollWin.clientWin = <the text view>  
sn.scrollWin.style.vertScrollbar = true;  
sn.scrollWin.style.autoVertScrollbar = false;  
sn.scrollWin.style.expandChildWidth = true;  
sn.scrollWin.style.expandChildHeight = true;  
sn.scrollWin.style.contractChildWidth = true;  
sn.scrollWin.style.contractChildHeight = true;  
sn.scrollWin.style.vertClient = swClientWin;  
sn.scrollWin.style.horizClient = swClientScrollWin;  
sn.win.flags.input |= inputHoldTimeout;  
sn.scrollWin.style.forward = swForwardGesture;  
if (<creating on screen>) {  
    sn.border.style.leftMargin = bsMarginMedium;  
    sn.border.style.rightMargin = bsMarginMedium;  
    sn.border.style.topMargin = bsMarginMedium;  
} else {  
    sn.border.style.leftMargin = bsMarginNone;  
    sn.border.style.rightMargin = bsMarginNone;  
    sn.border.style.topMargin = bsMarginNone;  
}  
ObjectCall(msgNew, clsScrollWin, &sn);  
*scrollWin = sn.object.uid;
```

Warning: When printing, the `scrollWin` and `textView` are probably restored, not created anew. Therefore the client needs to go in and set the `scrollWin`'s margins to 0.
typedef struct TEXTIP_METRICS {
    U16 flags; // Reserved.
} TEXTIP_METRICS, *P_TEXTIP_METRICS,
    TEXTIP_NEW_ONLY, *P_TEXTIP_NEW_ONLY;

msgNewDefaults

Initializes the NEW struct.

Takes P_TEXTIP_NEW, returns STATUS. Category: class message.

In response to this message, clsTextIP does the following:

    pArgs->win.flags.style = wsSendGeometry | wsSendFile |
                        wsShrinkWrapHeight;
    pArgs->ip.rows = 5;
    pArgs->ip.lines = 5;

If the user input pad style preference is Boxed:

    pArgs->ip.style.displayType = ipsCharBox;
    pArgs->ip.style.delayed = 1;

If the user input pad style preference is Ruled:

    pArgs->ip.style.displayType = ipsRuledLines;

If the user input pad style preference is RuledAndBoxed:

    pArgs->ip.style.displayType = ipsRuledLines;
    pArgs->ip.style.ruledToBoxed = true;

msgNew

Creates a new instance of clsTextIP.

Takes P_TEXTIP_NEW, returns STATUS. Category: class message.

msgTextIPGetMetrics

Passes back a textIP's metrics.

Takes P_TEXTIP_METRICS, returns stsOK.

#define msgTextIPGetMetrics MakeMsg(clsTextIP, 1)
msgTextIPSetMetrics

Sets a textIP's metrics.

Takes P_TEXTIP_METRICS, returns stsOK.

```c
#define msgTextIPSetMetrics MakeMsg(clsTextIP, 2)
#ifndef NO_NEW
#ifndef textIPNewFields
#ifndef INSERT_INCLUDED
#include <insert.h>
#endif
#define textIPNewFields
#endif
#endif
#endif

typedef struct TEXTIP_METRICS {
    U16 flags;  // Reserved.
} TEXTIP_METRICS, *P_TEXTIP_METRICS,
```
This file contains the Text subsystem additions to xlist (see xlist.h).

A Text View (see txtView.h) gathers input directly from the user via keyboard input delivered by msgInputEvent, with Cls(pArgs->devCode) == Cls(clsKey);

low-level pen input also msgInputEvent, but Cls(clsPen);

gestures delivered by msgGWinXlist; and

insertion pads which provide data starting with msgIPDataAvailable.

The user input delivered to a Text View from an insertion pad is communicated via an xlist. As a result of its processing of the xlist, the Text View modifies its associated data object. Each xlist moves through the following stages: (1) it comes into being as a way for the hwx system to provide low-level information about the user input to clsIP (see insert.h); (2) clsIP packages the low-level information into medium-level information which is self-sent; (3) finally, clsTextIP re-interprets this information and packages it into high-level information which requires concepts specific to the Text subsystem.

Thus, an xlist from a TextIP (see txtView.h) can contain one or more elements of the following specialized types. For each type, the constraint on the structure of the information pointed to by the pData field of the XLIST_ELEMENT is listed.

*xtCharAttrs* pData points to an XLIST_CHAR_ATTRS;

*xtParaAttrs* pData points to an XLIST_PARA_ATTRS;

*xtTabs* pData points to an XLIST_TABS;

*xtCharPos* pData is a TEXT_INDEX (cast to a P_UNKNOWN).

The types themselves are defined as part of XTYPE in xlist.h; the data structures and their semantics are defined below.

In general, an xlist is position-independent. However, the caller of msgGWinXlist often wants the associated xlist to modify a Text View's data object beginning at a particular character index; an element of type xtCharPos allows the caller to specify such an index.

To make it easier to maintain the position-independent property of an xlist, Text Views recognize maxTEXT_INDEX (see txtData.h) as having a special meaning when used as the value of the first field of the pData in an xlist element of type xtCharAttr, xtParaAttr, and xtTabs (i.e., pData->first == maxTEXT_INDEX). If the pData->length is 0, a pData->first of maxTEXT_INDEX causes the xlist processing code to remember the current index in the Text data object and to take no other action; if the pData->length is non-zero, the pData->first of maxTEXT_INDEX causes the xlist processing code to update pData->first with the previously remembered index. This allows the caller of msgGWinXlist to generate an xlist with the following structure:

*xtCharPos* to start processing at a particular index;

*xtText* one or more times, to add characters;

*xtCharAttr* with first of maxTEXT_INDEX, length of 0;

*xtText* one or more times, to add more characters;
xtCharAttrs with first of maxTEXT_INDEX, length not 0, thereby setting the character attributes for exactly the bracketed characters.

```
#ifndef TXTXLIST_INCLUDED
#define TXTXLIST_INCLUDED
#endif
#ifndef XLIST_INCLUDED
#include <xlist.h>
#endif
#ifndef TXTDATA_INCLUDED
#include <txtData.h>
#endif
```

Upon encountering an xlist element of type xtCharAttrs, a Text View does a msgTextChangeAttrs to its data object, making use of the fields of the P_XLIST_CHAR_ATTRS by mapping them to the corresponding fields of TEXT_CHANGE_ATTRS as follows:

- **tag** forced to atomChar
- **first** copied from first
- **length** copied from length
- **pNewMask** set to &mask
- **pNewValues** set to &attrs

```
typedef struct {
    TEXT_INDEX     first;
    TEXT_INDEX     length;
    TA_CHAR_MASK   mask;
    TA_CHAR_ATTRS  attrs;
} XLIST_CHAR_ATTRS, *P_XLIST_CHAR_ATTRS;
```

Upon encountering an xlist element of type xtParaAttrs, a Text View does a msgTextChangeAttrs to its data object, making use of the fields of the P_XLIST_PARA_ATTRS by mapping them to the corresponding fields of TEXT_CHANGE_ATTRS as follows:

- **tag** forced to atomPara
- **first** copied from first
- **length** copied from length
- **pNewMask** set to &mask
- **pNewValues** set to &attrs

```
typedef struct {
    TEXT_INDEX     first;
    TEXT_INDEX     length;
    TA_PARA_MASK   mask;
    TA_PARA_ATTRS  attrs;
} XLIST_PARA_ATTRS, *P_XLIST_PARA_ATTRS;
```

Upon encountering an xlist element of type xtTabs, a Text View does a msgTextChangeAttrs to its data object, making use of the fields of the P_XLIST_TABS by mapping them to the corresponding fields of TEXT_CHANGE_ATTRS as follows:

- **tag** forced to atomParaTabs
- **first** copied from first
- **length** copied from length
- **pNewMask** set to Nil()
pNewValues set to &tabs

typedef struct {
    TEXT_INDEX first;
    TEXT_INDEX length;
    TA_MANY_TABS tabs;
} XLIST_TABS, *P_XLIST_TABS;
Part 7 / File System
This file defines common file types used for import and export between PenPoint and other operating systems.

```c
#ifndef FILETYPE_INCLUDED
#define FILETYPE_INCLUDED
#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef UID_INCLUDED
#include <uid.h>
#endif
#endif
```

The following file types are common enough to merit a central registry. Contact GO Developer Technical Support if you want to add a file type to the registry.

The file types are defined as tags; they are primarily intended to be stored as the value of the `fsAttrFileType` file attribute. If a file is explicitly typed via this mechanism, applications can more easily decide if they can import it.

```c
#define fileTypeUndefined (TAG)OL
```

`fileTypeASCII` implies 8-bit bytes encoding the 7-bit ASCII set defined by ANSI X3.64. Any byte with value greater than \(0x7F\) will be interpreted in a manner dependent on the subsystem involved; e.g. `clsText` (and thus the MiniText application) will assume the bytes encode IBM-PC Code Page 850.

```c
#define fileTypeASCII MakeTag(clsFileHandle, 0)
```

`fileTypeASCIISoftLineBreaks` is similar to `fileTypeASCII`. The difference is that in a line that has no explicit new line or carriage return, a space is transformed into a line feed near the 72nd character.

```c
#define fileTypeASCIISoftLineBreaks MakeTag(clsFileHandle, 1)
```

`fileTypeRTF` implies Microsoft Corporation's Rich Text Format (RTF).

```c
#define fileTypeRTF MakeTag(clsFileHandle, 2)
```

`fileTypeTIFF` implies Aldus Corporation and Microsoft Corporation's Tag Image File Format (TIFF).

```c
#define fileTypeTIFF MakeTag(clsFileHandle, 3)
```

`fileTypePicSeg` implies Go Corporation's Picture Segment format.

```c
#define fileTypePicSeg MakeTag(clsFileHandle, 4)
```
This file contains the API for clsDirHandle and clsFileHandle. The functions described in this file are contained in PENPOINT.LIB.

clsFileSystem inherits from clsObject.

Provides file system support. theFileSystem is the only instance of clsFileSystem.

clsDirHandle inherits from clsObject.

Provides file system directory support. theBootVolume is a well known instance of clsDirHandle. theSelectedVolume is a well known instance of clsDirHandle. theWorkingDir is a well known instance of clsDirHandle.

clsFileHandle inherits from clsStream.

Provides file system file access support.

```
#ifndef FS_INCLUDED
#define FS_INCLUDED

#include <go.h>
#endif

#include <uid.h>
#endif

#include <clsmgr.h>
#endif

#include <uuid.h>
#endif

#include <stream.h>
#endif
```

**Debugging Flags**

File System Debugging Flag is '$', values are:

- 0001 Debug info when fs cache layer calls volume layer
- 0200 Breaks into debugger before asking to insert disk
- 20000 Display list of known volumes when prompting for unmounted disk

Include file dependencies for this include file

```
#ifndef GO_INCLUDED
#define GO_INCLUDED
#include <go.h>
#endif

#ifndef UID_INCLUDED
#define UID_INCLUDED
#include <uid.h>
#endif

#ifndef CLSMGR_INCLUDED
#define CLSMGR_INCLUDED
#include <clsmgr.h>
#endif

#ifndef UUID_INCLUDED
#define UUID_INCLUDED
#include <uuid.h>
#endif

#ifndef STREAM_INCLUDED
#define STREAM_INCLUDED
#include <stream.h>
#endif
```

Common abbreviations, terms:

- FS  File System
- Node  A file or a directory
- Dir  A directory
Rules concerning the destination of file system messages:

All messages defined in this file are directed to their destination via ObjectCall, the file system does not accept messages that are sent. All messages (with the exception of msgFSGetInstalledVolumes) of clsFileSystem can be "sent" to either a file or a dir object. Messages of clsDirHandle can only be "sent" to directory objects. Messages of clsFileHandle can only be "sent" to file objects.

Common #defines and typedefs

Defines

```c
#define fsMaxPathLength 254 // Max path length
#define fsPathBufLength (fsMaxPathLength+1) // Buffer size for max path
#define fsSeparator '"' // Pathname separator
#define fsEscapeChar 'I' // Escape char (invalid in paths)
#define fsUniqueSeparator ' ' // Char for unique name postfix
#define fsMaxHandles 255 // Max handles on a single node
#define fsMaxUnique 255 // Max tries to make name unique
#define fsMaxReadWrite 65535 // Max size for single read/write
#define fsMaxNestingLevel 20 // Max nesting for recursive ops
```

FS Attribute Intrinsics

These are used to build file/directory attribute labels or to get component pieces from an attribute label.

A client can define their own attribute using one of the FSMakeXXXAttr intrinsics, specifying a class and a tag. The attribute type will allow for storage of a 32 bit value (Fix32), a 64 bit value (Fix64), a null terminated string of any length up to 32K (Str), or a variable length value up to 32K (Var). The messages msgFSGetAttr, msgFSSetAttr, msgFSReadDir, msgFSReadDirFull and msgFSTraverse use file system attributes to represent the attribute label.

```c
#define fsFixAttr 0
#define fsFix64Attr 1
#define fsVarAttr 2
#define fsStrAttr 3
#define fsMaxAttrLength 255
#define FSMakeAttr(cls,t,f) \ FSMakeAttrWithFlags(cls,t,f)
#define FSMakeFix32Attr(cls,t) FSMakeAttr(cls,t,fsFixAttr)
#define FSMakeFix64Attr(cls,t) FSMakeAttr(cls,t,fsFix64Attr)
#define FSMakeVarAttr(cls,t) FSMakeAttr(cls,t,fsVarAttr)
#define FSMakeStrAttr(cls,t) FSMakeAttr(cls,t,fsStrAttr)
#define FSAttr(attr) TagNum(attr)
#define FSAttrCls(attr) ClsNum(attr)
#define FSAttrIsFix32 (attr) (TagFlags(attr) == fsFixAttr)
#define FSAttrIsFix64 (attr) (TagFlags(attr) == fsFix64Attr)
#define FSAttrIsVar (attr) (TagFlags(attr) == fsVarAttr)
#define FSAttrIsStr (attr) (TagFlags(attr) == fsStrAttr)
```

File System Attributes

These are the predefined attributes managed by the file system.

```c
#define fsNullAttrLabel FSMakeFix32Attr(objNull, 0)
#define fsAttrName FSMakeStrAttr(clsFileSystem, 0)
#define fsAttrFlags FSMakeFix32Attr(clsFileSystem, 0)
#define fsAttrDateCreated FSMakeFix32Attr(clsFileSystem, 2)
#define fsAttrDateModified FSMakeFix32Attr(clsFileSystem, 3)
#define fsAttrFileSize FSMakeFix32Attr(clsFileSystem, 4)
```
Common #defines and typedefs

#define fsAttrDirIndex (fsMakeFix64Attr(clsDirHandle, 0))
#define fsAttrOldDirIndex (fsMakeFix64Attr(clsDirHandle, 1))
#define fsFileType (FSMakeFix32Attr(clsFileHandle, 0))

See msgFSGetAttr for an explanation when to use these constants.

#define fsAllocAttrLabelsBuffer ((P_FS_ATTR_LABEL)maxU32)
#define fsAllocAttrValuesBuffer ((P_UNKNOWN)maxU32)
#define fsAllocAttrSizesBuffer ((P_FS_ATTR_SIZE)maxU32)

### Status Codes

#### Status Codes

Common return values:

There are a few status return values that are common to either all messages or to a group of messages (i.e. messages that try to change the volume).

stsFSHandleInvalid The dir/file object refers to a node that has been previously deleted.

stsFSVolDisconnected The volume is not connected.

stsFSVolFull The message cannot complete, due to insufficient space on the volume.

stsFSVolReadOnly The message cannot complete, because the volume is write protected.

Error Status Codes

#define stsFSVolDisconnected MakeStatus(clsFileSystem, 0)
#define stsFSVolReadOnly MakeStatus(clsFileSystem, 1)
#define stsFSVolFull MakeStatus(clsFileSystem, 2)
#define stsFSNodeNotFound MakeStatus(clsFileSystem, 3)
#define stsFSNodeReadOnly MakeStatus(clsFileSystem, 4)
#define stsFSSubjectDenied MakeStatus(clsFileSystem, 5)
#define stsFSCircularMoveCopy MakeStatus(clsFileSystem, 6)
#define stsFSVolBusy MakeStatus(clsFileSystem, 7)
#define stsFSNodeBusy MakeStatus(clsFileSystem, 8)
#define stsFSBadPath MakeStatus(clsFileSystem, 9)
#define stsFSUniqueFailed MakeStatus(clsFileSystem, 10)
#define stsFSDirFull MakeStatus(clsFileSystem, 11)
#define stsFSNodeExists MakeStatus(clsFileSystem, 12)
#define stsFSNotDir MakeStatus(clsFileSystem, 13)
#define stsFSNotFile MakeStatus(clsFileSystem, 14)
#define stsFSSubjectOnlyAttr MakeStatus(clsFileSystem, 15)
#define stsFSBuffTooSmall MakeStatus(clsFileSystem, 16)
#define stsFSNestingTooDeep MakeStatus(clsFileSystem, 17)
#define stsFSNodeParent MakeStatus(clsFileSystem, 18)
#define stsFSUnchangeable MakeStatus(clsFileSystem, 19)
#define stsFSNodeAncestor MakeStatus(clsFileSystem, 20)
#define stsFSDirPositionLost MakeStatus(clsFileSystem, 21)
#define stsFSTimeInvalid MakeStatus(clsFileSystem, 22)
#define stsFSDifferent MakeStatus(clsFileSystem, 23)
#define stsFSTooManyHandles MakeStatus(clsFileSystem, 24)
#define stsFSDirIndexExists MakeStatus(clsFileSystem, 25)
#define stsFSDirIndexNotFound MakeStatus(clsFileSystem, 26)
#define stsFSVolCorrupt MakeStatus(clsFileSystem, 27)

Informational Status Codes

#define stsFSAAttrBuffTooSmall MakeWarning(clsFileSystem, 1)
Locators are structures used to describe the location of a file or dir node. There are two types of locators: explicit and implicit. An explicit locator is defined with `FS_LOCATOR` which specifies both the starting node (uid) and the path relative to the starting node (pPath). An implicit locator is made up of a starting node (the object that receives a message) and the path relative to the starting node (pPath).

`msgFSMove` is a good example of a message that contains both types of locators. The receiver of `msgFSMove` and move.pSourcePath defines the implicit location of the source of the move. move.destLocator defines the explicit location of the dest of the move.

The uid field of a locator must be filled in and must be non-null. If no other choice can be decided upon, the WorkingDir may be a good one. The uid field does not always have to be a dir handle object. The uid can be a file handle object if the pPath field points to a path that begins with .. (parent), \ (root) or \ \ (fully specified path including volume name).

The path field of locators (explicit and implicit) are relative to the node defined by the uid (or object receiving the message) unless the path begins with a \ (root relative) or \ \ (fully specified path).

```c
typedef struct FS_LOCATOR {
    OBJECT uid;
    P STRING pPath;     // Relative to node defined by uid
} FS_LOCATOR, * P_FS_LOCATOR;
```

The file system interface never uses flat locators, but if it is more convenient to hold the entirety of the locator in a linear structure using flat locators.

```c
typedef struct FS_FLAT_LOCATOR {
    OBJECT uid;
    U8 path [fsPathBuf Length];
} FS_FLAT_LOCATOR, * P_FS_FLAT_LOCATOR;
```

**Types**

```c
typedef U16 FS_ATTR_SIZE;
typedef U32 FS_ATTR_LABEL;
Enum16 (FS_VOL_TYPE) {
    fsAnyVolType = 0,     // Match any vol type for msgNew
    fsVolTypeMemory = 0,
    fsVolTypeDisk = 1,
    fsVolTypeRemote = 2
};
```
Common #defines and typedefs

Enum16(FS_VOL_FLAGS) {
    fsVolReadOnly = flag0,
    fsVolConnected = flag1,
    fsVolRemovableMedia = flag2,
    fsVolEjectableMedia = flag3,
    fsVolDirsIndexable = flag4,
    fsVolFormattable = flag5,
    fsVolDuplicatable = flag6
};

This information is returned by msgFSGetVolMetrics.

typedef struct FS_VOL_HEADER {
    FS_VOL_TYPE type;
    FS_VOL_FLAGS flags;
    OBJECT rootDir;
    OBJECT volObj;
    U32 serialNum;
    U32 created;
    U16 optimalSize;
    U32 totalBytes;
    U32 freeBytes;
    U32 commSpeed;
    U8 pName[nameBufLength];
    U8 alignSpare; // Word align following values
    CLASS browserClass; // Class of browser to use for volume
    U32 nativeFS;
    RES_ID iconResId;
    U32 spare1;
    U32 spare2;
    U32 spare3;
    U32 spare4;
} FS_VOL_HEADER, *P_FS_VOL_HEADER;

typedef FS_VOL_HEADER FS_VOL_METRICS, *P_FS_VOL_METRICS;

Enum16(FS_EXIST) {
    // Lower byte: what to do if the node exists
    fsExistOpen = 0,
    fsExistGenError = 1,
    fsExistGenUnique = 2,
    fsExistTruncate = 3,
    // Upper byte: what to do if the node doesn't exist
    fsNoExistCreate = MakeU16(0, 0),
    fsNoExistGenError = MakeU16(0, 1),
    fsNoExistCreateUnique = MakeU16(0, 2),
    // Default setting
    fsExistDefault = fsExistOpen | fsNoExistCreate
};

Enum16(FS_MOVE_COPY_EXIST) {
    // What to do if the destination node exists
    fsMoveCopyExistOverwrite = 0,
    fsMoveCopyExistGenError = 1,
    fsMoveCopyExistGenUnique = 2,
    fsMoveCopyExistDelete = 3,
    // Default setting
    fsMoveCopyExistDefault = fsMoveCopyExistGenError
};
Enum16(FS_DIR_NEW_MODE) {
    // Delete directory at handle free time?
    fsTempDir = flag0,
    // Is handle changeable?
    fsUnchangeable = flag1,
    // Find node via its dir index?
    fsUseDirIndex = flag2,
    // Disable prompts (insert disk, write protected, etc)
    // fsDisablePrompts = flag4, (Defined in FS_FILE_NEW_MODE below)
    // System owned dir handle - ring 0 only
    fsSystemDir = flag7,
    // Default setting
    fsDirNewDefaultMode = 0 // permanent, changeable directory
};

Enum16(FS_FILE_NEW_MODE) {
    // Lower byte: flags
    // Delete file at handle free time?
    fsTempFile = flag0,
    // Read/write intentions for this handle
    fsReadOnly = flag2,
    // Memory mapped files accessibility
    fsSharedMemoryMap = flag3,
    // Disable prompts (insert disk, write protected, etc)
    // fsDisablePrompts = flag4,
    // System owned file handle - ring 0 only
    fsSystemFile = flag4,
    // Upper byte: exclusivity requirements for other handles
    fsNoExclusivity = MakeU16(0, 0),
    fsDenyWriters = MakeU16(0, 1),
    fsExclusiveOnly = MakeU16(0, 2),
    // Default setting
    fsFileNewDefaultMode = 0 // perm, read/write (noExclusivity)
};

Enum16(FS_GET_PATH_MODE) {
    // Get path relative to root, dir passed in, just name or vol and path
    fsGetPathRoot = 0,
    fsGetPathRelative = 1,
    fsGetPathName = 2,
    fsGetPathAbsolute = 3,
    // Default setting
    fsGetPathDefaultMode = fsGetPathRoot
};

Enum16(FS_MOVE_COPY_MODE) {
    // Use destination as container.
    fsMoveCopyIntoDest = flag0,
    // Check but don't move or copy.
    fsMoveCopyVerifyOnly = flag1,
    // Does source have live dir indexes.
    fsMoveCopySourceArchived = flag2,
    // Does dest have live dir indexes.
    fsMoveCopyArchiveDest = flag3,
    // Default setting
    fsMoveCopyDefaultMode = 0
};

Enum16(FS_TRAVERSE_MODE) {
    // Call back on files?
    fsCallBackOnFiles = flag0,
    // Call back before stepping into directory?
    fsCallBackPreDir = flag1,
    // Call back after stepping into directory?
    fsCallBackPostDir = flag2,
    // Default setting
    fsTraverseDefaultMode = fsCallBackOnFiles | fsCallBackPreDir
};
Enum16(FS_SEEK_MODE) {
    // Relative to beginning of file, end of file, or Current Byte Position
    fsSeekBeginning = 0,
    fsSeekEnd = 1,
    fsSeekCurrent = 2,
    // Default setting
    fsSeekDefaultMode = fsSeekBeginning
};
typedef OBJECT DIR_HANDLE, * P_DIR_HANDLE;
typedef OBJECT FILE_HANDLE, * P_FILE_HANDLE;

Class File System Messages understood by dirHandles and fileHandles

msgFSGetInstalledVolumes

Returns list of all installed volumes.

Takes P_LIST, returns STATUS.

#define msgFSGetInstalledVolumes MakeMsg(clsFileSystem, 21)

Comments

This message can only be directed to the well known class theFileSystem. Each object in the list is a
directory handle object that references the root node of the volume. The list is passed back and is not
used as an input parameter. The caller must free the returned list when finished using it, but do not free
any of the objects in the list.

See Also

msgFSEjectMedia to eject media from a floppy drive.
msgFSGetVolMetrics to get more info about the volume
msgFSSame to compare root dir to a well-known dir handle

Class File System Messages understood by dirHandles and fileHandles

msgNew

Creates a directory or file handle object on a new or existing dir/file.

Takes P_FS_NEW, returns STATUS. Category: class message.

Arguments

typedef struct FS_NEW_ONLY {
    FS_LOCATOR locator; // location of the target directory
    FS_VOL_TYPE volType; // hint for uninstalled fullpath vols
    UUID dirIndex; // used with fsUseDirIndex mode only
    U16 mode; // options for opening file/dir handle
    FS_EXIST exist; // action to take if exists or doesn't
    P_UNKNOWN pVolSpecific; // volume specific information
    U32 spare1; // for future use
    U32 spare2; // for future use
    BOOLEAN alreadyExisted; // Out: indicates if already exists
} FS_NEW_ONLY, * P_FS_NEW_ONLY;
#define fsNewFields
objectNewFields \
FS_NEW_ONLY fs;
typedef struct FS_NEW {
    fsNewFields
} FS_NEW, * P_FS_NEW;
The fields you commonly set are:

- `pNew->fs.locator` Location of the node
- `pNew->fs.mode` Options for opening file/dir handle
- `pNew->fs.exist` Action to take if the file/dir exists or doesn't exist

Accessing a directory using a `dirIndex`: Three pieces of information must be provided to open a directory by `dirIndex`. The `fsUseDirIndex` flag must be set in `new.fs.mode`, a valid `dirIndex` must be supplied in `new.fs.dirIndex` and the volume that the directory resides on must be identified. This can be done by specifying some location on the volume by filling in `new.fs.locator`. Either the `uid` can point to the root or any other handle on the volume or the path can be an absolute path that identifies the volume. See `msgFSSetAttr` on how to store a dir index with a directory so it can later be accessed by its dir index.

Use `FS_DIR_NEW_MODE` for mode if `new` is for dir handle. Use `FS_FILE_NEW_MODE` for mode if `new` is for file handle.

**Return Value**

- `stsBadParam` `locator.uid` is not a valid object.
- `stsFSAccessDenied` Access cannot be granted because node is locked for exclusive access, read only access or write only access.
- `stsFSBadPath` `locator.pPath` is malformed or a specified dir node is in fact a file.
- `stsFSDirFull` There is no space in the dir for a new node.
- `stsFSDirIndexNotFound` There is not a `dirIndex` for the dir node.
- `stsFSNodeBusy` Node cannot be deleted/truncated because it is being access by another client.
- `stsFSNodeExists` The requested node already exists.
- `stsFSNodeNotFound` The root node does not exist.
- `stsFSNodeReadOnly` Node cannot be deleted/truncated or read/write access has been denied because the read only flag is set on the node.
- `stsFSNotDir` A requested dir node already exists as a file.
- `stsFSNotFile` A requested file node already exists as a dir.
- `stsFSTooManyHandles` There are already `fsMaxHandles` on this node.
- `stsFSUniqueFailed` `fsMaxUnique` variants of the name already exist.

**See Also**

- `FSNameValid`
- `msgNewDefaults`

**msgNewDefaults**

Initializes the `FS_NEW` structure to default values.

Takes `_P_FS_NEW`, returns `STATUS`. Category: class message.

**Message Arguments**

```c
typedef struct FS_NEW {
    fsNewFields
} FS_NEW, * _P_FS_NEW;
```

**Comments**

Zeroes out `pNew->fs` and sets:

```c
pNew->fs.locator.uid = theWorkingDir;
pNew->object.cap = objCapCall;
```
msgDestroy

Destroys a directory or file handle.

Takes OBJ_KEY, returns STATUS.

Comments

This destroys the handle, NOT the actual node. An exception to this is if the fsTempFile/fsTempDir flag was set in pNew->fs.mode when the handle was created.

Return Value

stsFSNodeBusy  Temporary node cannot be deleted because it is being access by another client.

stsFSNodeReadOnly  Temporary node cannot be deleted because the read only flag is set on the node.

msgFSNull

Does nothing.

Takes void, returns STATUS.

#define msgFSNull MakeMsg(clsFileSystem, 20)

This message is used to time entering and exiting the file system.

msgFSGetVolMetrics

Returns metrics of the volume.

Takes P_FS_GET_VOL_METRICS, returns STATUS.

#define msgFSGetVolMetrics MakeMsg(clsFileSystem, 22)

typedef struct FS_GET_VOL_METRICS {
    BOOLEAN updateInfo;  // have volume recompute values?
    FS_VOL_METRICS volMetrics;  // Out: the volume’s metrics
} FS_GET_VOL_METRICS, *P_FS_GET_VOL_METRICS;

Return Value

stsFSVolDisconnected  This will never be returned, even if the volume is disconnected. Instead test fSVolConnected in volMetrics.flags.

You must set updateInfo to TRUE if you want the volMetrics.freeBytes field or the fsVolConnected flag of the volMetrics.flags field to be updated before returning the vol metrics. Setting updateInfo to FALSE will make this request faster, but these fields may not be correct.

msgFSSetVolName

Changes the name of a volume.

Takes P_STRING, returns STATUS.

#define msgFSSetVolName MakeMsg(clsFileSystem, 36)

Return Value

stsBadParam  New vol name is invalid (checked by FSNameValid).

stsFSHandleInvalid  The dir/file object refers to a node that has been previously deleted.

stsFSVolDisconnected  The volume is not connected.

stsFSVolReadOnly  The new volume name cannot be set, because the volume is write protected.

See Also

FSNameValid  Mechanism to precheck validity of new volume name.
### msgFSNodeExists
Tests the existence of a file or directory node.

Takes P_FS_NODE_EXISTS, returns STATUS.

```c
#define msgFSNodeExists MakeMsg(clsFileSystem, 37)
```

#### Arguments
- typedef struct FS_NODE_EXISTS {
  P_STRING pPath;  // path to node that may exist
  BOOLEAN isDir;   // Out: dir or file
} FS_NODE_EXISTS, *P_FS_NODE_EXISTS;

#### Comments
The return parm isDir is useful in deciding whether the msgNew, to create a handle to the node, should be sent to clsDirHandle or clsFileHandle. The parm pPath is relative to the object that receives this message.

#### Return Value
- stsOK The node exists.
- stsFSNodeNotFound The node does not exist.

### msgFSGetHandleMode
Returns the "new" mode for the object's fs handle.

Takes P_U16, returns STATUS.

```c
#define msgFSGetHandleMode MakeMsg(clsFileSystem, 23)
```

#### Comments
Directory handles interpret the P_U16 as a P_FS_FILE_NEW_MODE. File handles interpret the P_U16 as a P_FS_DIR_NEW_MODE.

### msgFSSetHandleMode
Changes the "new" mode for the object's fs handle.

Takes P_FS_SET_HANDLE_MODE, returns STATUS.

```c
#define msgFSSetHandleMode MakeMsg(clsFileSystem, 24)
```

#### Arguments
- typedef struct FS_SET_HANDLE_MODE {
  U16 mode;  // value of mode flags to change
  U16 mask;  // which mode flags are to change
} FS_SET_HANDLE_MODE, *P_FS_SET_HANDLE_MODE;

#### Comments
Directory handles interpret mode as a FS_FILE_NEW_MODE. File handles interpret mode as a FS_DIR_NEW_MODE.

### msgFSSame
Tests if another directory or file handle references the same node.

Takes OBJECT, returns STATUS.

```c
#define msgFSSame MakeMsg(clsFileSystem, 25)
```

### msgFSGetPath
Gets the path to (or name of) a directory or file handle node.

Takes P_FS_GET_PATH, returns STATUS.

```c
#define msgFSGetPath MakeMsg(clsFileSystem, 26)
```
**FS.H**

Class File System Messages understood by dirHandles and fileHandles

### Arguments

```c
typedef struct FS_GET_PATH {
    FS_GET_PATH_MODE mode; // options for get path operation
    DIR_HANDLE dir; // In-Out: rel dir or root dir
    U16 bufLength; // length of pPathBuf
    P_STRING pPathBuf; // Out: user buffer for path
} FS_GET_PATH, *P_FS_GET_PATH;
```

### Comments

If mode is fsGetPathRoot or fsGetPathAbsolute the root dir handle is passed back in dir. If mode is fsGetPathRelative the path passed back begins at the dir represent by dir and terminates at the node represented by the recipient of this client.

### Return Value

- **stsFSBuffTooSmall** : User supplied pPathBuf is not large enough.
- **stsFSNotAncestor** : dir is not ancestor of recipient of msgFSGetPath.

---

**msgFSGetAttr**

Gets an attribute or attributes of a file or directory node.

Takes P_FS_GET_SET_ATTR, returns STATUS.

```c
#define msgFSGetAttr MakeMsg(clsFileSystem, 27)
```

### Arguments

```c
typedef struct FS_GET_SET_ATTR {
    P_STRING pPath; // path to node to get/set attrs
    U16 numAttrs; // number of attrs of interest
    P_FS_ATTR_LABEL pAttrLabels; // In-Out: attr labels
    P_FS_ATTR_VALUES pAttrValues; // In-Out: attr values
    P_FS_ATTR_SIZE pAttrSizes; // In-Out: attr sizes
} FS_GET_SET_ATTR, *P_FS_GET_SET_ATTR;
```

### Comments

Specify which attributes you wish returned via an array of attribute labels pointed to by pAttrLabels. The number of attribute labels is specified by numAttrs. The values are passed back via an array of values. If the nth value represents a string or variable attribute a pointer must be filled in for the destination of the string/variable. If the nth value represents a Fix64 provide space for two consecutive U32s. The sizes are passed back via an array of sizes.

If either the values are of no interest or the sizes are of no interest, set pAttrValues to pNull and/or set pAttrSizes to pNull.

If you want all attributes of a node, but do not know what they may be set numAttrs to maxU16, pAttrLabels to fsAllocAttrLabelsBuffer, and pAttrValues to fsAllocAttrValuesBuffer (or pNull if unwanted) and pAttrSizes to fsAllocAttrSizesBuffer (or pNull if unwanted). Any buffers returned as a result of fsAllocXXXBuffer must be freed with OSHepBlockFree.

The parm pPath is relative to the object that receives this message.

---

**msgFSSetAttr**

Sets the attribute or attributes of a file or directory node.

Takes P_FS_GET_SET_ATTR, returns STATUS.

```c
#define msgFSSetAttr MakeMsg(clsFileSystem, 28)
```

### Arguments

```c
typedef struct FS_GET_SET_ATTR {
    P_STRING pPath; // path to node to get/set attrs
    U16 numAttrs; // number of attrs of interest
    P_FS_ATTR_LABEL pAttrLabels; // In-Out: attr labels
    P_FS_ATTR_VALUES pAttrValues; // In-Out: attr values
    P_FS_ATTR_SIZE pAttrSizes; // In-Out: attr sizes
} FS_GET_SET_ATTR, *P_FS_GET_SET_ATTR;
```
Specify which attributes you wish to set via an array of attribute labels pointed to by pAttrLabels. The number of attribute labels is specified by numAttrs. The values are specified via an array of values. If the nth value represents a string or variable attribute supply the pointer to the string/variable. If the nth value represents a Fix64 attribute two consecutive U32 values are expected. If there are no variable length attributes, pAttrSizes can be set to pNull, because the size of Fix32, Fix64 and string attributes can be inferred.

pAttrLabels, pAttrValues & pAttrSizes are inputs only for this message. The parm pPath is relative to the object that receives this message.

The attr fsAttrDirIndex (dir indexes) can be set on directories to establish an alternate access to a directory without having to specify the path to the directory. See msgNew above on how to access directories with a dir index. Only directories that reside under the PenPoint tree (any directories below the PenPoint directory on a given volume) can have dir index attributes. If another directory already has the same dir index as the one given then a stsFSDirIndexExists error is returned.

NOTE: Most attributes (with the exception of dir index and old dir index attributes) can be stored with either files or directories. The root of a volume is the exception. No attributes may be stored with the root.

stsFSBadPath New name for name attr is invalid.
stsFSNotDir Dir index attr cannot be set on a file.
stsFSReadOnlyAttr File size cannot be set via set attr, use msgFSSetSize.

msgFSMove
Moves a node (and any children) to a new destination.

Takes P_FS_MOVE_COPY, returns STATUS.

```
#define msgFSMove MakeMsg(clsFileSystem, 29)
```

### Arguments

```c
typedef struct FS_MOVE_COPY {
    P_STRING pSourcePath; // path of source of move or copy
    FS_LOCATOR destLocator; // locator to destination node
    FS_MOVE_COPY_MODE mode; // options that affect move or copy
    FS_MOVE_COPY_EXIST exist; // action to take if exists or doesn't alreadyExist;
    P_STRING pNewDestName; // Out: See comment above
    BOOLEAN alreadyExisted; // Out: indicates if already exists
    U32 spare;
} FS_MOVE_COPY, *P_FS_MOVE_COPY;
```

### Comments

The destination file/dir name of a move is derived as follows.

For "fsMoveCopyToDest" (the default): If non null path is provided then dest name is the leaf name of the path and the path up to the leaf name determines the destination directory. If the path is null then the name of the destination object is used as the dest name and the parent of the destination object is used as the destination directory.

For fsMoveCopyIntoDest: The entire destination uid and path are used for the destination directory. And the destination name is taken from the source name.

The parm pSourcePath is relative to the object that receives this message.

NOTE: pNewDestName is not an in parameter. It is an output parameter that gives the (new, if fsMoveCopyGenUnique was specified for exist) name of the copied node. Set pNewDestName to a buffer if you want to know the name, set pNewDestName to pNull if you do not.
**msgFSMoveNotify**

Same as msgFSMove with notification routine extensions.

Takes P_FS_MOVE_COPY_NOTIFY, returns STATUS.

```c
#define msgFSMoveNotify MakeMsg(clsFileSystem, 70)
// the time that the current event occurred

Enum16 ( FS_NOTIFY_TIME ) {
    fsBeginOperation = 1,  // beginning of whole operation
    fsBeforeOperation = 2, // before the sub operation
    fsDuringOperation = 3, // during the sub operation
    fsAfterOperation = 4,  // after the sub operation
    fsEndOperation = 5     // end of the whole operation
};
```
// the operation of the current event
Enum16 ( FS_NOTIFY_OP ) {  
  fsReadOperation = 1,    // read operation
  fsWriteOperation = 2,   // write operation
  fsCreateOperation = 3,  // create operation
  fsVerifyOperation = 4,  // verify operation
  fsDeleteOperation = 5   // delete operation
};

// information required by the notification routine
typedef struct FS_NOTIFY_RTN_INFO {
  OBJECT source;          // a handle to the current file
  BOOLEAN moveOperation;  // if move operation
  BOOLEAN isADirectory;   // if source is a directory
  P_FS_GET_SET_ATTR pFSGetSetAttr; // attributes for current file
  FS_NOTIFY_TIME fsNotifyTime; // time context of notification
  FS_NOTIFY_OP fsNotifyOp; // op context of notification
  U32 bufferSize;         // max size of operation buffer
  U32 operationSize;      // actual size of operation
  U32 fileSize;           // actual size of file
  U32 spare1;             // spare: unused
  U32 spare2;             // spare: unused
} FS_NOTIFY_RTN_INFO, *P_FS_NOTIFY_RTN_INFO;

// the definition of the notification routine
typedef STATUS FunctionPtr ( P_FS_NOTIFY_RTN ) ( P_FS_NOTIFY_RTN_INFO pFSNotifyRtnInfo, P_UNKNOWN pClientData );

// the information required for FSMove/CopyNotify
typedef struct FS_MOVE_COPY_NOTIFY {
  P_STRING pSourcePath;    // path of source of move or copy
  FS_LOCATOR destLocator;  // locator to destination node
  FS_MOVE_COPY_MODE mode;  // options that affect move or copy
  FS_MOVE_COPY_EXIST exist; // action to take if exists or doesn’t
  P_STRING pNewDestName;   // Out: see comment w/msgFSMove
  BOOLEAN alreadyExisting; // Out: indicates if already exists
  P_UNKNOWN pNotifyRtn;    // notification routine
  P_UNKNOWN pClientData;   // client data to notification routine
  P_UNKNOWN pQuickSortRtn; // quicksort routine
  U32 spare1;              // spare: unused
  U32 spare2;              // spare: unused
} FS_MOVE_COPY_NOTIFY, *P_FS_MOVE_COPY_NOTIFY;

Comments
The parm pSourcePath is relative to the object that receives this message.

msgFS_CopyNotify
Same as msgFS_Copy with notification routine extensions.

Takes P_FS_MOVE_COPY_NOTIFY, returns STATUS.

#define msgFS_CopyNotify MakeMsg(clsFileSystem, 71)

Message Arguments
typedef struct FS_MOVE_COPY_NOTIFY {
  P_STRING pSourcePath;    // path of source of move or copy
  FS_LOCATOR destLocator;  // locator to destination node
  FS_MOVE_COPY_MODE mode;  // options that affect move or copy
  FS_MOVE_COPY_EXIST exist; // action to take if exists or doesn’t
  P_STRING pNewDestName;   // Out: see comment w/msgFSMove
  BOOLEAN alreadyExisting; // Out: indicates if already exists
  P_UNKNOWN pNotifyRtn;    // notification routine
  P_UNKNOWN pClientData;   // client data to notification routine
  P_UNKNOWN pQuickSortRtn; // quicksort routine
  U32 spare1;              // spare: unused
  U32 spare2;              // spare: unused
} FS_MOVE_COPY_NOTIFY, *P_FS_MOVE_COPY_NOTIFY;

Comments
The parm pSourcePath is relative to the object that receives this message.
msgFSDelete
Deletes a node (and all of its children).
Takes P_STRING, returns STATUS.
#define msgFSDelete MakeMsg(clsFileSystem, 31)

Comments
The object of msgFSDelete is typically a dir handle, but it can also be a file handle, but the argument passed must be set to pNull. After a node is deleted, its handle is marked corrupt (since it is no longer valid). A dir handle object can be reused via msgFSSetTarget or destroyed via msgDestroy. A file handle must be destroyed after the node is deleted. The argument (a path) is relative to the object that receives this message.

Return Value
stsFSVolDisconnected The volume is not connected.
stsFSVolReadOnly A node cannot be deleted, because the volume is write protected.
stsFSNodeReadOnly Node cannot be deleted because the read only flag is set on the node.
stsFSNodeBusy Node cannot be deleted because it is being access by another client.

See Also
msgFSForceDelete

msgFSFlush
Flushes any buffers and attributes associated with the file or directory.
Takes void, returns STATUS.
#define msgFSFlush MakeMsg(clsFileHandle, 20)

Comments
This can be used to guarantee that cached buffers are flushed to the disk and can also be used to flush memory mapped files to disk.

msgFSMakeNative
Removes anything not supported by the native file system.
Takes P_FS_MAKE_NATIVE, returns STATUS.
#define msgFSMakeNative MakeMsg(clsFileSystem, 32)

typedef struct FS_MAKE_NATIVE {
    P_STRING pPath;            // path to node to make native
    P_STRING pNewName;         // Out: native name if changed
} FS_MAKE_NATIVE, *P_FS_MAKE_NATIVE;

Arguments

Comments
The parm pPath is relative to the object that receives this message.

msgFSEjectMedia
Ejects media from an ejectable, removable volume.
Takes void, returns STATUS.
#define msgFSEjectMedia MakeMsg(clsFileSystem, 34)

Return Value
stsOK The volume media has been ejected.
stsFSVolDisconnected The volume media is already ejected.
stsRequestNotSupported The volume does not have ejectable media
## msgFSForceDelete
Forcibly deletes a node (and all of its children).
Takes P_STRING, returns STATUS.

```c
#define msgFSForceDelete MakeMsg(clsFileSystem, 35)
```

### Arguments
- `P_STRING` `pPath`; // path of node to receive msg
- `MESSAGE` `msg`; // message to pass on to volume
- `P_UNKNOWN` `pArgs`; // In-Out: message specific args

### Return Value
Volume specific errors.

### Comments
WARNING. Normal restrictions do not apply. The node will still be deleted even if it is being accessed via another handle or if it is marked read only. However, if the volume is not connected or is write protected, the forced delete will still fail.

After a node is deleted, its handle is marked corrupt (since it is no longer valid). A dir handle object can be reused via msgFSSetTarget or destroyed via msgDestroy. A file handle must be destroyed after the node is deleted. The argument (a path) is relative to the object that receives this message.

### See Also
msgFSDelete

## msgFSVolSpecific
Sends a volume specific message via a dir or file handle.
Takes P_FS_VOL_SPECIFIC, returns STATUS.

```c
#define msgFSVolSpecific MakeMsg(clsFileSystem, 40)
```

### Arguments
- `typedef struct FS_VOL_SPECIFIC {
  P_STRING `pPath`; // path of node to receive msg
  MESSAGE `msg`; // message to pass on to volume
  P_UNKNOWN `pArgs`; // In-Out: message specific args
} FS_VOL_SPECIFIC, *P_FS_VOL_SPECIFIC;

### See Also
msgFSChanged

## msgFSChanged
Notifies observers of directory changes.
Takes P_FS_CHANGE_INFO, returns STATUS. Category: observer notification.

```c
#define msgFSChanged MakeMsg(clsFileSystem, 50)
```

### Arguments
- `typedef struct FS_CHANGE_INFO {
  MESSAGE `reason`; // fs message that caused the change
  OBJECT `observed`; // observed dir whose content changed
  U32 `sparel`;
  U32 `sparel2`;
} FS_CHANGE_INFO, *P_FS_CHANGE_INFO;

### Comments
These messages are the reason observers of a dir handle would be notified of a change and the circumstances that the change happens:

- **msgInit** A file or dir has been created.
- **msgFree** A temp file or temp directory has been deleted.
- **msgFSDelete** A file or directory has been deleted.
- **msgFSMove** A file or directory has been "fast" moved.

This notifies observers of directories (not files) when a file or dir within the directory changes. The change reasons described below are changes to the directory or file node, not the handle referencing the node.
msgFSVolChanged

Notifies observer of volume changes.
Takes P_FS_VOL_CHANGE_INFO, returns STATUS. Category: observer notification.

```
#define msgFSVolChanged MakeMsg(clsFileSystem, 51)
```

Arguments

```
Enum16(FS_VOL_CHANGE_FLAGS) {
    fsVolChangeWhilePrompting = flag0   // FS prompting caused change
};
```

typedef struct FS_VOL_CHANGE_INFO {
    MESSAGE reason;                      // fs message that caused the change
    OBJECT rootDir;                     // root dh of volume that changed
    FS_VOL_CHANGE_FLAGS flags;          // more info related to reason
    U16 _spare1;
    U32 _spare2;
} FS_VOL_CHANGE_INFO, *P_FS_VOL_CHANGE_INFO;

These messages are the reason observers of the FileSystem would be notified of a volume addition, removal or change of state. Note: msgFSSetVolName (defined above) is also a volume change reason.

```
#define msgFSInstallVol MakeMsg(clsFileSystem, 1)
#define msgFSRemoveVol MakeMsg(clsFileSystem, 2)
#define msgFSConnectVol MakeMsg(clsFileSystem, 3)
#define msgFSDisconnectVol MakeMsg(clsFileSystem, 4)
```

Observe the well known object, the FileSystem, if you want to receive this.

---

## Class DirHandle Messages

---

msgFSSetTarget

Changes the target directory to directory specified by locator.
Takes P_FS_LOCATOR, returns STATUS.

```
#define msgFSSetTarget MakeMsg(clsDirHandle, 20)
```

Message

typedef struct FS_LOCATOR {
    OBJECT uid;
    P_STRING pPath;                      // Relative to node defined by uid
} FS_LOCATOR, *P_FS_LOCATOR;

Comments

Setting a dir handle object to a new target also resets the read dir pointer.

Return Value

stsFSUnchangeable The recipient of this message has been "opened" with the fsUnchangeable flag set in pNew->mode.

---

msgFSReadDir

Reads the next entry (its attributes) from a directory.
Takes P_FS_READ_DIR, returns STATUS.

```
#define msgFSReadDir MakeMsg(clsDirHandle, 21)
```

Arguments

typedef struct FS_READ_DIR {
    struct FS_READ_DIR *pNext;           // Out: only used w/msgFSReadDirFull
    U16 numAttrs;                        // In-Out: attrs of interest
    P_FS_ATTR_LABEL pAttrLabels;         // In-Out: ptr to attr labels
    P_UNKNOWN pAttrValues;               // In-Out: ptr to attr values
    P_FS_ATTR_SIZE pAttrSizes;           // In-Out: ptr to attr sizes
} FS_READ_DIR, *P_FS_READ_DIR;
Argumolts
Specify which attributes you wish returned via an array of attribute labels pointed to by `pAttrLabels`. The number of attribute labels is specified by `numAttrs`. See `msgFSGetAttr` for a description on setting `pAttrValues` and `pAttrSizes`.

### msgFSReadDirReset

Resets the ReadDir position to the beginning.

Takes void, returns `STATUS`.

```c
#define msgFSReadDirReset MakeMsg(clsDirHandle, 22)
```

**Comments**
This will direct `msgFSReadDir` to begin reading from the first entry in the directory. This has no effect on `msgFSReadDirFull`. The default after creating a handle to a directory is to point to the first entry.

### msgFSReadDirFull

Reads all the entries in a directory into a local buffer.

Takes `P_FS_READ_DIR_FULL`, returns `STATUS`.

```c
#define msgFSReadDirFull MakeMsg(clsDirHandle, 23)
```

**Arguments**

```c
typedef struct FS_READ_DIR FULL {
  U16 numAttrs; \n  // num of labels in label array
  P_FS_ATTR_LABEL pAttrLabels; \n  // attrs of interest to be read
  U32 numEntries; \n  // Out: number of dir entries
  U32 bufLength; \n  // Out: length of pDirBuf
  P_FS_READ_DIR pDirBuf; \n  // Out: points to first entry
} FS_READ_DIR FULL, *P_FS_READ_DIR FULL;
```

**Comments**
Specify which attributes you wish returned via an array of attribute labels pointed to by `pAttrLabels`. The number of attribute labels is specified by `numAttrs`.

The returned data is a linked list of `FS_READ_DIR` entries, linked by the `pNext` field. The last link is specified by a `pLink == pNull`.

The client must free the returned buffer `pDirBuf`, using `OSHeapBlockFree`. The buffer should not be freed if it has a value of `pNull`, which will be the case if there are any errors or if `numEntries` is zero.

### msgFSTraverse

Traverse through the nodes of a tree starting with the target of this msg.

Takes `P_FS_TRAVERSE`, returns `STATUS`.

```c
#define msgFSTraverse MakeMsg(clsDirHandle, 24)
```

**Function Prototype**

```c
typedef STATUS FunctionPtr(P_FS_TRAVERSE_CALL_BACK) ( 
  OBJECT dir, \n  // dir handle to current node
  U16 level, \n  // level in the hierarchy
  P_FS_READ_DIR pNextEntry, \n  // info about next entry
  P_UNKNOWN pClientData \n  // the client’s data
);
```

```c
typedef struct FS_TRAVERSE {
  FS_TRAVERSE_MODE mode; \n  // call back order and criteria
  U16 numAttrs; \n  // num of labels in label array
  P_FS_ATTR_LABEL pAttrLabels; \n  // attr label array
  P_FS_TRAVERSE_CALL_BACK pCallBackRtn; \n  // called for each dir & file
  P_UNKNOWN pClientData; \n  // passed to call back routine
  P_UNKNOWN pQuickSortRtn; \n  // optional quick sort routine
} FS_TRAVERSE, *P_FS_TRAVERSE;
```
This message traverses the file system tree beginning with the directory which is the recipient of this message and traverses the node tree depth first. The client will be called back via pCallBackRtn at each node depending on mode (see FS_TRAVERSE_MODE above). Optionally, the nodes at each directory level can be sorted before being returned by specifying a quick sort routine via pQuickSortRtn (See quicksort in sort.h).

Specify which attributes you wish returned via an array of attribute labels pointed to by pAttrLabels. The number of attribute labels is specified by numAttrs. At a minimum, pAttrLabels must contain fsAttrName and fsAttrFlags.

- **stsBadParam** Did not specify fsAttrName/fsAttrFlags in labels.
- **stsFSUnchangeable** The recipient of this message has been "opened" with the fsUnchangeable flag set in pNew->mode. This is a common error if trying to traverse from the root dir (which is unchangeable) provided by msgFSGetInstalledVolumes/msgFSGetVolMetrics. Create a handle to the root and use that to traverse instead.
- **stsFSNestingTooDeep** Dir tree is deeper than fsMaxNestingLevel levels.

Prototype for the call back routine used by msgFSTraverseTree

### Class FileHandle Messages

#### msgStreamRead
Reads data from the file.

* Takes P_STREAM_READ_WRITE, returns STATUS. Category: descendant responsibility.

* **Comments**
  
  The maximum number of bytes read with a single request is determined by fsMaxReadWrite.

* **Return Value**
  
  - **stsBadParam** Requesting more than fsMaxReadWrite bytes.

* **See Also**
  
  msgStreamRead in stream.h

#### msgStreamWrite
Writes data to the file.

* Takes P_STREAM_READ_WRITE, returns STATUS. Category: descendant responsibility.

* **Comments**
  
  The maximum number of bytes writable with a single request is determined by fsMaxReadWrite. Note that writes to a memory mapped file that cause the file to grow will result in a stsFSNodeBusy error. Free the memory map file pointer before growing the file.

* **Return Value**
  
  - **stsBadParam** Requesting more than fsMaxReadWrite bytes.
  
  - **stsFSNodeReadOnly** This is a read only file.
  
  - **stsFSVolFull** The file could not be written - no space on volume.
  
  - **stsFSNodeBusy** The file is memory mapped and this write request would cause the file to be grown beyond the memory mapped size.

* **See Also**
  
  msgStreamWrite in stream.h
msgStreamFlush
Flushes any buffers associated with the file.
Takes void, returns STATUS. Category: descendant responsibility.

See Also
msgStreamFlush in stream.h

msgStreamSeek
Seeks to new position within the file.
Takes P_STREAM_SEEK, returns STATUS. Category: descendant responsibility.

Return Value
stsBadParam Seek mode is out of range.

See Also
msgStreamSeek in stream.h

msgFSSeek
Sets the value of the current byte position.
Takes P_FS_SEEK, returns STATUS.

#define msgFSSeek

typedef struct FS_SEEK {
    FS_SEEK_MODE mode; // seek from bof, cur pos, eof
    S32 offset; // relative change from seek origin
    U32 curPos; // Out: cur byte pos after seek
    U32 oldPos; // Out: cur byte pos before seek
    BOOLEAN eof; // Out: Is new pos at end of file?
} FS_SEEK, * P_FS_SEEK;

Return Value
stsBadParam Seek mode is out of range.

msgFSGetSize
Gets the size of the file.
Takes P_FS_FILE_SIZE, returns STATUS.

#define msgFSGetSize

msgFSSetSize
Sets the size of the file.
Takes P_FS_SET_SIZE, returns STATUS.

#define msgFSSetSize

typedef struct FS_SET_SIZE {
    FS_FILE_SIZE newSize; // new file size
    FS_FILE_SIZE oldSize; // Out: prior file size
} FS_SET_SIZE, * P_FS_SET_SIZE;

Comments
Note that a set size to a memory mapped file that causes the file to grow will result in a stsFSNodeBusy error. Free the memory map file pointer before growing the file.

Return Value
stsFSNodeReadOnly This is a read only file.
stsFSVolFull The file could not be grown - no space on volume.
stsFSNodeBusy The file is memory mapped and this set size request would cause the file to be grown beyond the memory mapped size.
**Public Functions**

**msgFSMemoryMap**
Associates the file with a directly accessible memory pointer.
Takes PP_MEM, returns STATUS.

```c
#define msgFSMemoryMap MakeMsg(clsFileHandle, 24)
```

Comments
To get a memory mapped file pointer from shared memory, the file handle must be created with `pNew->fs.mode |= fsSharedMemoryMap`.

**msgFSMemoryMapFree**
Frees the memory map pointer currently associated with the file.
Takes void, returns STATUS.

```c
#define msgFSMemoryMapFree MakeMsg(clsFileHandle, 25)
```

Comments
NOTE: Memory map pointers are freed for you at `msgFree` of a file handle.

**msgFSMemoryMapSetSize**
Sets the size of the file's memory map.
Takes SIZEOF, returns STATUS.

```c
#define msgFSMemoryMapSetSize MakeMsg(clsFileHandle, 26)
```

Comments
Determines the limit of a memory map for the file. The size can’t be less than the file size, nor less than a limit set by another client but can be larger. The memory map size must be set before memory mapping the file.

Return Value
`stsFSNodeBusy`  The file is currently memory mapped.

**msgFSMemoryMapGetSize**
Gets the size of the file's memory map.
Takes P_SIZEOF, returns STATUS.

```c
#define msgFSMemoryMapGetSize MakeMsg(clsFileHandle, 27)
```

**FSNameValid**
Checks a file/dir name for validity.
Returns STATUS.

Function Prototype

```c
STATUS EXPORTED FSNameValid(
    P_STRING pName
) ;
```

Return Value
`stsOK`  The node name is valid.
`stsFailed`  The node name was invalid.

Name is bad if it has no characters, is greater than 32 characters, has leading or trailing spaces, contains the pathname delimiter char, contains the file system escape character, or is the name of self (.) or parent (..).
FSUTIL.H

This file contains filesystem attribute helper procedures. The functions described in this file are contained in SYSUTIL.LIB.

These procedures make it easier to deal with filesystem attributes. They also support list attributes; variable attributes which maintains lists of 4-byte quantities.

```c
#ifndef FSUTIL_INCLUDED
#define FSUTIL_INCLUDED
#endif
#ifndef FS_INCLUDED
#include <fs.h>
#endif

GetNodeName

Gets the name attribute of a given filesystem node.

Returns STATUS.

Function Prototype: STATUS EXPORTED GetNodeName(
    OBJECT handle, // File or dir handle.
    P_STRING pName); // Out: name.

Comments: Use this function to easily get the name of a node.

GetAttr

Gets a single FIX32 attribute from a filesystem handle.

Returns STATUS.

Function Prototype: STATUS EXPORTED GetAttr(
    FS_ATTR_LABEL attrLabel, // Attribute label.
    OBJECT handle, // File or dir handle.
    P_U32 pValue); // Out: attribute value.

Comments: This is only for FIX32 attributes when you have a handle onto the node; see GetSingleAttr for a more general function.

GetSingleAttr

Gets a single FIX32, FIX64, or known-size STRING attribute.

Returns STATUS.

Function Prototype: STATUS EXPORTED GetSingleAttr(
    FS_ATTR_LABEL attrLabel, // In: Attribute label.
    OBJECT handle, // In: handle of node.
    P_STRING pPath, // In: path of node.
    P_UNKNOWN pValue); // Out: attribute value.
SetAttr
Sets a single FIX32 attribute on a filesystem handle.

Returns STATUS.

Function Prototype
STATUS EXPORTED SetAttr(
    FS_ATTR_LABEL attrLabel,   // Attribute label.
    OBJECT handle,            // File or dir handle.
    U32 value);               // Attribute value.

Comments
This is only for FIX32 attributes when you have a handle onto the node; see SetSingleAttr for a more general function.

SetSingleAttr
Sets a single FIX32, FIX64, or STRING attribute.

Returns STATUS.

Function Prototype
STATUS EXPORTED SetSingleAttr(
    FS_ATTR_LABEL attrLabel,   // In: Attribute label.
    OBJECT handle,            // In: handle of node.
    P_STRING pPath,           // In: path of node.
    P_UNKNOWN pValue);        // In: attribute value.

GetListX
Gets a VAR attribute that is organized as a list of values.

Returns STATUS.

Function Prototype
STATUS EXPORTED GetListX(
    OBJECT handle,            // File or dir handle.
    P_STRING pPath,           // Path relative to handle.
    FS_ATTR_LABEL attrLabel,  // Attribute label.
    PP UNKNOWN ppList,        // Out: list.
    P_U16 pSize);             // Out: size (in bytes) of list.

Comments
Allocates ppList from the process local stack. Caller must HeapBlockFree ppList when done adding, removing, and putting the list.

PutListX
Updates a list attribute with a new list.

Returns STATUS.

Function Prototype
STATUS EXPORTED PutListX(
    OBJECT handle,            // File or dir handle.
    P_STRING pPath,           // Path relative to handle.
    FS_ATTR_LABEL attrLabel,  // Attribute label.
    P UNKNOWN pList,          // List.
    U16 size);                // Size (in bytes) of list.
FindListItemX
Finds an element in a list.
Returns STATUS.

Function Prototype
STATUS EXPORTED FindListItemX(
    pItem, // Data to search for.
    itemSize, // Size of data to search for.
    pList, // List.
    listSize, // Size of list.
    pOffset); // Out: offset of found item.

Comments
The list must first be gotten via GetList. pOffset is 0 based. The list array can be indexed with pOffset to get the actual data. The comparison is done via a memcmp, so things must be EXACTLY the same.

Return Value
stsNoMatch  Item not found.

AddListItemX
Adds an item to the end of a list.
Returns STATUS.

Function Prototype
STATUS EXPORTED AddListItemX(
    pItem, // Item to add.
    itemSize, // Size of item in bytes.
    pList, // In:Out List.
    pSize); // In:Out size of list in bytes.

Comments
The list must first be gotten via GetList. The heap that the list uses is resized. pSize is updated to reflect the new list size.

RemoveListItemX
Removes an item from a list, given an offset.
Returns STATUS.

Function Prototype
STATUS EXPORTED RemoveListItemX(
    offset, // Offset of item to remove.
    size, // Size of item to remove.
    pList, // In:Out List.
    pSize); // In:Out Size of list.

Comments
The list must first be gotten via GetList. The heap that the list uses is resized. If pSize == 1 (only 1 item left) then *pSize is set to 0, but the list heap is not resized. offset is 0-based.

Private
Below are the "old" attribute list functions. These are here for backwards compatibility only!

GetList
Gets a VAR attribute that is organized as a list of 4 byte values.
Returns STATUS.

Function Prototype
STATUS EXPORTED GetList(
    handle, // File or dir handle.
    pPath, // Path relative to handle.
    attrLabel, // Attribute label.
    pList, // Out: list.
    pCount); // Out: number of elements.
Allocates ppList from the process local stack. Caller must HeapBlockFree ppList when done adding, removing, and putting the list.

**PutList**

Updates a list attribute with a new list.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED PutList(
    OBJECT handle,   // File or dir handle.
    P_STRING pPath,  // Path relative to handle.
    FS_ATTRIB_LABEL attrLabel, // Attribute label.
    P_OBJECT pList,  // List.
    U16 count);      // Number of elements.
```

**FindListItem**

Finds an element in a list.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED FindListItem(
    OBJECT item,     // Data to search for.
    P_OBJECT pList,  // List.
    U16 count,       // Number of elements in list.
    P_U16 pIndex);  // Out: index of found item.
```

Comments

The list must first be gotten via GetList. pIndex is 0 based. The list array can be indexed with pIndex to get the actual data.

Return Value

stsNoMatch  Item not found.

**AddListItem**

Adds an item to the end of a list.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED AddListItem(
    OBJECT item,     // Item to add.
    PP_OBJECT ppList, // In:Out List.
    P_U16 pCount);   // In:Out number of elements in list.
```

Comments

The list must first be gotten via GetList. The heap that the list uses is resized. pCount is updated to reflect the new list size.

**RemoveListItem**

Removes an item from a list, given an index.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED RemoveListItem(
    U16 index,       // Index of item to remove.
    PP_OBJECT ppList, // In:Out List.
    P_U16 pCount);   // In:Out Number of elements in list.
```

Comments

The list must first be gotten via GetList. The heap that the list uses is resized. If pCount == 1 (only 1 item left) then *pCount is set to 0, but the list heap is not resized. index is 0-based.
This file contains the API definition for clsStream.

clsStream inherits from clsObject.

clsStream is an abstract class -- it does not completely implement its own protocol. Subclasses of clsStream must complete the implementation. clsFileHandle is an important subclass of clsStream (see fs.h).

The functions described in this file are contained in PENPOINT.LIB.

```c
#ifndef STREAMINCLUDED
#define STREAMINCLUDED
#else
#endif

#ifndef GOINCLUDED
#include <go.h>
#endif

#ifndef UIDINCLUDED
#include <uid.h>
#endif

#ifndef OSTYPESINCLUDED
#include <ostypes.h>
#endif

#ifndef CLSMGRINCLUDED
#include <clsmgr.h>
#endif
```

Common #defines and typedefs

```c
#define streamNewFields 
objectNewFields

typedef struct STREAM_NEW {
    streamNewFields
} STREAM_NEW, *P_STREAM_NEW;
```

Several types in this file contain "streamElements."

The streamElements fields are:

- numBytes: In: size of buffer
- pBuf: In: buffer
- count: Out: number of bytes transferred

```c
#define streamElements 
U32 numBytes; 
P_UNKNOWN pBuf; 
U32 count;
```

Status codes

```c
#define ststimeOutWithData MakeWarning(clsStream, 1)
```

stsStreamDisconnected status is returned by all stream calls when the service executing the stream function is no longer in a connected state (A disconnectable service is clsMILAsyncSIO).

Clients must not send other stream messages to the disconnected service.
Penpoint can notify clients or clients may find services’ connected states (see service.h and servmgr.h).

```
#define stsStreamDisconnected MakeStatus(clsStream, 1)
```

### Messages

**msgStreamRead**

Reads data from stream.

Takes P_STREAM_READ_WRITE, returns STATUS. Category: descendant responsibility.

```
#define msgStreamRead MakeMsg(clsStream, 1)
```

**Arguments**

```c
typedef struct {
    streamElements
} STREAM_READ_WRITE, * P_STREAM_READ_WRITE;
```

**Comments**

`msgStreamRead` reads data from the stream into `pBuf`. `pBuf` must point to a buffer which can hold at least `numBytes` bytes. The number of bytes read is passed back in `count`.

If you try to read 0 bytes when at the end of the data stream `stsOK` is returned.

**Return Value**

- `< stsOK`  No data read.
- `>= stsOK`  Count of bytes is non-zero.
- `stsEndOfData`  Count is zero and at the end of data.

**msgStreamWrite**

Writes data to stream.

Takes P_STREAM_READ_WRITE, returns STATUS. Category: descendant responsibility.

```
#define msgStreamWrite MakeMsg(clsStream, 2)
```

**Arguments**

```c
typedef struct {
    streamElements
} STREAM_READ_WRITE, * P_STREAM_READ_WRITE;
```

**Comments**

`msgStreamWrite` writes `numBytes` from `pBuf` into the stream. Returns `stsOK` if all bytes are written.

**msgStreamReadTimeOut**

Reads data from stream with timeout.

Takes P_STREAM_READ_WRITE_TIMEOUT, returns STATUS. Category: descendant responsibility.

```
#define msgStreamReadTimeOut MakeMsg(clsStream, 3)
```

**Arguments**

```c
typedef struct {
    streamElements
    osMilliseconds timeout;
} STREAM_READ_WRITE_TIMEOUT, * P_STREAM_READ_WRITE_TIMEOUT;
```

**Comments**

`msgStreamReadTimeOut` reads data from the stream into `pBuf`. `pBuf` must point to a buffer which can hold at least `numBytes` bytes. The number of bytes read is passed back in `count`.

When count is greater than zero the status returned is always greater than or equal to `stsOK`.

**Return Value**

- `stsTimeOutWithData`  Count is greater than zero but less than `numBytes` because of a timeout.
- `stsTimeOut`  Count is zero and the timeout has expired.
- `stsEndOfData`  Count is zero and at the end of data.
msgStreamWriteTimeOut

Writes to the stream with timeout.

Takes P_STREAM_READ_WRITE_TIMEOUT, returns STATUS. Category: descendant responsibility.

#define msgStreamWriteTimeOut MakeMsg(clsStream,4)

typedef struct {
    streamElements
    OSMilliseconds timeOut;  // In: milliseconds until timeout
} STREAM_READ_WRITE_TIMEOUT, * P_STREAM_READ_WRITE_TIMEOUT;

Comments
Writes numBytes from pBuf into the stream.

Return Value
stsOK All bytes were written.
stsTimeOut Timeout has expired before all data written.

msgStreamFlush

The stream flushes any buffered data.

Takes pNull, returns STATUS. Category: descendant responsibility.

#define msgStreamFlush MakeMsg(clsStream,5)

Comments
clsStream's default response is to return stsMessageIgnored. Most subclasses override clsStream's response.

Return Value
stsOK Buffers were successfully emptied.
stsFailed Buffers do not empty after some timeout period.

msgStreamSeek

Sets the stream's Current Byte Position.

Takes P_STREAM_SEEK, returns STATUS.

#define msgStreamSeek MakeMsg(clsStream,6)

Arguments
Enum16 (STREAM_SEEK_MODE) {
    // Relative to beginning of file, end of file, or Current Byte Position
    streamSeekBeginning = 0,
    streamSeekEnd = 1,
    streamSeekCurrent = 2,
    // Default setting
    streamSeekDefaultMode = streamSeekBeginning
};

typedef struct STREAM_SEEK {
    STREAM_SEEK_MODE mode;  // relative change from seek origin
    S32 offset;  // Out: byte position after seek
    U32 curPos;  // Out: byte position before seek
    U32 oldPos;  // Out: Is new pos at end of file?
    BOOLEAN eof;
} STREAM_SEEK, * P_STREAM_SEEK;

Comments
clsStream's default response is to return stsMessageIgnored. Most subclasses override clsStream's response.
**msgStreamBlockSize**

Passes back the most efficient write block size for this stream.

Takes P_STREAM_BLOCK_SIZE, returns STATUS. Category: descendant responsibility.

```c
#define msgStreamBlockSize MakeMsg(clsStream,7)
```

**Arguments**

typedef struct {
    U32 blockSize;  // out: preferred write block size
} STREAM_BLOCK_SIZE, *P_STREAM_BLOCK_SIZE;

**Comments**

clsStream's default response is to return a blockSize of 512. Most subclasses override clsStream's response.

---

**Functions**

The P_UNKNOWN declarations for the following are assumed to be FILE*. Maintaining a clean separation between ANSI and PenPoint header files prevents the use of the true type.

**StdioStreamBind**

Returns a stdio file pointer bound to a stream object.

Returns pointer to FILE.

**Function Prototype**

```c
int EXPORTED StdioStreamBind(
    OBJECT obj);
```

**StdioStreamUnbind**

Frees the stdio file handle bound to a stream object.

Returns int.

**Function Prototype**

```c
int EXPORTED StdioStreamUnbind(
    P_UNKNOWN pFile);
```

**StdioStreamToObject**

Returns the stream object bound to a stdio file pointer.

Returns OBJECT.

**Function Prototype**

```c
OBJECT EXPORTED StdioStreamToObject(
    P_UNKNOWN pFile);
```
This file contains the API for UUID routines. The functions described in this file are contained in PENPOINT.LIB.

This file contains macros for creating and testing Nil and Invalid UUIDs, to compare two UUIDs for equality, and to create a well known UUID and a function to create dynamic uuids.

UUID is an acronym for Universal Unique ID.

```c
#ifndef UUID_INCLUDED
#define UUID_INCLUDED

Include files

#ifndef GO_INCLUDED
#include <go.h>
#endif

Common #defines and typedefs

Macros

For setting and testing for a Nil UUID

#define MakeNilUUID(uuid) ((uuid).machine = (uuid).id = 0L)
#define NilUUID(uuid) (((uuid).machine == 0L) && ((uuid).id == 0L))

For setting and testing for an invalid UUID

#define MakeInvalidUUID(uuid) ((uuid).id = (uuid).machine = maxU32)
#define InvalidUUID(uuid) ((uuid).id == maxU32 &&
                           (uuid).machine == maxU32)

To compare two UUIDs for equality

#define SameUUIDs(a,b) (((a).machine == (b).machine) &&
                       ((a).id == (b).id))

To set the fields of a well known uuid

#define MakeWknUUID(uuid,tag,i) 
   ((uuid).machine = (tag), (uuid).id = (U32)(i))

Typedefs

typedef struct UUID {
   U32 id;           // Unique counting value
   U32 machine;      // Unique machine identifier
} UUID, *P_UUID;
```
Public Functions

MakeDynUUID

Creates a dynamic UUID.

Returns nothing.

Function Prototype

```c
void EXPORTED MakeDynUUID ( P_UUID pUUID );
```
VOL.H

clsVolume inherits from clsObject.

Provides volume support.

Information in this file is useful if you are writing an installable volume. Also see volgodir.h for additional information.

```c
#ifndef VOL_INCLUDED
#define VOL_INCLUDED

Include file dependencies

#ifndef GO_INCLUDED
#include <go.h>
#endif

#ifndef OS_INCLUDED
#include <os.h>
#endif

#ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif

#ifndef FS_INCLUDED
#include <fs.h>
#endif

#endif
```

**Common #defines and typedefs**

**Defines**

```c
#define fsDirPosFirst (U32)0
#define VOL_METHOD STATUS EXPORTED

Flag to direct VNCreate to create short directory names (See VNCreate)

#define fsShortDirName fsNodeReadOnly

Error status codes

#define stsNoMoreBuffers MakeStatus(clsVolume, 1)

Informational status codes

#define stsVolFormatIsTimeConsuming MakeWarning(clsVolume, 1)

Resource ids for volume icons

Defined with MakeWknResId (clsVolume, tag)

Stored in groups of 10 values:

Base value defines large icon,

+1 value defines smaller icon,

+2 thru +9 reserved for future.

#define tagVolHardDiskIcon 0 // 1-9 define variants, see above
#define tagVolFloppyDiskIcon 10 // 11-19 define variants, see above
#define tagVolRemotePCIcon 20 // 21-29 define variants, see above
#define tagVolRemoteMacIcon 30 // 31-39 define variants, see above
```
typedef OBJECT VOL;
typedef P_FS_ATTR_SIZE *PP_FS_ATTR_SIZE;
typedef P_FS_ATTR_LABEL *PP_FS_ATTR_LABEL;
typedef U32 FS_ATTR_VALUE, *P_FS_ATTR_VALUE, **PP_FS_ATTR_VALUE;
typedef U32 VOL_VNODE, *P_VOL_VNODE;

typedef struct DIR_ID_CACHE {
    PMEM pBuf;
    U32 used;     // Number used, of allocated space
    U32 free;     // Number free, of allocated space
} DIR_ID_CACHE;

typedef struct VOL_CACHE {
    VOL VNODE vnodeNotKnown; // Used to fake volRAM
    PMEM pRoot;              // Cache dir elem for root vnode
    DIR_ID_CACHE dirIds;    // Dir id cache
    OS_MILLISECONDS lastAccess; // Last access to cache layer
    OS_MILLISECONDS lastVolAccess; // Last access to volume
    OS_MILLISECONDS lastVolWrite; // Last write to volume
    OS_MILLISECONDS refreshRate; // Check with volume this often
to see if volume has changed
    since last vol access
    maxU32 implies unchangeable
    OS_MILLISECONDS flushRate; // Flush cached dirty files
    after this much time has passed
    0 implies flush immediately
    maxU32 implies no flushing
    Default is 2000 (2 secs)
    U16 numDirs;             // Total num of dirs in the cache
    U16 numFiles;            // Total num of files in the cache
    U16 openDirs;            // Num of dirs in the cache
    U16 openFiles;           // Num of files in the cache
    U16 refdDirs;            // Num of opened dirs that have
    non-zero reference counts
    U16 refdFiles;           // Num of opened files that have
    non-zero reference counts
    U16 maxOpenDirs;         // Max dirs that can be left open
    for caching purposes.
    0 implies no dirs
    maxU16 implies as many as wanted
    Default is maxU16
    U16 maxOpenFiles;        // Max files that can be left open
    for caching purposes.
    0 implies no files
    maxU16 implies as many as wanted
    Default is maxU16
    PMEM pFirst;             // First cache entry
    PMEM pLast;              // Last cache entry
    PMEM pWrite;             // Write is to this cache entry
    U32 writePos;            // Write at this position
    U32 writeAmt;            // Write for this amount
    U8 readDirFullInProgress; // If non-zero then fully cached
dirs will not be "purged",
    spareU8;
    spareU16;
    spares[5];
} VOL_CACHE;
Enum16 (VOL_CMN_FLAGS) {
    vcVolIsOnBootDevice = flag0, // This volume is on the boot device (as defined by the MIL)
    // but isn’t necessarily THE boot volume.
    vcVolIsDetachable = flag1, // This volume is not removable but may be detachable.
    vcVolIsSwapVolume = flag2 // This is the swap volume.
};

typedef struct VOL_COMMON {
    struct VOL_RTNS *pRtns;
    OS_SEMA_ID fsSema;
    OS_SEMA_ID volSema;
    VOL_CMN_FLAGS flags;
    U16 vnodeCount;
    OS_HEAP_ID vnodeHeap;
    U16 spare1;
    U16 dhCount;
    P_MEM dhRead;
    U16 spare2;
    U16 fhCount;
    P_MEM fhRead;
    VOL_CACHE cache;
    OBJECT dirIndexFile;
    BOOLEAN dirIndexFileVerified;
    U16 spare;
    U32 spares[5];
} VOL_COMMON;

typedef struct VOL_INFO {
    struct VOL_INFO *pNext;
    FS_VOL_HEADER hdr;
    VOL_COMMON cmn;
    // Volume specific volInfo struct goes here...
} VOL_INFO, *P_VOL_INFO, **PP_VOL_INFO;

Enum16 (VNODE_ACCESS) {
    // Delete node at handle free time?
    vnodeTemp = flag0,
    // Read/write intentions for this handle
    vnodeReadOnly = flag2,
    // Upper byte: exclusivity requirements
    vnodeNoExclusivity = MakeU16(0, 0),
    vnodeDenyWriters = MakeU16(0, 1),
    vnodeExclusiveOnly = MakeU16(0, 2),
    // Uncompress file at VNGet time?
    vnodeUncompress = flag14,
    // Default
    vnodeDefaultAccess = 0 // perm, read/write, noExclusivity
};

#define vnodeIgnoreAccessInfo 0x8000

typedef struct VNODE_CMN_ATTRS {
    FS_NODE_FLAGS nodeFlags;
    FS_DATE_TIME nodeCreated;
    FS_DATE_TIME nodeModified;
} VNODE_CMN_ATTRS, *P_VNODE_CMN_ATTRS;

Enum16 (VNODE_ATTR_FLAGS) {
    vnAttrNodeFlags = flag0,
    vnAttrNodeCreated = flag1,
    vnAttrNodeModified = flag2,
    vnAttrLabelsBuffer = flag8,
    vnAttrValuesBuffer = flag9,
    vnAttrSizesBuffer = flag10
};
## Typedefs for functions supported by each volume class

### Volume related functions follow:

**VolStatus**

Has a volume check for readiness.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_VOL_STATUS) {
  P_VOL_INFO pVolInfo,
  P_BOOLEAN pChanged   // In/Out: Has volume changed?
};
#define VolStatus(pVolInfo, pChanged) 
  ((pVolInfo)->cmn.pRtns->pVolStatus) 
  (pVolInfo, pChanged)

Comments Possible return status are stsOK, stsFSVolDisconnected, other errors. If status is okay, should indicate if volume has changed.

**VolSetVolName**

Has a volume change its volume name.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_VOL_SET_VOL_NAME) {
  P_VOL_INFO pVolInfo,
  P_STRING pName   // New volume name
};
#define VolSetVolName(pVolInfo, pName) 
  ((pVolInfo)->cmn.pRtns->pVolSetVolName) 
  (pVolInfo, pName)

**VolUpdateVolInfo**

Requests that a volume updates its user accessible volume info.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_VOL_UPDATE_VOL_INFO) {
  P_VOL_INFO pVolInfo   // Vol Info
};
#define VolUpdateVolInfo(pVolInfo) 
  (pVolInfo)

**VolSpecificMsg**

Passes a volume specific message down to a volume.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_VOL_SPECIFIC_MSG) {
  P_VOL_INFO pVolInfo,
  VOL_vNODE vnode,   // Handle of vnode
  MESSAGE msg,      // Message
  P_UNKNOWN pArgs   // In/Out: Arguments for message
};
#define VolSpecificMsg(pVolInfo, vnode, msg, pArgs) 
  ((pVolInfo)->cmn.pRtns->pVolSpecificMsg) 
  (pVolInfo, vnode, msg, pArgs)
Common vnode access/release functions follow:

**VNGet**

Gets a vnode given `pVolInfo`, `dirVNode` and name of node in the directory.

Returns `STATUS`.

```c
typedef STATUS Functionptr(P_VNODE_GET) (P_VOL_INFO pVolInfo, VOL_VNODE dirVNode, P_STRING pName, P_VOL_VNODE pVolSpecific, P_VOL_VNODE pVNode);

#define VNGet(pVolInfo, dirVNode, pName, access, pVolSpecific, pVNode) 

   ((pVolInfo)->cmn.pRtns->pVNodeGet) 

      (pVolInfo, dirVNode, pName, access, pVolSpecific, pVNode)
```

**VNNextChild**

Gets a vnode given `pVolInfo`, `dirVNode` and dir position in a directory.

Returns `STATUS`.

```c
typedef STATUS Functionptr(P_VNODE_NEXT_CHILD) (P_VOL_INFO pVolInfo, VOL_VNODE dirVNode, P_U32 pDirPos, P_STRING pName, P_VOL_VNODE pVNode);

#define VNNextChild(pVolInfo, dirVNode, pDirPos, access, pName, pVNode) 

   ((pVolInfo)->cmn.pRtns->pVNodeNextChild) 

      (pVolInfo, dirVNode, pDirPos, access, pName, pVNode)
```

**VNGetByDirId**

Gets the vnode of a directory (and its name) given its directory id.

Returns `STATUS`.

```c
typedef STATUS Functionptr(P_VNODE_GET_BY_DIR_ID) (P_VOL_INFO pVolInfo, VOL_VNODE dirVNode, U32 dirId, P_STRING pName, P_VOL_VNODE pVNode);

#define VNGetByDirId(pVolInfo, dirVNode, dirId, pName, pVNode) 

   ((pVolInfo)->cmn.pRtns->pVNodeGetByDirId) 

      (pVolInfo, dirVNode, dirId, pName, pVNode)
```
VNDup

Increments the reference count on a vnode.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_VNODE_DUP) {
    P_VOL_INFO pVolInfo, // Vol Info
    VOL_VNODE vnode, // The vnode being dupped
    VNODE_ACCESS access // R/W, exclusivity, etc.
};
#define VNDup(pVolInfo, vnode, access) 
    ((pVolInfo)->cmn.pRtns->pVNodeDup)
    (pVolInfo, vnode, access)

VNRelease

Returns a vnode to the volume.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_VNODE_RELEASE) {
    P_VOL_INFO pVolInfo, // Vol Info
    VOL_VNODE vnode // The vnode being released
};
#define VNRelease(pVolInfo, vnode) 
    ((pVolInfo)->cmn.pRtns->pVNodeRelease)
    (pVolInfo, vnode)

Directory handle related functions follow:

VNCCreate

Creates a new file or directory node in the given (directory) node.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_VNODE_CREATE) {
    P_VOL_INFO pVolInfo,
    VOL_VNODE dirVNode, // Handle of directory vnode
    P_STRING pName, // Name of the new file
    FS_NODE_FLAGS type // File or directory?
};
#define VNCreatepVolInfo, dirVNode, pName, type
    (pVolInfo)->cmn.pRtns->pVNodeCreate)
    (pVolInfo, dirVNode, pName, type)

Comments

Note: the parameter type only uses the flag fsNodeDir to distinguish between directories and files and the flag fsShortDirName to direct the volume to use a short name replacement for the directory name. Directories are only shortened if they reside in the PenPoint tree. The flag fsShortDirName overlaps fsNodeReadOnly, which is never used in conjunction with directories.
**VNDelete**

Deletes the given node.

Returns STATUS.

Function Prototype:

```c
typedef STATUS FunctionPtr(P_VNODE_DELETE) (  
    P_VOL_INFO pVolInfo,  
    VOL_VNODE vnode,           // VNode to delete  
    BOOLEAN visible           // At root of hierarchical delete?
);
```

#define VNDelete(pVolInfo, vnode, visible)    
    ((pVolInfo)->cmn.pRtns->pVNodeDelete)    
        (pVolInfo, vnode, visible)

Comments:

VNode may be returned differently to mark it as a vnode that points to a deleted vnode.

**VNMove**

Moves/renames a node (and any children) to a new node.

Returns STATUS.

Function Prototype:

```c
typedef STATUS FunctionPtr(P_VNODE_MOVE) (  
    P_VOL_INFO pVolInfo,  
    VOL_VNODE srcDirVNode,     // Handle of dir node of source  
    VOL_VNODE srcVNode,        // Handle of source vnode of move  
    VOL_VNODE dstDirVNode,     // Handle of dir node of dest  
    P_STRING pDstName,         // New name to give the node
);
```

#define VNMove(pVolInfo, srcDirVNode, srcVNode, dstDirVNode, pDstName)    
    ((pVolInfo)->cmn.pRtns->pVNodeMove)    
        (pVolInfo, srcDirVNode, srcVNode, dstDirVNode, pDstName)

**VNDirPosDeleteAdjust**

Makes any necessary adjustment to the dirPos after a node has been deleted.

Returns STATUS.

Function Prototype:

```c
typedef STATUS FunctionPtr(P_VNODE_DIR_POS_DEL_ADJ) (  
    P_VOL_INFO pVolInfo,  
    VOL_VNODE dirVNode,     // Handle of directory vnode  
    VOL_VNODE vnode,        // Handle of deleted vnode  
    P_U32 pDirPos           // Dir position data before delete
);
```

#define VNDirPosDeleteAdjust(pVolInfo, dirVNode, vnode, pDirPos)    
    ((pVolInfo)->cmn.pRtns->pVNodeDirPosDelAdj)    
        (pVolInfo, dirVNode, vnode, pDirPos)

**VNGetDirId**

Gets a directory node's dir id, given the vnode.

Returns STATUS.

Function Prototype:

```c
typedef STATUS FunctionPtr(P_VNODE_GET_DIR_ID) (  
    P_VOL_INFO pVolInfo,  
    VOL_VNODE vnode,        // Handle of vnode  
    P_U32 pDirId            // In/Out: dir id of dir node
);
```

#define VNGetDirId(pVolInfo, vnode, pDirId)    
    ((pVolInfo)->cmn.pRtns->pVNodeGetDirId)    
        (pVolInfo, vnode, pDirId)
File handle related functions follow:

**VNRead**

Transfers n bytes from position m in a file to a buffer.

Returns STATUS.

```c
typedef STATUS FunctionPtr(P_VNODE_READ) {
    P_VOL_INFO pVolInfo,
    VOL_VNODE vnode,          // Handle of vnode
    U32   filePos,            // Starting point of read
    U32   numBytes,           // Number of bytes to read
    P_U8  pReadBuffer,        // Destination of bytes read
    P_U32 pCount              // In/Out: Actual bytes read
};
#define VNRead(pVolInfo, vnode, filePos, numBytes, pReadBuffer, pCount) 
    ((pVolInfo)->cmn.pRtns->pVNodeRead) 
        (pVolInfo, vnode, filePos, numBytes, pReadBuffer, pCount)
```

**VNWrite**

Transfers n bytes from a buffer to position m in a file.

Returns STATUS.

```c
typedef STATUS FunctionPtr(P_VNODE_WRITE) {
    P_VOL_INFO pVolInfo,
    VOL_VNODE vnode,          // Handle of vnode
    U32   filePos,            // Starting point of the write
    U32   numBytes,           // Number of bytes to write
    P_U8  pWriteBuffer,       // Destination of bytes to write
    P_U32 pCount              // In/Out: Actual bytes written
};
#define VNWrite(pVolInfo, vnode, filePos, numBytes, pWriteBuffer, pCount) 
    ((pVolInfo)->cmn.pRtns->pVNodeWrite) 
        (pVolInfo, vnode, filePos, numBytes, pWriteBuffer, pCount)
```

**VNGetSize**

Gets a node's size given the vnode.

Returns STATUS.

```c
typedef STATUS FunctionPtr(P_VNODE_GET_SIZE) {
    P_VOL_INFO pVolInfo,
    VOL_VNODE vnode,          // Handle of vnode
    P_FS_FILE_SIZE pFileSize  // In/Out: Node’s size
};
#define VNGetSize(pVolInfo, vnode, pFileSize) 
    ((pVolInfo)->cmn.pRtns->pVNodeGetSize) 
        (pVolInfo, vnode, pFileSize)
```
**VNSetSize**
Sets a node's size given the vnode and the new size.
Returns STATUS.

Function Prototype:
```c
typedef STATUS FunctionPtr(P_VNODE_SET_SIZE) (  
    P_VOL_INFO pVolInfo,  
    VOL_VNODE vnode,    // Handle of vnode  
    FS_FILE_SIZE fileSize    // Node's new size
);
```

```c
#define VNSetSize(pVolInfo, vnode, fileSize)  
```

Comments:
This function could be used to either truncate or grow the file/resFile.

**Attribute related functions follow:**

**VNGetName**
Gets a node's name, given the vnode.
Returns STATUS.

Function Prototype:
```c
typedef STATUS FunctionPtr(P_VNODE_GET_NAME) (  
    P_VOL_INFO pVolInfo,  
    VOL_VNODE vnode,    // Handle of vnode  
    P_STRING pName      // In/Out: name of node
);
```

```c
#define VNGetName(pVolInfo, vnode, pName)  
    ((pVolInfo)->cmn.pRtns->pVNodeGetName) (pVolInfo, vnode, pName)
```

**VNGetNumAttrs**
Returns the number of non-standard attributes, given the vnode.
Returns STATUS.

Function Prototype:
```c
typedef STATUS FunctionPtr(P_VNODE_GET_NUM_ATTRS) (  
    P_VOL_INFO pVolInfo,  
    VOL_VNODE vnode,    // Handle of vnode  
    P_U16 pNumAttrs    // Out: num of attrs to get
);
```

```c
#define VNGetNumAttrs(pVolInfo, vnode, pNumAttrs)  
```

**VNGetAttrInfo**
Returns a node's attributes, given the vnode.
Returns STATUS.

Function Prototype:
```c
typedef STATUS FunctionPtr(P_VNODE_GET_ATTR_INFO) (  
    P_VOL_INFO pVolInfo,  
    VOL_VNODE vnode,    // Handle of vnode  
    U16 num,        // Num of attrs to get  
    VNODE_ATTR_FLAGS flgs, // Get which attrs  
    P_VNODE_CNM_ATTRS pCmn, // Common attrs  
    P_US pWhich,      // Which user defined attrs  
    P_FS_ATTR_LABEL pLbls,    // In/Out: attribute labels
);
```
P FS ATTR VALUE pVals, // In/Out: attribute values
P FS ATTR SIZE pSizs // In/Out: attribute sizes

#define VNGetAttrInfo(pVolInfo, vnode, num, flgs, pCmn, pWhich, pLbls, pVals, pSizs) \
  ((pVolInfo)->cron.pRtns->pVNodeGetAttrInfo) \
  (pVolInfo, vnode, num, flgs, pCmn, pWhich, pLbls, pVals, pSizs)

Comments
Which common attributes and which arrays of the label/value/size arrays that need to be filled in are
defined by the flgs field. Which particular elements of each (label/value/size) array to be filled in is
defined by the pWhich byte array. If num is 0 or pWhich is null then no label/value/size array elements
should be filled in. If an element of pWhich is maxU8 then the corresponding label/value/size array
element should be filled in. If the data is known and set then the pWhich array element should be set to
1 after setting the values.

VNSetAttrInfo
Sets a node's attributes, given the vnode.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_VNODE_SET_ATTR_INFO) {
P_VOL_INFO pVolInfo,
VOL_VNODE vnode, // Handle of vnode
U16 num, // Num of attrs to set
VNODE_ATTRIB_FLAGS flgs, // Set which attrs
P_VNODE_CMN_ATTRIB pCmn, // Common attrs
P_U8 pWhich, // Which user defined attrs
P_FS_ATTRIB_LABEL pLbls, // In/Out: attribute labels
P_FS_ATTRIB_VALUE pVals, // In/Out: attribute values
P_FS_ATTRIB_SIZE pSizs // In/Out: attribute sizes
};
#define VNSetAttrInfo(pVolInfo, vnode, num, flgs, pCmn, pWhich, pLbls, pVals, pSizs) \
  ((pVolInfo)->cron.pRtns->pVNodeSetAttrInfo) \
  (pVolInfo, vnode, num, flgs, pCmn, pWhich, pLbls, pVals, pSizs)

Comments
Which common attributes and which arrays of the label/value/size arrays that need to be stored are
defined by the flgs field. Which particular elements of each (label/value/size) array to be filled in is
defined by the pWhich byte array. If num is 0 or pWhich is null then no label/value/size array elements
should be stored. If an element of pWhich is maxU8 then the corresponding label/value/size array
element should be stored. If the data is stored successfully then the pWhich array element should be set
to 1.

VNMakeNative
Gets rid of all concepts not native to a file system (ie res/info fields) and return the native form name of
the file after being "stripped".

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_VNODE_MAKE_NATIVE) {
P_VOL_INFO pVolInfo,
VOL_VNODE vnode, // Handle of vnode
P_STRING pName // In/Out: Return buffer for native name
};
#define VNMakeNative(pVolInfo, vnode, pName) \
  ((pVolInfo)->cron.pRtns->pVNodeMakeNative) \
  (pVolInfo, vnode, pName)
Misc functions follow:

VNFlush

Flushes all buffers associated with this vnode.
Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_VNODE_FLUSH) {
    P_VOL_INFO pVolInfo,
    VOL_VNODE vnode  // Handle of vnode
};
#define VNFlush(pVolInfo, vnode) 
    ((pVolInfo)->cmn.pRtns->pVNodeFlush) 
    (pVolInfo, vnode)

DirIdGetParent

Gets the dir id of the parent of a node (also identified by dir id).
Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_DIRID_GET_PARENT) ( 
    P_VOL_INFO pVolInfo,
    U32 node,  // Node identified by dir id
    P_U32 pParent,  // In/Out: dir id of parent
    P_BOOLEAN pParentIsRoot  // In/Out: parent is root
);
#define DirIdGetParent(pVolInfo, node, pParent, pParentIsRoot) 
    ((pVolInfo)->cmn.pRtns->pDirIdGetParent) 
    (pVolInfo, node, pParent, pParentIsRoot)

Debugging functions follow:

VNRefCount

Gets the volume's ref count for a vnode.
Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_VNODE_REF_COUNT) ( 
    P_VOL_INFO pVolInfo,  // Vol Info
    VOL_VNODE vnode,  // The vnode to get info about
    P_U16 pRefCount  // Out: Reference count on vnode
);
#define VNRefCount(pVolInfo, vnode, pRefCount) 
    ((pVolInfo)->cmn.pRtns->pVNodeRefCount) 
    (pVolInfo, vnode, pRefCount)

This is the definition for the table of volume routines:

typedef struct VOL_RTNS {
    // Vol General...
    P_VOL_STATUS    pVolStatus;
    P_VOL_SET_VOL_NAME    pVolSetVolName;
    P_VOL_UPDATE_VOL_INFO    pVolUpdateVolInfo;
    P_VOL_SPECIFIC_MSG    pVolSpecificMsg;
    // VNode Access...
    P_VNODE_GET    pVNodeGet;
    P_VNODE_NEXT_CHILD    pVNodeNextChild;
    P_VNODE_GET_BY_DIR_ID    pVNodeGetByDirId;
    P_VNODE_DUP    pVNodeDup;
} VOL.H
Class FileSystem Messages
These messages are used by volume code

msgFSRegisterVolClass
Registers a volume class with the file system.
Takes P_FS_REGISTER_VOL_CLASS, returns STATUS.

#define msgFSRegisterVolClass MakeMsg(clsFileSystem, 0)

Arguments
typedef struct FS_REGISTER_VOL_CLASS {
  CLASS volClass;   // Vol class of volume
  FS_VOL_TYPE volType; // Type of volume
} FS_REGISTER_VOL_CLASS, *P_FS_REGISTER_VOL_CLASS;

msgFSInstallVol
Creates a volume’s root dir handle and register it with the file system.
Takes P_FS_INSTALL_VOL, returns STATUS.

Arguments
typedef struct FS_INSTALL_VOL {
  OBJ KEY key;       // Volume’s key.
  CLASS volClass;    // Class of the volume.
  VOL VNODE vnode;   // Root directory vnode.
  P_VOL INFO pVolInfo; // In/Out: Volume info block.
} FS_INSTALL_VOL, *P_FS_INSTALL_VOL;

Comments
The volume should mark itself as connected and all observers of the FileSystem will be notified that a volume has been installed. (Note: The message is defined in fs.h so observers can use it.)

#define msgFSInstallVol MakeMsg(clsFileSystem, 1)
msgFSRemoveVol

Removes a volume from the file system and destroy its root dir handle.

Takes P_FS_REMOVE_VOL, returns STATUS.

**Arguments**

```c
typedef struct FS_REMOVE_VOL {
    OBJ_KEY key;                  // Volume's key.
    CLASS volClass;               // Class of the volume.
    P_VOL_INFO pVolInfo;          // Volume info block.
) FS_REMOVE_VOL, *P_FS_REMOVE_VOL;
```

**Comments**

Observers of the FileSystem will be notified of the change. (Note: The message is defined in fs.h so observers can use it.)

```c
#define msgFSRemoveVol MakeMsg(clsFileSystem, 2)
```

msgFSConnectVol

Marks a volume as connected and notify observers of the FileSystem.

Takes P_FS_CONNECT_VOL, returns STATUS.

**Arguments**

```c
typedef struct FS_CONNECT_VOL {
    P_VOL_INFO pVolInfo;          // Volume info block.
} FS_CONNECT_VOL, *P_FS_CONNECT_VOL;
```

**Comments**

(Note: The message is defined in fs.h so observers can use it.)

```c
#define msgFSConnectVol MakeMsg(clsFileSystem, 3)
```

msgFSDisconnectVol

Marks a volume as disconnected and notify observers of the FileSystem.

Takes P_FS_DISCONNECT_VOL, returns STATUS.

**Arguments**

```c
typedef struct FS_DISCONNECT_VOL {
    P_VOL_INFO pVolInfo;          // Volume info block.
} FS_DISCONNECT_VOL, *P_FS_DISCONNECT_VOL;
```

**Comments**

(Note: The message is defined in fs.h so observers can use it.)

```c
#define msgFSDisconnectVol MakeMsg(clsFileSystem, 4)
```

msgFSVolList

Returns device list for given class and count of volumes of that class.

Takes P_FS_VOL_LIST, returns STATUS.

```c
#define msgFSVolList MakeMsg(clsFileSystem, 5)
```

**Arguments**

```c
Aligned16(FS_VOL_LIST_ACCESS) {
    fsAccessVolList = 0,       // Also returns head of list.
    fsReleaseVolList = 1,
    fsGetHeadOfVolList = 2
};
```

```c
typedef struct FS_VOL_LIST {
    FS_VOL_LIST_ACCESS access;  // See above.
    OBJECT volClass;            // Class of the volumes.
    U16 volCount;               // Out: Number of volumes.
    P_VOL_INFO pVolInfo;        // Out: First vol info block.
) FS_VOL_LIST, *P_FS_VOL_LIST;
```
msgFSUnRegisterVolClass
UnRegisters a volume class from the file system.
Takes _P_CLASS, returns STATUS.
#define msgFSUnRegisterVolClass MakeMsg(clsFileSystem, 6)

msgFSVolIsBusy
Checks to see if a volume can be removed.
Takes _P_FS_VOL_INFO, returns STATUS.
#define msgFSVolIsBusy MakeMsg(clsFileSystem, 7)

Comments
If no user files/dirs are open and all caches have been written to the volume then the volume may be removed. This method should only be called by the volume to be removed.

If the volume can be removed then _stsOK is returned. If the volume can not be removed then _stsFSVolBusy is returned.

msgFSExclVolAccess
Allows a volume class to obtain exclusive access to a volume and to release the exclusive access.
Takes _P_FS_EXCL_VOL_ACCESS, returns STATUS.
#define msgFSExclVolAccess MakeMsg(clsFileSystem, 8)

Arguments
Enum16(_EXCL_VOL_ACCESS) {
    xvaAcquireVolIfNotBusy = 1,  // Acquire volume if not accessed
    xvaReleaseVol = 2
};
typedef struct _FS_EXCL_VOL_ACCESS {
    _EXCL_VOL_ACCESS mode;
    _P_VOL_INFO pVolInfo;
} _FS_EXCL_VOL_ACCESS, *P_FS_EXCL_VOL_ACCESS;

Comments
This is used during the update volume list portions of volume classes. Volume classes should not try to update a volume if it is busy.

If the volume was not busy and was acquired then _stsOK is returned. If the volume was busy then a non _stsOK is returned.

Class Volume Messages

msgVolUpdateVolumes
Has the volume class update its list of volumes.
Takes _P_VOL_UPDATE_VOLUMES, returns STATUS.
#define msgVolUpdateVolumes MakeMsg(clsVolume, 0)

Arguments
Enum16(_FS_UPDATE_VOLS_MODE) {
    // An update should be done to all devices
    fsUpdateAllDevices = flag0,
    // The update request is in response to a power down notification
    fsUpdatePoweringDown = flag1,
    // The update request is in response to a power up notification
    fsUpdatePoweringUp = flag2,
    // Update searching for a volume?
    fsUpdateSearchingForVolume = flag3
};
typedef struct VOL_UPDATE_VOLUMES {
    FS_UPDATE_VOLS_MODE updateMode;  // See above.
    U32 spare1;                      // For future use.
    U32 spare2;                      // For future use.
} VOL_UPDATE_VOLUMES, *P_VOL_UPDATE_VOLUMES;

Comments
All volumes are sent this message every two seconds to give them a chance to do periodic volume updating. If the user has requested a disk/volume that is not connected then volumes are sent this message with the fsUpdateSearchingForVolume flag set. Volumes should not notify observers of volume connections, disconnections etc if a search is in progress. The notification should be deferred until a later update request is sent. If the user has triple tapped on the connections notebook, asking to update all volumes, then volumes are sent this message with the fsUpdateAllDevices flag set.

Volume Specific Messages

msgVolEjectMedia
Has the volume eject its media.
Takes void, returns STATUS.
#define msgVolEjectMedia MakeMsg(clsVolume, 10)

Comments
Passed as a volume specific msg by the file system.

msgVolInvalidateCaches
Allows volumes to invalidate cache buffers at warm boot time.
Takes void, returns STATUS.
#define msgVolInvalidateCaches MakeMsg(clsVolume, 11)

Comments
Passed as a volume specific msg by the file system at power up time.

msgVolUpdateBootCode
Reads image of boot sector from mil.res and stores onto boot sector.
Takes void, returns STATUS.
#define msgVolUpdateBootCode MakeMsg(clsVolume, 12)

Comments
Passed as a volume specific msg by the installation utility.

Class Volume Messages Formatting

msgVolFormatVolumeInit
This msg is sent to a volume to initiate a reformat of the volume.
Takes P_VOL_FORMAT_MEDIA_INIT, returns STATUS.
#define msgVolFormatVolumeInit MakeMsg(clsVolume, 20)

Comments
This initiates the format from the current owner of the block device. The volume object is destroyed (although there is a possibility that the destroy will fail) and then the block device of that volume, the volume offset on the block device and the volume size are returned. Call the volume class that is to format the volume with the message msgVolFormatMediaInit passing it this information. It will return a format id.
Note that all other format related messages are sent to the class of the volume, because the volume will no longer exist.

**msgVolFormatMediaInit**

Takes a block device object and returns a format id to be used with the other format messages.

Takes `P_VOL_FORMAT_MEDIA_INIT`, returns `STATUS`.

```c
#define msgVolFormatMediaInit MakeMsg(clsVolume, 21)
```

**Arguments**

```c
typedef struct VOL_FORMAT_MEDIA_INIT {
    OBJECT blockDevice;  // A block device
    U32 volumeOffset;     // Format device beginning here
    U32 volumeSize;       // Amount of device to be formatted
    P_UNKNOWN formatId;   // Out: Format id
} VOL_FORMAT_MEDIA_INIT, *P_VOL_FORMAT_MEDIA_INIT;
```

**Comments**

NOTE: `volumeOffset` should be zero and `volumeSize` should be zero if you wish to format the entire device (vs a partition of the device).

**msgVolMediaCapacities**

Returns the possible format capacities for the device requesting format.

Takes `P_VOL_MEDIA_CAPACITIES`, returns `STATUS`.

```c
#define msgVolMediaCapacities MakeMsg(clsVolume, 22)
```

**Arguments**

```c
typedef struct VOL_MEDIA_CAPACITIES {
    P_UNKNOWN formatId;  // Format id from format/reformat.
    U16 maxCapacities;   // Size of output capacities array.
    U16 numCapacities;   // Out: Actual number of capacities.
    P_U32 pCapacities;   // In/Out: Capacities.
} VOL_MEDIA_CAPACITIES, *P_VOL_MEDIA_CAPACITIES;
```

**Comments**

This message is sent to the class of the volume.

**msgVolFormatMediaSetup**

Has the vol class set the media to be ready for a format and determines if the block device will require format media (vs format track).

Takes `P_VOL_FORMAT_MEDIA`, returns `STATUS`.

```c
#define msgVolFormatMediaSetup MakeMsg(clsVolume, 23)
```

**Arguments**

```c
typedef struct VOL_FORMAT_MEDIA {
    P_UNKNOWN formatId;   // Format id from format/reformat.
    U32 capacity;         // Desired capacity to format for.
    P_STRING pName;       // Name of re/formatted volume.
    U16 percentDone;      // Out: Progress report.
} VOL_FORMAT_MEDIA, *P_VOL_FORMAT_MEDIA;
```

**Comments**

This message is sent to the class of the volume.

**msgVolFormatMediaBegin**

Has the vol class begin the format of its media.

Takes `P_VOL_FORMAT_MEDIA`, returns `STATUS`.

```c
#define msgVolFormatMediaBegin MakeMsg(clsVolume, 24)
```
typedef struct VOL_FORMAT_MEDIA {
    P_UNKNOWN formatId;  // Format id from format/reformat.
    U32 capacity;  // Desired capacity to format for.
    P_STRING pName;  // Name of re/formatted volume.
    U16 percentDone;  // Out: Progress report.
} VOL_FORMAT_MEDIA, *P_VOL_FORMAT_MEDIA;

This step may do a format media if format track is not supported by the block device and may partition
the media if it needs partitioning.

This messages is sent to the class of the volume.

msgVolFormatMediaCont
Has the vol class do a format of its media.
Takes P_VOL_FORMAT_MEDIA, returns STATUS.

#define msgVolFormatMediaCont MakeMsg(clsVolume, 25)

msgVolCancelFormat
Has the vol class cancel the format.
Takes P_UNKNOWN, returns STATUS.

#define msgVolCancelFormat MakeMsg(clsVolume, 26)

msgVolDuplicateVolume
This msg is sent to a volume to initiate a duplication of that volume.
Takes PP_UNKNOWN, returns STATUS.

#define msgVolDuplicateVolume MakeMsg(clsVolume, 30)

msgVolDuplicateMedia
Has the volume class duplicate more of the disk.
Takes P_VOL_DUPLICATE_MEDIA, returns STATUS.

#define msgVolDuplicateMedia MakeMsg(clsVolume, 31)
typedef struct VOL_DUPLICATE_MEDIA {
    P_UNKNOWN duplicateId; // Duplicate id from duplicate.
    BOOLEAN sourceDisk; // Is this source or destination?
    U16 percentDone; // Out: Progress report.
} VOL_DUPLICATE_MEDIA, *P_VOL_DUPLICATE_MEDIA;

Comments
If source is TRUE then data will be read from the source disk. If source is FALSE then data is written to the destination disk. The value percentDone is updated to reflect how much of the duplication has been completed. If percentDone is not 100, then keep calling this until it is.

msgVolDuplicateReady
Checks to see if the source/dest disk of the duplicate is ready.

Takes P_VOL_DUPLICATE_MEDIA, returns STATUS.
#define msgVolDuplicateReady MakeMsg(clsVolume, 32)

typedef struct VOL_DUPLICATE_MEDIA {
    P_UNKNOWN duplicateId; // Duplicate id from duplicate.
    BOOLEAN sourceDisk; // Is this source or destination?
    U16 percentDone; // Out: Progress report.
} VOL_DUPLICATE_MEDIA, *P_VOL_DUPLICATE_MEDIA;

Comments
The return percentDone is unused.

msgVolCancelDuplication
Have the vol class cancel the duplication.

Takes P_UNKNOWN, returns STATUS.
#define msgVolCancelDuplication MakeMsg(clsVolume, 33)
This file contains declarations for the common part of godir volumes. Examples of these include clsVolDisk and clsVolTOPS.

Information in this file is useful if you are trying to understand the format of PenPoint.dir files or if you are writing an installable volume.

```c
#ifndef VOLGODIR_INCLUDED
#define VOLGODIR_INCLUDED

#include <go.h>
#include <os.h>
#include <clsmgr.h>
#include <fs.h>
#include <vol.h>

Common #defines and typedefs

Defines

GO directory related defines
```
#define goNameIndex
#define goDirSearchFromFirst
#define goDirHeaderBufSize
```

Types

General types

Enumerated constants for searching for particular directory entries

```c
Enum16 (GO_DIR_FINDTYPE) {
    gdFindEmpty = 0,
    gdFindNextName = 1,
    gdFindNativeName = 2,
    gdFindGoDirName = 3
};
```
Note that this can also be treated as an array of U32, using the tag part of the associated fsAttr as the index into the array, except flags and unused together form a special case of a U32!!!

typedef struct VOLGODIR_CMN_ATTRS {
    FS_NODE_FLAGS flags;
    U16 unused;       // Was sequence
    FS_DATE_TIME dateCreated;
    FS_DATE_TIME dateModified;
    FS_FILE_SIZE fileSize;
} VOLGODIR_CMN_ATTRS, *P_VOLGODIR_CMN_ATTRS;

GO directory related types

Each directory entry is identified as either erased (e) or full (f).

Enum16(GO_DIR_ENTRY_TYPES) {
    goDirUnusedEntry = 'e',
    goDirNodeEntry = 'f'
};

typedef struct GO_DIR_USER_ATTR {
    FS_ATTR_LABEL label;     // file system attribute label.
    U16 size;                // size of value field.
    U8 value;                // a U32, string or var length attr.
} GO_DIR_USER_ATTR, *P_GO_DIR_USER_ATTR;

typedef struct GO_DIR_ENTRY_HEADER {
    U8 type;                  // 'e': erased or 'f' for file/dir.
    U16 size;                 // Actual size on disk is modulo 32.
} GO_DIR_ENTRY_HEADER, *P_GO_DIR_ENTRY_HEADER;

Go name is located at goDirEntry.buf, always the first entry. The define goNameIndex can be used to index to the name. It is important that the size of GO_DIR_ENTRY is modulo 32.

typedef struct GO_DIR_ENTRY {
    GO_DIR_ENTRY_HEADER hdr;
    U16 numUserAttrs;        // Number of user attributes.
    U8 nativeNameIndex;      // Offset to native file name.
    U8 rsrvdForLater;        // UNUSED SPARE.
    U8 userAttrsIndex;       // Offset to first user attr.
    FS_NODE_FLAGS flags;
    U16 rsrvdForLater2;      // WAS SEQUENCE
    FS_DATE_TIME dateCreated;
} GO_DIR_ENTRY, *P_GO_DIR_ENTRY, **PP_GO_DIR_ENTRY;

VNode types

VNode related type declarations

Enum16(VOLGODIR_VNODE_FLAGS) {
    gdfPenPointDir = flag1,    // This is a PenPoint.Dir file
    gdfRootDir    = flag2,
    gdfNodeCorrupt = flag3,
    gdfNodeModified = flag4,
    gdfHasGoDirParent = flag5,
    gdfHasGoDirSister = flag6,
    gdfNoGoDirSister = flag7
};

typedef struct VOLGODIR_VNODE_COMMON {
    U16 refCount;
    U16 numUserAttrs;
    U32 goDirPos;
    VOLGODIR_VNODE_FLAGS flags;
    VOLGODIR_CMN_ATTRS attrs;
} VOLGODIR_VNODE_COMMON;
typedef struct VOLGODIR_VNODE {
    struct VOLGODIR_VNODE *pNext;
    VOLGODIR_VNODE_COMMON cmn;
} VOLGODIR_VNODE, *P_VOLGODIR_VNODE, **PP_VOLGODIR_VNODE;

Penpoint dir cache
typedef struct GO_DIR_CACHE {
    U32 size;     // How much of data is valid?
    U32 base;     // Position in penpoint dir.
    P_VOLGODIR_VNODE owner; // Cache for which dir.
    U8 buffer [512]; // Fixed size buffer.
} GO_DIR_CACHE, *P_GO_DIR_CACHE;

VolInfo types
This is the instance data for a GO dir volume object
typedef struct VOLGODIR_INFO {
    // Common volume info...
    struct VOLGODIR_INFO *pNext;
    FS_VOL_HEADER hdr;
    VOL_COMMON cmn;
    // Pointer to the low level volumes routines...
    struct VOLGODIR_RTNS *pRtns;
    // Head of the vnode chain...
    P_VOLGODIR_VNODE pFirstVNode;
    // Buffer used by the GO DIR volume part - does not need to be inited...
    GO_DIR_ENTRY goDirEntry;
    // GO DIR buffer & info...
    GO_DIR_CACHE goDirCache;
    // Beyond this point each volume will have their own info...
    / /
    / /
    / /
} VOLGODIR_INFO, *P_VOLGODIR_INFO;

Exported routine that returns pointer GoDirShell entrypoint table
P_VOL_RTNS EXPORTED GoDirShellEntrypoint (void);

Typedfs for functions supported by each godir lower level volume

LVStatus
Has a volume check for readiness.
Returns STATUS.

typedef STATUS FunctionPtr(P LVOL_STATUS) {
    P_VOLGODIR_INFO pVolInfo,
    P_BOOLEAN pChanged      // In/Out: Has volume changed?
};
#define LVStatus(pVolInfo, pChanged) \
    ((pVolInfo)->pRtns->pLVolStatus) \
        (pVolInfo, pChanged)

Comments Possible return status are stsOK, stsFSVolDisconnected, other errors. If status is okay, should indicate if volume has changed.
LVSetVolName
Requests for a volume to set/change its volume name.
Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_LVOL_SET_VOL_NAME) (  
P_VOLGODIR_INFO pVolInfo, // Vol Info  
P_STRING pName, // Vol name
);
#define LVSetVolName(pVolInfo, pName) \  
((pVolInfo)->pRtns->pLVolSetVolName) \  
(pVolInfo, pName)

LVUpdateInfo
Requests for a volume to update its user accessible volume info.
Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_LVOL_UPDATE_INFO) (  
P_VOLGODIR_INFO pVolInfo \  
);
#define LVUpdateInfo(pVolInfo) \  
((pVolInfo)->pRtns->pLVolUpdateInfo) \  
(pVolInfo)

LVSpecificMsg
Passes a volume specific message down to a volume.
Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_LVOL_SPECIFIC_MSG) (  
P_VOLGODIR_INFO pVolInfo, \  
P_VOLGODIR_VNODE pVNode, // Handle of vnode  
MESSAGE msg, // Message  
P_UNKNOWN pArgs, // In/Out: Arguments for message
);
#define LVSpecificMsg(pVolInfo, pVNode, msg, pArgs) \  
((pVolInfo)->pRtns->pLVolSpecificMsg) \  
(pVolInfo, pVNode, msg, pArgs)

LVNGet
Gets a vnode given pVolInfo, dirVNode and name of node in the directory.
Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_LVNODE_GET) (  
P_VOLGODIR_INFO pVolInfo, // Vol Info  
P_VOLGODIR_VNODE pDirVNode, // VNode of parent directory  
P_STRING pFileName, // Name of file node  
P_UNKNOWN pVolSpecific, // Vol specific info  
PP_VOLGODIR_VNODE ppVNode, // Out: Returned vnode handle
);
#define LVNGet(pVolInfo, pDirVNode, pFileName, pVolSpecific, ppVNode) \  
((pVolInfo)->pRtns->pLVNodeGet) \  
(pVolInfo, pDirVNode, pFileName, pVolSpecific, ppVNode)
LVNGetAndOpenParent

Gets a vnode's parent given pVolInfo and a vnode and open it.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_LVNODE_GET_OPEN_PARENT) {
    P_VOLGODIR_INFO pVolInfo, // Vol Info
    P_VOLGODIR_VNODE pVNode, // VNode to get parent of
    PP_VOLGODIR_VNODE ppDirVNode, // Out: VNode handle of parent
    P_BOOLEAN pComplete // Out: Did the vnode already exist?
};
define LVNGetAndOpenParent(pVolInfo, pVNode, ppDirVNode, pComplete)

LVNGetAndOpenByDirId

Gets a dir vnode given pVolInfo and the directory's dirID.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_LVNODE_GET_OPEN_BY_DIR_ID) {
    P_VOLGODIR_INFO pVolInfo, // Vol Info
    P_VOLGODIR_VNODE pVNode, // VNode of parent of dir
    U32 dirId, // Dir ID of vnode to get & open
    PP_VOLGODIR_VNODE ppDirVNode, // Out: Returned vnode handle of dir
    P_BOOLEAN pComplete // Out: Did the vnode already exist?
};
define LVNGetAndOpenByDirId(pVolInfo, pDirVNode, dirId, ppDirVNode, pComplete)
    ((pVolInfo)->pRtns->pLVNodeGetAndOpenByDirId)(pVolInfo, pDirVNode, dirId, ppDirVNode, pComplete)

Comments
Note: pDirVNode could be null. If it isn't then it can be used.

LVNRelease

Releases a vnode.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_LVNODE_RELEASE) {
    P_VOLGODIR_INFO pVolInfo, // Vol Info
    P_VOLGODIR_VNODE pVNode // VNode to release
};
define LVNRelease(pVolInfo, pVNode)

LVNOpen

Opens a vnode.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_LVNODE_OPEN) {
    P_VOLGODIR_INFO pVolInfo, // Vol Info
    P_VOLGODIR_VNODE pVNode, // VNode to open
    P_STRING pName, // Name of node
    VNODE_ACCESS access // R/W, exclusivity, etc.
};
define LVNOpen(pVolInfo, pVNode, pName, access)
    ((pVolInfo)->pRtns->pLVNodeOpen)(pVolInfo, pVNode, pName, access)
LVNClose
Closes a vnode.
Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr (P_LVNODE_CLOSE) {
    P_VOLGODIR_INFO    pVolInfo,    // Vol Info
    P_VOLGODIR_VNODE   pVNode       // VNode to close
};
#define LVNClose(pVolInfo, pVNode) 
    ((pVolInfo)->pRtns->pLVNodeClose) 
    (pVolInfo, pVNode)

LVNCreate
Creates a file or directory within the directory given.
Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr (P_LVNODE_CREATE) {
    P_VOLGODIR_INFO    pVolInfo,    // Vol Info
    P_VOLGODIR_VNODE   pDirVNode,   // Directory where new node belongs
    P_STRING           pName,       // Name of new file/dir
    FS_NODE_FLAGS      fileType     // Create a dir or a file
};
#define LVNCreate(pVolInfo, pDirVNode, pName, fileType) 
    ((pVolInfo)->pRtns->pLVNodeCreate) 
    (pVolInfo, pDirVNode, pName, fileType)

LVNDelete
Deletes a file system node; either a dir or a file node.
Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr (P_LVNODE_DELETE) {
    P_VOLGODIR_INFO    pVolInfo,    // Vol Info
    P_VOLGODIR_VNODE   pVNode,      // VNode to release
    BOOLEAN            visible      // At root of hierarchical delete?
};
#define LVNDelete(pVolInfo, pVNode, visible) 
    ((pVolInfo)->pRtns->pLVNodeDelete) 
    (pVolInfo, pVNode, visible)

LVNMove
Moves a file or directory to a directory w/ the new (old) name.
Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr (P_LVNODE_MOVE) {
    P_VOLGODIR_INFO    pVolInfo,    // Vol Info
    P_VOLGODIR_VNODE   pSrcDirVNode, // Dir of source node
    P_VOLGODIR_VNODE   pSrcVNode,    // Source node
    P_VOLGODIR_VNODE   pDstDirVNode, // Dir of dest
    P_STRING           pDstName,    // Name to give the dest node
};
#define LVNMove(pVolInfo, pSrcDirVNode, pSrcVNode, pDstDirVNode, pDstName) 
    ((pVolInfo)->pRtns->pLVNodeMove) 
    (pVolInfo, pSrcDirVNode, pSrcVNode, pDstDirVNode, pDstName)
LVNReadDir

Returns the next entry from the specified directory.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_LVNODE_READ_DIR) {
  P_VOLGODIR_INFO pVolInfo, // Vol Info
  P_VOLGODIR_VNODE pDirVNode, // Directory to read from
  P_U32 pDirPos, // In/Out: Current position
  P_STRING pName // Out: Name of the node
};
#define LVNReadDir(pVolInfo, pDirVNode, pDirPos, pName) 
  ((pVolInfo)->pRtns->pLVNodeReadDir) 
  (pVolInfo, pDirVNode, pDirPos, pName)

LVNDirPosDeleteAdjust

Makes any necessary adjustment to the dirPos after a node has been deleted.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_LVNODE_DIR_POS_DEL_ADJUST) {
  P_VOLGODIR_INFO pVolInfo,
  P_VOLGODIR_VNODE dirVNode, // Handle of directory vnode
  P_VOLGODIR_VNODE vnode, // Handle of deleted vnode
  P_U32 pDirPos // In/Out: Dir pos data before delete
};
#define LVNDirPosDeleteAdjust(pVolInfo, dirVNode, vnode, pDirPos) 
  ((pVolInfo)->pRtns->pLVNodeDirPosDelAdjust) 
  (pVolInfo, dirVNode, vnode, pDirPos)

LVNGetDirId

Returns a well known constant dir id that represents this directory.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_LVNODE_GET_DIR_ID) {
  P_VOLGODIR_INFO pVolInfo, // Vol Info
  P_VOLGODIR_VNODE pVNode, // Return dir id of this dir vnode
  P_U32 pDirId // In/Out: The directory's id
};
#define LVNGetDirId(pVolInfo, pVNode, pDirId) 
  ((pVolInfo)->pRtns->pLVNodeGetDirId) 
  (pVolInfo, pVNode, pDirId)

LVNName

Returns the name a file system node.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_LVNODE_NAME) {
  P_VOLGODIR_INFO pVolInfo, // Vol Info
  P_VOLGODIR_VNODE pVNode, // VNode to get name of
  P_STRING pName // In/Out: Name
};
#define LVNName(pVolInfo, pVNode, pName) 
  ((pVolInfo)->pRtns->pLVNodeName) 
  (pVolInfo, pVNode, pName)
LVNGetNumAttrs

Returns the number of non-standard attributes, given the vnode.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr (P_LVNODE_GET_NUM_ATTRS) (  
P_VOLGODIR_INFO pVolInfo, // Vol Info  
P_VOLGODIR_VNODE pVNode, // VNode of node to read from  
U16 nnumAttr // Out: num of attrs to get  
);
#define LVNGetNumAttrs(pVolInfo, pVNode, pNumAttrs)  
((pVolInfo)->pRtns->pLVNodeGetNumAttrs)  
(pVolInfo, pVNode, pNumAttrs)

LVNGetAttrInfo

Gets a node's attributes, given the vnode.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr (P_LVNODE_GET_ATTR_INFO) (  
P_VOLGODIR_INFO pVolInfo, // Vol Info  
P_VOLGODIR_VNODE pVNode, // VNode of node to read from  
U16 nnum, // Num of attrs to get  
VNODE_ATTR_FLAGS flgs, // Get which common attrs  
P_VNODE_CMN_ATTRS pCmn, // Common attrs  
P_U8 pWhich, // Which user defined attrs  
P_FS_ATTR_LABEL pLbls, // In/Out: attribute labels  
P_FS_ATTR_VALUE pVals, // In/Out: attribute values  
P_FS_ATTR_SIZE pSizs // In/Out: attribute sizes  
);
#define LVNGetAttrInfo(pVolInfo, pVNode, num, flgs, pCmn, pWhich, pLbls, pVals, pSizs)  
((pVolInfo)->pRtns->pLVNodeGetAttrInfo)  
(pVolInfo, pVNode, num, flgs, pCmn, pWhich, pLbls, pVals, pSizs)

Comments

Which common attributes and which arrays of the label/value/size arrays that need to be filled in are defined by the flgs field. Which particular elements of each (label/value/size) array to be filled in is defined by the pWhich byte array. If num is 0 or pWhich is null then no label/value/size array elements should be filled in. If an element of pWhich is maxU8 then the corresponding label/value/size array element should be filled in. If the data is known and set then the pWhich array element should be set to 1 after setting the values.

LVNSetAttrInfo

Sets a node's attributes, given the vnode.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr (P_LVNODE_SET_ATTR_INFO) (  
P_VOLGODIR_INFO pVolInfo, // Vol Info  
P_VOLGODIR_VNODE pVNode, // VNode of node to read from  
U16 nnum, // Num of attrs to set  
VNODE_ATTR_FLAGS flgs, // Set which common attrs  
P_VNODE_CMN_ATTRS pCmn, // Common attrs  
P_U8 pWhich, // Which user defined attrs  
P_FS_ATTR_LABEL pLbls, // In/Out: attribute labels  
P_FS_ATTR_VALUE pVals, // In/Out: attribute values  
P_FS_ATTR_SIZE pSizs // In/Out: attribute sizes  
);
#define LVNSetAttrInfo(pVolInfo, pVNode, num, flgs, pCmn, pWhich, pLbls, pVals, pSizs)  
((pVolInfo)->pRtns->pLVNodeSetAttrInfo)  
(pVolInfo, pVNode, num, flgs, pCmn, pWhich, pLbls, pVals, pSizs)
Which common attributes and which arrays of the label/value/size arrays that need to be stored are defined by the flgs field. Which particular elements of each (label/value/size) array to be filled in is defined by the pWhich byte array. If num is 0 or pWhich is null then no label/value/size array elements should be stored. If an element of pWhich is maxU8 then the corresponding label/value/size array element should be stored. If the data is stored successfully then the pWhich array element should be set to 1.

LVNRead

Transfers n bytes from position m in a file to a buffer.

Returns STATUS.

Function Prototype:
```c
typedef STATUS FunctionPtr(P_LVNODE_READ) {
    P_VOLGODIR_INFO pVolInfo, // Vol Info
    P_VOLGODIR_VNODE pVNode, // VNode of node to read from
    U32 filePos, // Starting point of read
    U32 numBytes, // Number of bytes to be read
    P_U8 pReadBuffer, // Destination of bytes read
    P_U32 pCount // Out: Actual number of bytes read
};
#define LVNRead(pVolInfo, pVNode, filePos, numBytes, pReadBuffer, pCount) ...
```

LVNWrite

Transfers n bytes from a buffer to position m in a file.

Returns STATUS.

Function Prototype:
```c
typedef STATUS FunctionPtr(P_LVNODE_WRITE) {
    P_VOLGODIR_INFO pVolInfo, // Vol Info
    P_VOLGODIR_VNODE pVNode, // VNode of node to write to
    U32 filePos, // Starting point of the write
    U32 numBytes, // Number of bytes to write
    P_U8 pWriteBuffer, // Destination of bytes to write
    P_U32 pCount // Out: Actual number of bytes written
};
#define LVNWrite(pVolInfo, pVNode, filePos, numBytes, pWriteBuffer, pCount) ...
```

LVNGetSize

Returns the size of a file.

Returns STATUS.

Function Prototype:
```c
typedef STATUS FunctionPtr(P_LVNODE_GET_SIZE) {
    P_VOLGODIR_INFO pVolInfo, // Vol Info
    P_VOLGODIR_VNODE pVNode, // VNode of node to change size of
    P_FS_FILE_SIZE pSize // The size of the file
};
#define LVNGetSize(pVolInfo, pVNode, pSize) ...
```
LVNSetSize

Adjusts the size of a file.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_LVNODE_SET_SIZE) {
    P_VOLGODIR_INFO pVolInfo,  // Vol Info
    P_VOLGODIR_VNODE pVNode,   // VNode of node to change size of
    FS_FILE_SIZE newSize,     // The new size
};
#define LVNSetSize(pVolInfo, pVNode, newSize) 
    ((pVolInfo)->pRtns->pLVNodeSetSize) 
    (pVolInfo, pVNode, newSize)

LVNFlush

Flushes a file.

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_LVNODE_FLUSH) {
    P_VOLGODIR_INFO pVolInfo,  // Vol Info
    P_VOLGODIR_VNODE pVNode    // VNode of node to flush
};
#define LVNFlush(pVolInfo, pVNode) 
    ((pVolInfo)->pRtns->pLVNodeFlush) 
    (pVolInfo, pVNode)

LVNativeName

Returns the native file system form of this name.

Returns BOOLEAN.

Function Prototype
typedef BOOLEAN FunctionPtr(P_LV_NATIVE_NAME) {
    P_VOLGODIR_INFO pVolInfo,  // Vol Info
    P_STRING pName              // In/Out: Name
};
#define LVNativeName(pVolInfo, pName) 
    ((pVolInfo)->pRtns->pLVNativeName) 
    (pVolInfo, pName)

Comments
A return of true implies that the name was not changed (was native), and a return of false implies that
the name was changed to be native.

LDirIdGetParent

Gets the dir id of the parent of a node (also identified by dir id).

Returns STATUS.

Function Prototype
typedef STATUS FunctionPtr(P_LDIRID_GET_PARENT) {
    P_VOLGODIR_INFO pVolInfo,  // Vol Info
    U32 node,                  // Node identified by dir id
    P_U32 pParent,            // In/Out: dir id of parent
    P:boolean pParentIsRoot   // In/Out: parent is root
};
#define LDirIdGetParent(pVolInfo, node, pParent, pParentIsRoot) 
    ((pVolInfo)->pRtns->pLDirIdGetParent) 
    (pVolInfo, node, pParent, pParentIsRoot)
This is the definition for the table of volume routines

typedef struct VOLGODIR_RTNS {
    P_LVOL_STATUS      plVolStatus;
    P_LVOL_SET_VOL_NAME plVolSetVolName;
    P_LVOL_UPDATE_INFO  plVolUpdateInfo;
    P_LVOL_SPECIFIC_MSG plVolSpecificMsg;
    P_LVNODE_GET       pLVNodeGet;
    P_LVNODE_GET_OPEN_PARENT pLVNodeGetAndOpenParent;
    P_LVNODE_GET_OPEN_BY_DIR_ID  pLVNodeGetAndOpenByDirId;
    P_LVNODE_RELEASE   pLVNodeRelease;
    P_LVNODE_OPEN      pLVNodeOpen;
    P_LVNODE_CLOSE     pLVNodeClose;
    P_LVNODE_CREATE    pLVNodeCreate;
    P_LVNODE_DELETE    pLVNodeDelete;
    P_LVNODE_MOVE      pLVNodeMove;
    P_LVNODE_READ_DIR  pLVNodeReadDir;
    P_LVNODE_DIR_POS_DEL_ADJUST pLVNodeDirPosDelAdjust;
    P_LVNODE_GET_DIR_ID pLVNodeGetDirId;
    P_LVNODE_NAME      pLVnodeName;
    P_LVNODE_GET_NUM_ATTRS pLVNodeGetNumAttrs;
    P_LVNODE_GET_ATTR_INFO pLVNodeGetAttrInfo;
    P_LVNODE_SET_ATTR_INFO pLVNodeSetAttrInfo;
    P_LVNODE_READ      pLVNodeRead;
    P_LVNODE_WRITE     pLVNodeWrite;
    P_LVNODE_GET_SIZE  pLVNodeGetSize;
    P_LVNODE_SET_SIZE  pLVNodeSetSize;
    P_LVNODE_FLUSH     pLVNodeFlush;
    P_LV_NATIVE_NAME   pLVNativeName;
    P_LDIRID_GET_PARENT pLDirIdGetParent;
} VOLGODIR_RTNS, *P_VOLGODIR_RTNS;
VSEARCH.H

This file contains the API for clsVolSearch.

clsVolSearch inherits from clsObject.

Provides file system ui support, including formatting & duplicating disks. theVolSearcher is the only instance of clsVolSearch.

The categories of functionality provided by theVolSearcher are:

- Reformatting/duplicating a volume:
  These are sent from the disk viewer when a user selects the format or duplicate volume items from the volume menu. The user is lead thru a series of system notes to get the information and for disk swapping.

- Searching for a volume (because it doesn’t exist or is write protected):
  This is sent from the file system when a file system request internally returns a stsFSVolDisconnected or stsFSVolReadOnly.

#ifndef VSEARCHINCLUDED
#define VSEARCHINCLUDED

Include file dependencies
#ifndef GOINCLUDED
#include <go.h>
#endif
#ifndef OSTYPESINCLUDED
#include <ostypes.h>
#endif
#ifndef CLSMGRINCLUDED
#include <clsmgr.h>
#endif
#ifndef FSINCLUDED
#include <fs.h>
#endif

Common #defines and typedefs

These defines and enums define the text for the notes displayed by the volSearcher. The resources are stored in the system resource file.

Defines

Resource ids

#define vsResUIStrings

MakeTag (clsVolSearch, 1)
Types

Resource string numbers

Enum16 (VS_STRING_IDS) {
    vsFindVolumeStrsBase = 0,
    vsFindGenVolumeStr = 0,
    vsFindDiskVolumeStr = 1,
    vsFindRemoteVolumeStr = 2,
    vsWriteProtectedVolumeStr = 3,
    vsCancelButtonStr = 4,
    vsContinueButtonStr = 5,
    vsPercentDoneStr = 6,
    vsFmtNoticeStr = 7,
    vsFmtChooseSizeStr = 8,
    vsFmtWarningStr = 9,
    vsFmtAskForNameStr = 10,
    vsFmtBlankNameErrStr = 11,
    vsFmtBadCharErrStr = 12,
    vsFmtInProgressStr = 13,
    vsDupInProgressStr = 14,
    vsDupInsertSrcDiskStr = 15,
    vsDupInsertDstDiskStr = 16,
    vsDupWriteProtectedStr = 17,
    vsDupReadingStr = 18,
    vsDupWritingStr = 19,
    vsFormattingMediaStr = 20,
};

Messages

msgVSFormatVolume

Reformats an existing volume.

Takes P_VS_FORMAT_VOLUME, returns STATUS.

typedef struct VOL_FORMAT_VOLUME {
    OBJECT volumeRootDir;
    CHAR pVolumeName[nameBufLength];
    U16 reserved:13,
        noWarning:1,
        maxSize:1,
        withName:1;
    U32 reserved1;
    U32 reserved2;
} VOL_FORMAT_VOLUME, *P_VOL_FORMAT_VOLUME;
#define msgVSFormatVolume MakeMsg(clsVolSearch, 5)

The volumeRootDir must be the actual root of the volume to format and there cannot be any other handles open on the volume or an error will be returned. pVolumeName will be the initial name when the user is asked to provide a name or will be the name if the user is not asked to provide a name (controlled by the withName flag). The warning message can be controlled with the noWarning flag. And the choose a size interaction can be controlled with the maxSize flag.

Return Value

stsRequestNotSupported The volume does not support formatting.
**msgVSDuplicateVolume**

Copy an existing volume from one floppy disk to another floppy disk.
Takes dir/file handle of a volume, returns STATUS.

```c
#define msgVSDuplicateVolume MakeMsg(clsVolSearch, 6)
```

Return Value

- `stsRequestNotSupported`  The volume does not support duplicating.

**msgVSFormatMedia**

Formats unformatted media that does belong to any volume.
Takes block device object, returns STATUS.

```c
#define msgVSFormatMedia MakeMsg(clsVolSearch, 7)
```

Comments

This message is sent by the `BlockDeviceManager` when it receives a block device reset all and in the process discovers unformatted media on a device.

**msgVSUpdateVolumes**

Requests the `VolSearcher` to update all volumes.
Takes BOOLEAN, returns STATUS.

```c
#define msgVSUpdateVolumes MakeMsg(clsVolSearch, 8)
```

Comments

This message requests the `VolSearcher` to ask all volume classes to update their list of volumes. This may result in volumes being installed, removed, connected or disconnected. Interested parties should become observers of the `FileSystem` and look for `msgFSVolChanged` (see `fs.h`). The argument passed should be true to update all volumes.

This message can only be sent via `ObjectSendXXX`.

**msgVSFormatCompleteNotify**

Notifies observers of the `VolSearcher` that a format has completed.
Takes BOOLEAN, returns STATUS.

```c
#define msgVSFormatCompleteNotify MakeMsg(clsVolSearch, 20)
```

Comments

The argument passed to the observer indicates whether the format was successful or not. False would be returned if there was an error or if the format was cancelled.

**msgVSNameVolume**

Prompts user to name an unlabelled volume and adds new name.
Takes root dir handle of volume, returns STATUS.

```c
#define msgVSNameVolume MakeMsg(clsVolSearch, 9)
```

Comments

This message is used by volumes that have discovered unlabeled volumes. This message can only be sent via `ObjectPostXXX`.
Part 8 /
System Services
This file contains the API definition for the compose-text package.

This package is used to compose a text string that needs to have pieces inserted into it. The format of the strings makes it easy to internationalize and localize the text.

The functions described in this file are contained in SYSUTIL.LIB.

**Format Strings**

The format strings contain literal text and format codes. A format code starts with 'A', has a sequence of one or more digits in the middle, and a single letter at the end. The digits specify which argument to the function to use and the letter indicates the type of the argument. For instance, format code "A2s" indicates that the second argument should be inserted, and that the argument should be a string.

The following fills 'buffer' with the string "a B b A c":

```c
SComposeText(&buffer, &size, heap, "a "2s b "ls c", "A", "B");
```

The available argument types are:

- A: Literal 'A' character. E.g. use "AA" to put a A in a string.
- s: String.
- r: Resource ID of a string resource.
- l: Group number and indexed list resource ID for string list. This uses two arguments.
- d: U32 printed as a decimal number.
- x: U32 printed as a hexadecimal number.
- {: Singular/Plural word forms of the form "{islare}". When this argument type is used, the routine examines the specified argument. If its value is 1, the first string is used. Otherwise the second string is used.

The following code reads in a string from the TK group for a 'sample' project.

```c
SComposeText(&buffer, &size, heap, "The filled in string is "11.", resGrpTK, sampleListResId);
```

As an example of the '{' format code, the following code generates the first string if `numApples==1` and the second string if `numApples==5`.

```c
SComposeText(&buffer, &size, heap, "There "1{islare} "ld "1{apple|apples}.", numApples);

"There is 1 apple."
"There are 5 apples."
```

**Memory Management**

All of the procedures fill in a buffer with the generated string. There are two ways of supplying the buffer memory.

- You can supply a buffer pointer and buffer length. Do this by passing the pointer as *ppString, the length in *pLength, and a null heapId. If this technique is used, and the buffer is too small to hold the results, an error status is returned.
You can specify a heap from which memory will be allocated. Do this by passing in a valid heapId. You are obligated to free the memory when finished.

```c
#ifndef CMPSTEXT_INCLUDED
#define CMPSTEXT_INCLUDED
#endif
#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef RESFILE_INCLUDED
#include <resfile.h>
#endif
#include <stdarg.h>
```

## Common `#defines and Typedefs`

```c
#define ComposeTextMaxArguments 20 // Maximum number of parameters
```

### Functions

#### SComposeText

Composes a string from a format and arguments.

**Returns** STATUS.

**Function Prototype**

```c
STATUS CDECL SComposeText ( 
    PP_CHAR      ppString, 
    PU32         pLength, 
    OS_HEAP_ID   heap, 
    const P_CHAR pFormat,
    ...
); 
```

**Comments**

Copy the format argument into the output string, doing the appropriate substitutions for the format codes. See the section "Memory Management" for information on what values to use for the first three arguments.

#### VSComposeText

Composes a string from a format and a pointer to the argument list.

**Returns** STATUS.

**Function Prototype**

```c
STATUS CDECL VSComposeText ( 
    PP_CHAR      ppString, 
    PU32         pLength, 
    OS_HEAP_ID   heap, 
    const P_CHAR pFormat,
    va_list      argList
); 
```

**Comments**

This is the same as SComposeText except the arguments are passed as a pointer to a list. See the section "Memory Management" for information on what values to use for the first three arguments.
This file contains the API definition for fixed point arithmetic. The functions described in this file are contained in PENPOINT.LIB.

The API in this file is all function oriented.

```c
#ifndef GOMATH_INCLUDED
#define GOMATH_INCLUDED
#ifndef GO_INCLUDED
#include <go.h>
#endif
#endif
```

### Math Operation Error Codes

- `#define stsUnderflow` MakeStatus(clsGOMath, 1)
- `#define stsOverflow` MakeStatus(clsGOMath, 2)
- `#define stsMathInvOp` MakeStatus(clsGOMath, 3)
- `#define stsMathInvStrOp` MakeStatus(clsGOMath, 4)
- `#define stsMathEqual` MakeStatus(clsGOMath, 5)
- `#define stsMathFirstHigher` MakeStatus(clsGOMath, 6)
- `#define stsMathFirstLower` MakeStatus(clsGOMath, 7)
- `#define stsZeroDivide` MakeStatus(clsGOMath, 8)

// The following two values are used by the runtime.lib as ERRNO values
- `#define stsMathDomain` MakeStatus(clsGOMath, 9) // Argument too large
- `#define stsMathRange` MakeStatus(clsGOMath, 10) // Result too large

### Math Constants

- `#define GoFx0 ((FIXED) 0x00000000) // 0.0`
- `#define GoFx1 ((FIXED) 0x00010000) // 1.0`
- `#define GoFxMinus1 ((FIXED) 0xffff0000) // -1.0`

### Fixed-point Function Prototypes

**FxCmp**

Compares two FIXED.

Returns S16.

Function Prototype: S16 PASCAL FxCmp(FIXED a, FIXED b);

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>a &lt; b</td>
</tr>
<tr>
<td>0</td>
<td>a = b</td>
</tr>
<tr>
<td>1</td>
<td>a &gt; b</td>
</tr>
</tbody>
</table>
**FxAdd**

Adds two FIXED numbers, producing a FIXED.

Returns STATUS.

Function Prototype: `STATUS PASCAL FxAdd (FIXED a, FIXED b, P_FIXED pC);`

Return Value: `stsOverflow` The integer part of the result overflows a 16-bit signed.

**FxAddSC**

Macro form of FxAdd with no overflow detection.

Returns FIXED.

```
#define FxAddSC(_f1, _f2) ((FIXED)((_f1) + (_f2)))
```

**FxSub**

Subtracts two FIXED numbers, producing a FIXED.

Returns STATUS.

Function Prototype: `STATUS PASCAL FxSub (FIXED a, FIXED b, P_FIXED pC);`

Return Value: `stsOverflow` The integer part of the result overflows a 16-bit signed.

**FxSubSC**

Macro form of FxSub with no overflow detection.

Returns FIXED.

```
#define FxSubSC(_f1, _f2) ((FIXED)((_f1) - (_f2)))
```

**FxNegate**

Negates a FIXED.

Returns FIXED.

```
#define FxNegate(_f) ((FIXED)(-_f))
```

**FxMul**

Multiplies two FIXED numbers, producing a FIXED.

Returns STATUS.

Function Prototype: `STATUS PASCAL FxMul (FIXED a, FIXED b, P_FIXED pC);`

Return Value: `stsOverflow` The integer part of the result overflows a 16-bit signed.

**FxMulSC**

Multiplies two FIXED numbers returning the product.

Returns FIXED.

Function Prototype: `FIXED PASCAL FxMulSC (FIXED a, FIXED b);`

Comments: No overflow detection is performed.
FxMulInt
Multiplies a FIXED number by an S32, producing a FIXED.
Returns STATUS.

Function Prototype
STATUS PASCAL FxMulInt(FIXED a, S32 b, P_FIXED pC);
Return Value
stsOverflow The integer part of the result overflows a 16-bit signed.

FxMulIntSC
Multiplies a FIXED number by an S32, returning the FIXED product.
Returns FIXED.
#define FxMulIntSC(_a,_b) ((FIXED) (_a*_b))
Comments
No overflow detection is performed.

FxMulIntToInt
Multiplies a FIXED number by an S32, producing a rounded S32 product.
Returns STATUS.

Function Prototype
STATUS PASCAL FxMulIntToInt(FIXED a, S32 b, P_S32 pC);
Return Value
stsOverflow The integer part of the result overflows a 32-bit signed.

FxMulIntToIntSC
Multiplies a FIXED number by an S32, returning a rounded S32 product.
Returns S32.

Function Prototype
S32 PASCAL FxMulIntToIntSC(FIXED a, S32 b);
Comments
No overflow detection is performed.

FxDiv
Divides two FIXED numbers, producing a FIXED quotient.
Returns STATUS.

Function Prototype
STATUS PASCAL FxDiv(FIXED top, FIXED bottom, P_FIXED pC);
Return Value
stsOverflow The integer part of the result overflows a 16-bit signed.
stsZeroDivide The input divisor is zero.

FxDivSC
Divides two FIXED numbers, returning a FIXED quotient.
Returns FIXED.

Function Prototype
FIXED PASCAL FxDivSC(FIXED top, FIXED bottom);
Comments
No overflow or zero-divide detection is performed.
FxDivInts
Divides two 32-bit signed integers, producing a FIXED quotient.
Returns STATUS.

Function Prototype
STATUS PASCAL FxDivInts(S32 top, S32 bottom, P_FIXED pC);

Return Value
stsOverflow  The integer part of the result overflows a 16-bit signed.
stsZeroDivide  The input divisor is zero.

FxDivIntsSC
Divides two FIXED numbers, returning a FIXED quotient.
Returns FIXED.

Function Prototype
FIXED PASCAL FxDivIntsSC(S32 top, S32 bottom);

Comments
No overflow or zero-divide detection is performed.

FxDivIntToInt
Divides an S32 by a FIXED, producing a rounded S32 quotient.
Returns STATUS.

Function Prototype
STATUS PASCAL FxDivIntToInt(S32 top, FIXED bottom, P_S32 pC);

Return Value
stsOverflow  The integer part of the result overflows a 16-bit signed.
stsZeroDivide  The input divisor is zero.

FxDivIntToIntSC
Divides an S32 by a FIXED, producing a rounded S32 quotient.
Returns S32.

Function Prototype
S32 PASCAL FxDivIntToIntSC(S32 top, FIXED bottom);

Comments
No overflow or zero-divide detection is performed.

FxSin
Returns the sine of an integer angle in degrees.
Returns FIXED.

Function Prototype
FIXED PASCAL FxSin(S16 angle);

FxCos
Returns the cosine of an integer angle in degrees.
Returns FIXED.

Function Prototype
FIXED PASCAL FxCos(S16 angle);
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FxTan</strong></td>
<td>Returns the tangent of an integer angle in degrees.</td>
<td>FIXED PASCAL FxTan(S16 angle);</td>
</tr>
<tr>
<td><strong>FxSinFx</strong></td>
<td>Returns the sine of a FIXED angle in degrees.</td>
<td>FIXED PASCAL FxSinFx(FIXED angle);</td>
</tr>
<tr>
<td><strong>FxCosFx</strong></td>
<td>Returns the cosine of a FIXED angle in degrees.</td>
<td>FIXED PASCAL FxCosFx(FIXED angle);</td>
</tr>
<tr>
<td><strong>FxTanFx</strong></td>
<td>Returns the tangent of a FIXED angle in degrees.</td>
<td>FIXED PASCAL FxTanFx(FIXED angle);</td>
</tr>
<tr>
<td><strong>FxArcTanInt</strong></td>
<td>Returns an arctangent value as a FIXED angle.</td>
<td>FIXED PASCAL FxArcTanInt(S32 top, S32 bottom);</td>
</tr>
<tr>
<td><strong>FxArcTanFx</strong></td>
<td>Returns an arctangent value as a FIXED angle.</td>
<td>FIXED PASCAL FxArcTanFx(S32 top, S32 bottom);</td>
</tr>
<tr>
<td><strong>FxAbs</strong></td>
<td>Takes the absolute value of a FIXED.</td>
<td>#define FxAbs(_f) (((_f)&lt;0)?FxNegate(_f):(_f))</td>
</tr>
</tbody>
</table>
**FxRoundToInt**
Rounds a FIXED number to a 32-bit signed integer.
Returns S32.

Function Prototype: `S32 PASCAL FxRoundToInt(FIXED fx);`

**FxRoundToIntSC**
Rounds a FIXED number to a 16-bit signed integer.
Returns S16.

*define FxRoundToIntSC(_f) (S16)(((_f)+0x8000)>>16)*

Comments:
No overflow detection is performed.

**FxChop**
Returns the 16-bit signed integer part of a FIXED.
Returns S16.

*define FxChop(_f) (S16)((_f)>>16)*

**FxFraction**
Returns the 16-bit fractional part of the absolute value a FIXED.
Returns U16.

*define FxFraction(_f) (U16)(FxAbs(_f))*

**FxIntToFx**
Converts a 16-bit signed integer into a FIXED.
Returns FIXED.

*define FxIntToFx(_i) ((FIXED)((S32)(_i)<<16))*

**FxMakeFixed**
Makes a FIXED with an S16 (integer) and a U16 (fraction).
Returns FIXED.

FIXED PASCAL FxMakeFixed(S16 whole, U16 frac); (now in go.h)

**FxBinToStr**
Converts a FIXED format value into an ascii string in decimal.
Returns nothing.

Function Prototype: `void PASCAL FxBinToStr(FIXED a, P_CHAR pStr, U8 fracDigits, U8 maxLen, BOOLEAN showCommas);`
The string will have the format:
{-}xxxx.xxxxx or {-}xx.xxx.xxxxx.
The number of digits to the left of the decimal point is the minimum number required, and the number of digits to the right of the decimal point is specified in fracDigits. The last digit is rounded accurately. If the string will not fit within maxLen bytes, then the string "*******" (maxLen-1 "s) will be returned; maxLen = 9+fracDigits is sufficient, although any higher number is also acceptable. If showCommas is true, then commas will separate the thousands.

FxStrToBin
Converts a null-terminated ascii string to a FIXED.
Returns STATUS.

Function Prototype
STATUS PASCAL FxStrToBin(
P_CHAR pStr,
P_FIXED pC
);

Comments
The fractional portion will be rounded to fit within 16 bits.

Return Value
stsOverflow The integer part of the result overflows a 16-bit signed.
stsMathInvStrOp A character in the string does not represent a valid number. *pC is set to zero.
Definitions used while internationalizing code.

The main content of this file is macros that map the names of UNICODE string functions for PENPOINT 2.0 to the 8-bit functions used currently. They are intended to be used with items of type CHAR, which are 8-bit currently and will switch to 16-bit in 2.0. By using these macros code that deals with strings will have a chance of working in 2.0 with only a recompile.

```c
 ifndef INTL_INCLUDED
 define INTL_INCLUDED

 UNICODE strings/characters

To define characters or strings in PENPOINT 1.0, use the "U_L" macro on them. This maps to the original string, and thus does nothing. In 2.0 the define will be changed so that it inserts "L" in front of the string. This will convert the character or string into a wide character or string to match the 2.0 definition of CHAR.

Here is some sample code to show its use. This code would compile and run under both 1.0 and 2.0, the only difference would be the space allocated for each character (1 vs. 2 bytes).

```c
 CHAR cc;
P_CHAR pString;

 pString = U_L("sample string");
 cc = U_L(’s’);

 if (cc == pString[0])
  pString[0] = U_L(’S’);
```

```c
#define U_L(str) str
// #define U_L(str) L$$str // Definition to be used in PENPOINT 2.0
```

Mapping of 16-bit string/character functions for 1.0

For each of the sections below, it is necessary to include the base header file in order to use the macros defined here.

These macros are intended to be used with variables of type CHAR. CHAR is currently U8, and will be converted to U16 in PENPOINT 2.0.
**Extensions to STRING.H**

- `#define Ustrcat` `strcat`
- `#define Ustrncat` `strncat`
- `#define Ustrncmp` `strncmp`
- `#define Ustrncpy` `strncpy`
- `#define Ustrcpy` `strcpy`
- `#define Ustrndup` `strndup`
- `#define Ustrrev` `strrev`
- `#define Ustrset` `strset`
- `#define Ustrnset` `strnset`
- `#define Ustrchr` `strchr`
- `#define Ustrrchr` `strrchr`
- `#define Ustrstr` `strstr`
- `#define Ustrtok` `strtok`
- `#define Ustrdup` `strdup`
- `#define Ustrerror` `strerror`
- `#define Ustrlen` `strlen`

'`strcmpi' the same as 'stricmp', we don't need U versions of both.

- `#define Ustrnicmp` `strnicmp`
- `#define Ustrlwr` `strlwr`
- `#define Ustrupr` `strupr`
- `#define Umemcpy` `memcpy`
- `#define Umemccpy` `memccpy`
- `#define Umemchr` `memchr`
- `#define Umemcmp` `memcmp`
- `#define Umemicmp` `umemicmp`
- `#define Umemmove` `memmove`
- `#define Umemset` `memset`
- `#define Umemmove` `memmove`

**Extensions to CTYPE.H**

- `#define Uisalpha` `isalpha`
- `#define Uisalnum` `isalnum`
- `#define Uisascii` `isascii`
- `#define Uiscntrl` `iscntrl`
- `#define Uisprint` `isprint`
- `#define Uisgraph` `isgraph`
- `#define Uisdigit` `isdigit`
- `#define Uisxdigit` `isxdigit`
- `#define Uislower` `islower`
- `#define Uisupper` `isupper`
- `#define Uisspace` `isspace`
- `#define Uispunct` `ispunct`
- `#define Utolower` `tolower`
- `#define Utoupper` `toupper`

**Extensions to STDLIB.H**

- `#define Uatoi` `atoi`
- `#define Uatol` `atol`
- `#define Uitoa` `itoa`
- `#define Uitoa` `itoa`
- `#define Utoa` `utoa`
- `#define Ustrlen` `strlen`
- `#define Ustrtol` `strtol`
- `#define Uatof` `atof`
- `#define Ustrtoc` `strtoc`
- `#define Ustrtohl` `strtohl`
This goes directly to its 2.0 definition because it does not make sense on an ascii text stream, and if the current text is not ascii, then having it automatically convert to Unicode by recompile in 2.0 won't work. It is included mostly to reserve the name, and let programmers know that it will be available.

#define Uswab(s,d,n) swab((char *)s, (char *)d, n*2)

Extensions to STDIO.H

#define Ufopen fopen
#define Usprintf sprintf
#defineUvsprintf vsprintf
#define Usscanf scanf
#define Uputc putc
#define Ufputc fputc
#define Ugetc getc
#define Ufgetc fgetc
#define Uungetc ungetc
#define Ufdopen fopen
#define Ufreopen recompile
#define Uprintf printf
#define Ufprintf fprintf
#define Uvprintf vprintf
#define Uputc.printf
#define Uopen open
#define Ucreat creat
#define Uasctime asctime
#define Uctime ctime

Extensions to FCNTL.H

#define Uopen open
#define Usopen sopen
#define Ucreat creat

Extensions to TIME.H

#define Uasctime asctime
#define Uctime ctime

Extensions to UNISTD.H

#define Urmkdir rmdir
#define Uchdir chdir
#define Ugetcwd getcwd
#define Uopendir opendir
#define Ureaddir readdir
This file contains the API for the PenPoint kernel. The functions described in this file are contained in PENPOINT.LIB.

The PenPoint kernel provides support for tasking, memory management, inter-task communication and timer services.

```c
#ifndef OS_INCLUDED
#define OS_INCLUDED
#endif

/* Debugging Flags */
PenPoint kernel flag is 'G', values are:

0001 User configuration (copy exes from boot to theSelectedVolume)
0002 Enter debugger on faults while scavenging
0004 Display memory sizes for each module loaded and run
0008 Display Stack grow/shrink messages
0010 Save page fault information in a memory buffer
0020 Run in the Ram only configuration
0100 Print various memmgr details
1000 see resfile.h
2000 see resfile.h
4000 see resfile.h
8000 see resfile.h
10000 Internal use only
20000 Call the MIL using the common entry point for full debugging

#ifndef GO_INCLUDED
#include <go.h>
#endif
#endif
#endif
#endif
```

```c
Common #defines and typedefs
#define osPageSize (4*1024)

Defines for OS_ITMSG_INFO (mode field)

// To generate the mode, OR in OS_TASK_MODE with the defines below.
#define osITMsgNoCopy flag7 // vs copy buffer
#define osITMsgFrontOfQ flag6 // vs end of queue
#define osITMsgDefaultMode 0 // Copy msg to end of msg queue
```
• Defines for setting priority

#define osNumPriorities 51
#define osDefaultPriority 0

• Defines for region information

typedef U8 OS_REGION_ATTRS;
#define osRgnLocal flag0
#define osRgnHasAliases flag1
#define osRgnLocked flag2 // Not yet implemented!!
#define osRgnNotSwappable flag3
#define osRgnFrozen flag4 // Not yet implemented!!
#define osRgnInSlowMem flag5

Enum16(OS_REGION_TYPE) {
    osRgnData, // data region
    osRgnHeap, // heap region
    osRgnStack, // stack region
    osRgnMemMapFile, // memory mapped file region
    osRgnCode // code region
};

• Subtask function type

typedef void FunctionPtr(P_OS_SUBTASK_ENTRY) (U32 arg);
Enum16(OS_SET_GET) {
    osValuesSet = flag0, // Set the value(s) passed in
    osValuesReturn = flag1, // return the value(s)
    osValuesReturnAndSet = flag0 | flag1 // return and set the value(s)
};

• Memory access attributes

Enum16(OS_ACCESS) { // access rights of a page
    osReadAccess, // page allows read access only
    osReadWriteAccess, // page allows read and write access
    osExecuteAccess, // page allows execute access only
    osExecuteReadWrite // page allows execute and read access
};

Enum16(OS_SET_TIME_MODE) {
    osSetTime = flag0, // set the time
    osSetDate = flag1, // set the date
    osSetTimeZone = flag2, // set only the time zone
    osSetDateAndTime = osSetTime|osSetDate, // set both the date and time
    // set date, time, and time zone
    osSetAll = osSetTime|osSetDate|osSetTimeZone
};

• Display modes

Enum16(OS_DISPLAY_MODE) {
    osConsole, // display mode is console
    osGraphics // display mode is graphics
};

• Beep error tones

Enum16(OS_ERROR_TYPE) {
    osWarning,
    osFatal
};
typedef struct OS_MEM_INFO {
    U32 taskMemAllocated; // amt of mem allocated by the task
    U32 localTaskMemAllocated; // amt of local mem allocated by the task
    U16 numAllocatedRgns; // # allocated regions by the task
    U16 numAllocatedLocalRgns; // # local regions allocated
    U32 taskMemResident; // amt of allocated mem in ram-this task
    U32 taskMemSwapped; // amt of allocated mem in swap file-this task
    U32 systemRamSize; // total amt of memory in the system
    U32 amtInMemoryPool; // amt of memory in the memory pool
    U32 memFree; // amt of free ram
    U32 memAllocated; // amt of mem allocated by all
    U16 numRgnsAllocated; // total # regions allocated by all
    U16 numSharedRgnsAllocated; // # shared regions used by all
    U32 pageSize; // system page size
    U32 memNotSwappable; // amt of memory not swappable
    U32 swapFileSize; // size of the swap file
    U32 swapMediaFreePages; // number of pages free on the swap media
    U32 dataAllocated; // amt of data allocated
    U32 heapsAllocated; // amt of heap space allocated
    U32 stacksAllocated; // amt of stack space allocated
    U32 memMapFilesAllocated; // amt of mem map file space allocated
    U32 codeAllocated; // amt of code space allocated
} OS_MEM_INFO, *P_OS_MEM_INFO;

typedef struct OS_REGTYPE_INFO {
    U32 allocated; // Max size of the region
    U32 swappable; // swappable pages in memory
    U32 nonSwappable; // non-swappable pages in memory
    U32 committed; // committed pages
} OS_REGTYPE_INFO, *P_OS_REGTYPE_INFO;

typedef struct OS_REGSCOPE_INFO {
    OS_REGTYPE_INFO code; // Executable code
    OS_REGTYPE_INFO data; // Data
    OS_REGTYPE_INFO heap; // Data used as heaps
    OS_REGTYPE_INFO stack; // Stack space
    OS_REGTYPE_INFO memMapFile; // Memory-mapped files
} OS_REGSCOPE_INFO, *P_OS_REGSCOPE_INFO;

typedef struct OS_MEM_USE_INFO {
    OS_REGSCOPE_INFO local; // Owned by this task only, in local memory
    OS_REGSCOPE_INFO shared; // Owned by this task only, in shared memory
    OS_REGSCOPE_INFO multiOwner; // Owned by this task and at least one other
    OS_REGSCOPE_INFO total; // System-wide totals
    U32 pageSize; // System page size
    U32 systemRamSize; // total amt of memory in the system
    U32 memFree; // mem in the "free" list
    U32 memAllocated; // mem not in the "free" list
    U32 swapFileSize; // size of the swap file
} OS_MEM_USE_INFO, *P_OS_MEM_USE_INFO;
• Address information

typedef struct OS_ADDRESS_INFO { // Info for a given memory address
    PMEM pRegionBase; // base of region
    SIZEOF regionLength; // length of the region
    OS_ACCESS access; // access rights of the region
    OS_TASK_ID owner; // owning task for this region
    BOOLEAN userPriv; // TRUE - user region, FALSE - kernel
    OS_REGION_ATTRS flags; // see defines above
    SIZEOF residentSize; // amount of region that is resident
    SIZEOF committedSize; // amount of region that is committed
    OS_REGION_TYPE regionType; // type of region
} OS_ADDRESS_INFO, * P_OS_ADDRESS_INFO;

• System configuration information

typedef struct OS_SYSTEM_INFO { // system configuration information
    BOOLEAN mathProcessorPresent; // TRUE = present
    OS_MILLISECONDS millisecondsPerSystick; // ms per clock tick
} OS_SYSTEM_INFO, * P_OS_SYSTEM_INFO;

• Date and time information

// The time zone string is a POSIX format string. See the Watcom library
// reference for PenPoint, TZ environment variable set section for more info.
typedef struct OS_DATE_TIME { // struct used to set time
    U32 seconds; // seconds after the minute -- [0,61]
    U32 minutes; // minutes after the hour -- [0,59]
    U32 hours; // hours after midnight -- [0,23]
    U32 day; // day of the month -- [1,31]
    U32 month; // months since January -- [0,11]
    U32 year; // years since 1900
    U32 dayOfWeek; // days since Sunday -- [0,6]
    U32 dayOfYear; // days since January 1 -- [0,365]
    PCHAR pTimeZone; // time zone string (POSIX format)
} OS_DATE_TIME, * P_OS_DATE_TIME;

• Loaded program information

typedef struct OS_PROG_INFO { // program identifying handle
    OS_PROG_HANDLE progHandle; // module name (without the .exe)
    CHAR name[32+1]; // exe-header initial heap allocation
    U32 initHeapSize; // .exe-header initial stack allocation
    U32 initStackSize; // initial CS (selector, not segment#)
    U32 initIP; // initial IP
    U32 initCS; // initial DS
    U16 isDLL; // 0 for .exes, 1 for DLLs
    isUser; // 1 for user priv, 0 for system priv
    rsvd; // reserved for future use.
    U32 fixedSize; // read-only segments + initialization data
    U32 sharedSize; // shared read/write segments
    U32 privateSize; // private read/write segments
    U32 nRequiredModules; // # modules this depends upon
} OS_PROG_INFO, * P_OS_PROG_INFO;

• Interrupt information

// Note: OR in the flag osIntNumIsHardwareLevel if intNum is a hardware
// interrupt level (vs a MIL logical device id). The flag is defined
// in ostypes.h.
typedef struct OS_INTERRUPT_INFO { // struct used to set interrupts
    OS_INTERRUPT_ID intNum; // logical interrupt id
    P_UNKNOWN pCode; // ptr to interrupt routine
} OS_INTERRUPT_INFO, * P_OS_INTERRUPT_INFO;
Functions

**OSProgramInstall**
Installs a program into the loader database.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED OSProgramInstall(
    P_CHAR pCommandLine,  // dlc or exe name (and arguments)
    P_CHAR pWorkingDir,   // working dir of the program
    P_OS_PROG_HANDLE pProgHandle,  // Out: program handle
    P_CHAR pBadName,     // Out: If error, dll/exe that was bad
    P_CHAR pBadRef      // Out: If error, reference that was bad
);
```

Comments

If a dlc file is provided, all dlls in the file will also be loaded if not loaded already.

OSProgramInstall will not return until instance 0 of all loaded dlls and exe are completed. No message dispatching will occur during this time. If communication to the calling task is required, use IMProgramInstall (install.h, install.lib).

See Also

OSProgramDeinstall

Return Value

- **stsOSBadDLCFormat**  DLC file is incorrectly formatted
- **stsOSBadExeFormat**  A DLL or EXE is invalid in the dlc file
- **stsOSProgInstallError**  Use debug version of PenPoint for more info
- **stsOSModuleNotFound**  Module name specified in dlc file is invalid
- **stsOSMissingDependency**  Import module in an exe or dll was not found
- **stsOSMissingEntryName**  Import name in an exe or dll was not found
- **stsOSMissingEntryOrdinal**  Import number in an exe or dll was not found
**OSProgramDeinstall**

Deinstalls a program already loaded into the loader database.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED OSProgramDeinstall(
    OS_PROG_HANDLE progHandle    // program handle
);
```

Comments

This routine will terminate any dll task wrappers before deinstalling the code. If an exe is being deinstalled, all tasks must be terminated before calling this routine.

See Also

OSProgramInstall

Return Value

- **stsOSInvalidProgramHandle**  Program handle is incorrect
- **stsOSDependenciesExist**  Another program requires this dll or a task is using this module

**OSProgramInstantiate**

Creates an instance of a program.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED OSProgramInstantiate(
    OS_PROG_HANDLE progHandle,     // program handle from install
    P_CHAR pCommandLine,           // pathname + arguments
    P_OS_TASK_ID pTaskId           // Out: Task id of the new task
);
```

Comments

The newly created process will be set to the same priority as the caller.

Return Value

- **stsBadParam**  Program handle is invalid

**OSSubTaskCreate**

Creates a new execution thread in this context.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED OSSubTaskCreate(
    P_OS_SUBTASK_ENTRY pEntrypoint,      // Function entrypoint
    SIZEOF stackSize,                   // ignored.
    U16 mustBeZero,                     // reserved
    U32 arg,                            // arg passed to function
    P_OS_TASK_ID pTaskId               // Out: new task id
);
```

Comments

The entrypoint that starts the subtask must NOT return. To terminate the task, use OSTaskTerminate (OSThisTask ()) as the last line in the routine. The newly created task will be set to the same priority as the caller.

The initial stack size of the subtask will be set to 4096 bytes. The `stackSize` parameter will be ignored. Stacks will automatically grow to accommodate a program's stack requirements.

**OSTaskTerminate**

Terminates a task.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED OSTaskTerminate(
    OS_TASK_ID taskId,                  // task to terminate
    OS_TASK_ERROR exitCode              // reason for terminating exit code
);
```
Functions

Callers to OSTaskTerminate will not return until the task has successfully terminated. Task termination will cause the following events to occur:

1) if a process is terminated, all subtasks are first terminated
2) observers of the Process will be notified (see clsmgr.h). The error code is provided with the notification.
3) objects owned by the terminating task will be scavenged
4) a broadcast message will be sent to all tasks to notify them of the task termination. The message will be sent on the filter osTerminatedTaskFilter. This filter is by default off.

OSNextTerminatedTaskId
Notifies the caller of the tasks that have terminated.

Returns the next task that has terminated.

Function Prototype

OS_TASK_ID EXPORTED OSNextTerminatedTaskId(
    const P_OS_TASK_ERROR pExitCode,  // Out: exit code of terminating task
);

Comments

The broadcast message for task termination does not include the task identifier of the task that has terminated. To find this out, this routine should be called to get the list of terminated tasks. When osNullTaskId is returned, the list ends.

OSThisTask

Passes back the task identifier of the current running task.

Returns OS_TASK_ID.

Function Prototype

OS_TASK_ID EXPORTED OSThisTask(void);

OSTaskPrioritySet

Sets the priority of a task or a set of tasks.

Returns STATUS.

Function Prototype

STATUS EXPORTED OSTaskPrioritySet(
    OS_TASK_ID taskId,            // target task
    OS_TASK_MODE mode,           // task mode
    OS_PRIORITY_CLASS priorityClass,  // new priority class
    U8 priority                   // new priority number
);

Comments

The task mode can be used to set the priority of just one task or all tasks in the process family.

See Also

OSTaskPriorityGet

OSTaskPriorityGet

Passes back the priority of a task.

Returns STATUS.

Function Prototype

STATUS EXPORTED OSTaskPriorityGet(
    OS_TASK_ID taskId,            // target task
    P_OS_PRIORITY_CLASS pPriorityClass,  // Out: task’s priority class
    P_U8 pPriority                 // Out: task’s priority number
);
Both the priority class and the priority within that class are returned.

OSTaskPrioritySet

OSTaskDelay

Delays the current task for a specified period of time.

Returns STATUS.

Function Prototype:
```c
void EXPORTED0 OSTaskDelay(
    OS_MILLISECONDS timeLimit  // milliseconds to delay
);
```

Comments:
When the machine is turned off, the delay time freezes until the system is turned back on again.
OSTaskDelay cannot be called from an interrupt subtask.

OSITMsgSend

Sends an inter-task message to a task or set of tasks.

Returns STATUS.

Function Prototype:
```c
STATUS EXPORTED0 OSITMsgSend(
    P_OS_ITMSG_INFO pITMsgInfo
    // inter-task message info block
);
```

Comments:
OSTITMsgSend is used to send an inter-task message to 1) a single task, or 2) all tasks in a task family, or 3) all tasks in the system. The combination of the taskId and mode fields are used to accomplish this. If broadcasting to all tasks, the taskId field is ignored.

An inter-task message is an array of bytes completely uninterpreted by the kernel stored in the pITMsg field. If the inter-task message is short (up to U32), it can be stored in the token field for improved performance. The length field is used to store the length of the inter-task message in pITMsg. If the length field is 0, the pITMsg field is ignored and can be used for more information passing.

Inter-task messages are passed to the destination task in two ways: copy and alias. In copy mode, the message is copied into a new buffer allocated in the context of the destination task. In alias mode, the message is aliased into the destination task. Messages must be full regions when using alias mode.

Messages are normally inserted into the end of the destination message queue. However, it is possible to specify that a message be inserted into the front of the message queue.

Inter-task messages will get delivered to tasks that have a filter mask set to allow messages of the sending messages filter. If sending a message on multiple filters, the message will be delivered if any one of the filters are allowed by the receiving task. No error status is returned if the receiving task does not receive the message due to its filter mask setting.

See Also:
OSITMsgReceive

OSITMsgReceive

Receives a message from the task's message queue.

Returns STATUS.

Function Prototype:
```c
STATUS EXPORTED0 OSITMsgReceive(
    P_OS_ITMSG_INFO pITMsgInfo,
    // In-Out: message info block
    OS_MILLISECONDS timeLimit  // amount of time to wait for message
);
```
Messages are received by specifying a filter or set of filters in the pITMsgInfo struct. Any message with a filter that is in that set will match the receive request. The filter in the pITMsgInfo struct must always be set on entry.

When a message is received that matches the input filter, the message is removed from the queue and provided to the client.

See Also OSITMsgSend

---

**OSITMsgPeek**

gets the next message from the message queue without removing it.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED OSITMsgPeek(
    P_OS_ITMSG_INFO pITMsgInfo,       // In-Out: message info block
    OS_MILLISECONDS timeLimit,        // amount of time to wait for message
    P_OS_ITMSG_ID pITMsgId           // In-Out: id of message received
);
```

Comments

*pITMsgId of null peeks from the front of the queue. Use the previous message id to peek further into the queue. The filter in the pITMsgInfo struct must always be set on entry.

See Also OSITMsgFromId

---

**OSITMsgFromId**

Passes back the message associated with the message identifier.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED OSITMsgFromId(
    P_OS_ITMSG_INFO pITMsgInfo,       // In-Out: message info block
    OS_ITMSG_ID itMsgId               // message id obtained from OSITMsgPeek
);
```

Comments

The message identifier should be obtained by calling OSITMsgPeek.

See Also OSITMsgPeek

---

**OSITMsgQFlush**

Flushes the message queue of all messages matching the message filter.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED OSITMsgQFlush(
    OS_ITMSG_FILTER itMsgFilter       // message filter of messages to flush
);
```

Comments

If a message has other filters set in addition to itMsgFilter, then the message will NOT be flushed.

---

**OSITMsgFilterMask**

Sets the filter mask for this task.

Returns the old filter mask.

Function Prototype

```c
OS_ITMSG_FILTER EXPORTED OSITMsgFilterMask(
    OS_ITMSG_FILTER newITMsgFilter,     // new filter mask for this task
    BOOLEAN setNewFilter               // if true, the new filter mask will be set
);
```
Setting the mask bit to 1 indicates the message is allowed by this task; 0 otherwise. Any messages sent to this task whose filter bits are off in the filter mask will be discarded.

If `setNewFilter` is FALSE, `newITMsgFilter` is ignored and only the old filter mask is returned.

See Also

OSITMsgSend

---

**OSSemaCreate**

Creates a semaphore.

Returns STATUS.

Function Prototype

```
STATUS EXPORTED OSSemaCreate(
    P_OS_SEMA_ID pSema      // Out: new open semaphore
);
```

Comments

The semaphore will automatically be opened for the process.

See Also

OSSemaOpen

---

**OSSemaOpen**

Opens (accesses) an already existing semaphore.

Returns STATUS.

Function Prototype

```
STATUS EXPORTED OSSemaOpen(
    OS_SEMA_ID sema,        // semaphore
    OS_TASK_ID task         // task wanting to share ownership of sema
);
```

Comments

Tasks should always open someone else’s semaphore to guarantee that the semaphore will be around even if the original owner of the semaphore terminates.

See Also

OSSemaCreate

---

**OSSemaDelete**

Deletes a semaphore.

Returns STATUS.

Function Prototype

```
STATUS EXPORTED OSSemaDelete(
    OS_SEMA_ID sema      // the semaphore to delete
);
```

Comments

The semaphore will be removed from the system when all owners of the semaphore have deleted it.

See Also

OSSemaCreate

---

**OSSemaRequest**

Locks the counting semaphore (increments the count).

Returns STATUS.

Function Prototype

```
STATUS EXPORTED OSSemaRequest(
    OS_SEMA_ID sema,        // the semaphore to lock
    OS_MILLISECONDS timeLimit // max time to wait if already locked
);
```

Comments

OSSemaRequest should be used in conjunction with OSSemaClear when using semaphores to protect critical sections of code. OSSemaRequest/OSSemaClear implement a counting semaphore model which
allows nesting of OSSemaRequest calls. Only after the same number of OSSemaClear calls will the next waiting task enter the critical section. Up to 64K nestings are allowed.

If a task has obtained a semaphore via OSSemaRequest and subsequently dies, the semaphore will be given to the next requestor and that requestor will be given the status stsOSSemaLockBroken.

Return Value

stsOSSemaLockBroken Previous locker of semaphore died without clearing the semaphore

stsOSTimeOut The timelimit expired before obtaining the semaphore

See Also OSSemaClear

OSSemaClear
Unlocks the counting semaphore (decrements the count).

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED0 OSSemaClear(
    OS_SEMA_ID sema   // the semaphore to unlock
);
```

Comments

OSSemaClear should be used in conjunction with OSSemaRequest when using semaphores to protect critical sections of code. OSSemaRequest/OSSemaClear implement a counting semaphore model which allows nesting of OSSemaRequest calls. Only after the same number of OSSemaClear calls will the next waiting task enter the critical section. Up to 64K nestings are allowed.

See Also OSSemaRequest

OSSemaReset
Resets event semaphore (no matter what count).

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED0 OSSemaReset(
    OS_SEMA_ID sema   // the semaphore to reset
);
```

Comments

OSSemaReset is used with OSSemaSet and OSSemaWait to support event handling. In this model, the client waiting on the event should use OSSemaSet to set the semaphore to 1, and OSSemaWait to wait until the semaphore has been reset to 0. OSSemaReset will reset the semaphore to 0, thereby notifying all tasks waiting on the event. OSSemaReset is normally used in interrupt tasks. The task that is processing the event may actually have received more than one event and should process all events after resetting the semaphore to avoid losing any events.

See Also OSSemaSet

OSSemaSet
Sets the event semaphore to 1.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED0 OSSemaSet(
    OS_SEMA_ID sema   // the semaphore to set
);
```

Comments

OSSemaSet is used with OSSemaWait and OSSemaReset to support event handling. In this model, the client waiting on the event should use OSSemaSet to set the semaphore to 1, and OSSemaWait to wait
until the semaphore has been reset to 0. OSSemaReset will reset the semaphore to 0, thereby notifying the task waiting on the event.

See Also
OSSemaReset

OSSemaWait
Waits for the event semaphore to be reset.
Returns STATUS.

Function Prototype
STATUS EXPORTED OSSemaWait(
    OS_SEMA_ID sema,       // the semaphore to wait on
    OS_MILLISECONDS timeLimit // max time to wait for the count to go to 0
);

Comments
OSSemaWait is used with OSSemaSet and OSSemaReset to support event handling. In this model, the client waiting on the event should use OSSemaSet to set the semaphore to 1, and OSSemaWait to wait until the semaphore has been reset to 0. OSSemaReset will reset the semaphore to 0, thereby notifying the task waiting on the event.

Return Value
stsOSSemaLockBroken  Previous locker of semaphore died without clearing the semaphore
stsOSTimeOut The timelimit expired before obtaining the semaphore

See Also
OSSemaReset

OSFastSemaInit
Initialize fast sema.
Returns nothing..

#define OSFastSemaInit (pSem) memset (_pSem), 0, sizeof(OS_FAST_SEMA)

Comments
Fast semaphores provide a fast but unprotected semaphore model. Fast semaphores are simply memory provided by the client as storage area for the state of the semaphore. This storage area must initially be set to 0.

See Also
OSFastSemaRequest

OSFastSemaRequest
Fast version of sema request.
Returns STATUS.

Function Prototype
STATUS EXPORTED OSFastSemaRequest (
    P_OS_FAST_SEMA pSema
);

Comments
OSFastSemaRequest should be used in conjunction with OSFastSemaClear when using semaphores to protect critical sections of code. OSFastSemaRequest/OSFastSemaClear implement a counting semaphore model which allows nesting of OSFastSemaRequest calls. Only after the same number of OSFastSemaClear calls will the next waiting task enter the critical section. Up to 64K nestings are allowed.

Fast semaphores are fast by sacrificing protection. The semaphore structure passed into this routine is modified in the same privilege level as the caller. Only if another task owns the semaphore will a privilege level transition occur.
There are a number of important limitations that a developer should understand about fast semaphores.
1) If a task owns a fast semaphore and then dies before releasing it, the semaphore will not be released automatically by the system.
2) The fast semaphores should not be copied from one location to another.
The routines rely on the address of the semaphore structure being the same.

**See Also**
OSFastSemaClear

### OSFastSemaClear

Fast version of sema clear.
Returns STATUS.

**Function Prototype**

```
STATUS EXPORTED OSFastSemaClear (  
P_OS_FAST_SEMA pSema  
);  
```

**Comments**
OSFastSemaClear should be used in conjunction with OSFastSemaRequest when using semaphores to protect critical sections of code. OSFastSemaRequest/OSFastSemaClear implement a counting semaphore model which allows nesting of OSFastSemaRequest calls. Only after the same number of OSFastSemaClear calls will the next waiting task enter the critical section. Up to 64K nestings are allowed.

Fast semaphores are fast by sacrificing protection. The semaphore structure passed into this routine is modified in the same privilege level as the caller. Only if another task is waiting on the semaphore will a privilege level transition occur.

There are a number of important limitations that a developer should understand about fast semaphores.
1) If a task owns a fast semaphore and then dies before releasing it, the semaphore will not be released automatically by the system.
2) The fast semaphores should not be copied from one location to another.
The routines rely on the address of the semaphore structure being the same.

**See Also**
OSFastSemaRequest

### OSGetTime

Returns local time.
Returns STATUS.

**Function Prototype**

```
STATUS EXPORTED OSGetTime (  
SIZEOF structLength,  
P_OS_DATE_TIME pDateTime  
);  
```

**Comments**
If an error is returned, the time returned will be Jan 1, 1900.
**OSSetTime**
Sets the time or time zone.
Returns STATUS.

Function Prototype
```c
STATUS EXPORTED OSSetTime(
    OS_SET_TIME_MODE setMode, // which attributes to set
    SIZEOF structLength, // size of the date/time struct
    P_OS_DATE_TIME pDateTime // date, time and time zone information
);
```

**OSProgramInfo**
Returns information on the program from the loader.
Returns STATUS.

Function Prototype
```c
STATUS EXPORTED OSProgramInfo(
    OS_PROG_HANDLE progHandle, // program handle given out by the loader
    P_OS_PROG_INFO pInfo // Out: information buffer
);
```
OSProgramInfo will return information on the program handle passed in. If no valid handle exists for that number, then the routine will return the numerically smallest program handle just larger than the number passed in. The program handle found will be put in the information buffer. If no valid handle exists that is larger than `progHandle`, then `Nil` will be returned in the handle field of the information structure with `stsOK` being `from` the function.

To iterate over all program handles in the system, simply start by `OSProgramInfo` with a `progHandle` of 0. This will return the smallest program handle. On the next call, use that handle + 1, and on and on until the returned program handle 0.

**OSPowerUpTime**
Passes back the number of milliseconds since the last reset.
Returns OS_MILLISECONDS.

Function Prototype
```c
OS_MILLISECONDS EXPORTED0 OSPowerUpTime(void);
```

**ScreenOnlyStringPrint**
Prints a string onto the console.
Returns nothing.

Function Prototype
```c
void EXPORTED0 ScreenOnlyStringPrint(
    P_STRING pString // string to print
);
```
Comments
This routine will not log output through the debug log. It will only display characters on the screen.

**Debugger**
Enter the debugger.
Returns nothing.

```c
#ifdef DEBUG
    #define Debugger() OSDebugger()
#else
    #define Debugger()
#endif
```
This macro will call the symbolic debugger (DB). If the symbolic debugger is not available the low-level kernel debugger is called. In production code (i.e., compiled without /DDEBUG) this macro does nothing.

### OSDebugger

Enters the debugger, should only be called in special situations.

Returns nothing.

**Function Prototype**

```c
void EXPORTED OSDebugger(void);
```

**Comments**

Most clients should call Debugger NOT OSDebugger. OSDebugger is used in special situations were a debugger needs to be called in production code. When a call to the production version of OSDebugger is made, the debug flag /DD10000 must be set to actually enter the debugger. If the debug flag is not set the call is a NOP.

NOTE: OSDebugger should only be called in exceptional cases, such as, page fault handling.

### KeyPressed

Determines if a key is available.

Returns BOOLEAN.

**Function Prototype**

```c
BOOLEAN EXPORTED KeyPressed(
    P_U16 pCh
);  // Out: the char if true is returned
```

**Comments**

This routine is provided for support of low level code below the input system.

The high byte of the key is the scan code.

**Return Value**

- TRUE  if a key is available
- FALSE  if no key is available

**See Also**

KeyPressed, KeyIn

### KeyIn

Passes back the next key and the scan code from the keyboard.

Returns a keyboard character.

**Function Prototype**

```c
U16 EXPORTED KeyIn(void);
```

**Comments**

The KeyIn routine is provided for support of low level code below the input system.

The high byte of the key is the scan code.

**See Also**

KeyPressed, KeyIn

### OSDisplay

Changes the display to the console or the graphics screen.

Returns the old display mode.

**Function Prototype**

```c
OS_DISPLAY_MODE EXPORTED OSDisplay(
    OS_DISPLAY_MODE newDisplayMode
);  // set the display mode.
```

**Comments**

This call is only valid on single headed development systems. In all other configurations, the call does nothing.
**OSSetInterrupt**

Sets up an interrupt handler.

Returns STATUS.

**Function Prototype**

```
STATUS EXPORTED OSSetInterrupt(
    P_OS_INTERRUPT_INFO pIntInfo  // In-Out: interrupt info
);
```

**Comments**
The old interrupt info is also returned. Callable only in ring 0.

**OSTimerAsyncSema**

Reset a semaphore after time milliseconds.

Returns STATUS.

**Function Prototype**

```
STATUS EXPORTED OSTimerAsyncSema(
    OS_MILLISECONDS time,       // waiting period before sema reset
    OS_SEMA_ID sema,            // semaphore to reset
    P_OS_HANDLE pTransactionHandle  // Out: ptr to transaction handle
);
```

**Comments**
The transaction handle can be used to stop the request if desired.

**OSTimerIntervalSema**

Resets a semaphore after each time interval has elapsed.

Returns STATUS.

**Function Prototype**

```
STATUS EXPORTED OSTimerIntervalSema(
    OS_MILLISECONDS timeInterval,   // time interval in milliseconds
    OS_SEMA_ID sema,               // semaphore to reset
    P_OS_HANDLE pTransactionHandle  // Out: timer transaction handle
);
```

**Comments**
The transaction handle can be used to stop the request if desired.

**OSTimerStop**

Stops a timer request given its transaction handle.

Returns STATUS.

**Function Prototype**

```
STATUS EXPORTED OSTimerStop(
    OS_HANDLE transactionHandle    // transaction to stop
);
```

**OSTimerTransactionValid**

Checks to see if the timer transaction is valid.

Returns STATUS.

**Function Prototype**

```
STATUS EXPORTED OSTimerTransactionValid(
    OS_HANDLE transactionHandle
);
OSModuleLoad

Loads a module into the loader's database.

Returns STATUS.

Function Prototype:

```c
STATUS EXPORTED0 OSMODULELOAD(
  P_CHAR moduleName,          // Module name or dlc name
  P_CHAR pWorkingDir,         // Working dir of the app
  P_OS_PROG_HANDLE pProgHandle, // Out: Program handle
  P_CHAR pBadMod,            // Out: If error, name of module that
                           // failed, buffer must be
                           // maxModNameLength+1 long
  P_CHAR pBadReference       // Out: If error, ref name not understood
                           // buffer must be maxModNameLength+1 long
);
```

Comments:

- If a dlc file is provided, all dlls in the file will also be loaded if not loaded already.
- OSMODULELOAD will not return until instance 0 of all loaded dlls are completed. No message
  dispatching will occur during this time. If communication to the calling task is required, use
  IMModuleLoad (install.h, install.lib).

See Also:

OSProgramInstall

OSEntrypointFind

Finds an entrypoint in a loaded module either by name or by ordinal.

Returns STATUS.

Function Prototype:

```c
STATUS EXPORTED0 OSENTRYPOINTFIND(
  OS_ENTRYPOINT_TYPE entryType,       // name or ordinal
  P_STRING pName,                     // name if entryType is name
  UI64 ordinal,                      // ordinal if entryType is ordinal
  OS_PROG_HANDLE progHandle,         // Program handle
  PP_MEM ppEntrypoint                // Out: ptr to entrypoint address
);
```

See Also:

OSModuleLoad

OSProcessProgHandle

Pases back the program handle for the process.

Returns the program instance number.

Function Prototype:

```c
UI64 EXPORTED0 OSPROCESSPROGHANDLE(
  P_OS_PROG_HANDLE pProgHandle       // Out: ptr to program handle
);
```

OSEnvSearch

Searches the environment for the specified variable and returns its value.

Returns STATUS.

Function Prototype:

```c
STATUS EXPORTED0 OSENVSEARCH(
  P_STRING pVariable,               // variable name
  P_STRING outBuf,                  // Out: Output buffer for variable value
  SIZEOF bufLen                     // output buffer length
);
```
OSTaskNameSet
Sets a 4 character name for the given task.

Returns STATUS.

Function Prototype
STATUS EXPORTED0 OSTaskNameSet(
    OS_TASK_ID taskId, // task to name
    P_CHAR name      // name of task
);

OSThisApp
Passes back the application object stored with the current process.

Returns OBJECT.

Function Prototype
OBJECT EXPORTED0 OSThisApp(void);

OSTaskApp
Passes back the application object for a given process.

Returns OBJECT.

Function Prototype
OBJECT EXPORTED0 OSTaskApp(OS_TASK_ID task);

OSAppObjectPoke
Stores the application object for the current process.

Returns nothing.

Function Prototype
void EXPORTED0 OSAppObjectPoke(
    OBJECT object       // current processes application object
);

OSPowerDown
Powers down the machine.

Returns nothing.

Function Prototype
void EXPORTED0 OSPowerDown(void);

OSErrorBeep
Outputs a tone based on the type of error encountered.

Returns nothing.

Function Prototype
void EXPORTED0 OSErrorBeep(
    OS_ERROR_TYPE errorType // type of error
);

OSTone
Sends a tone for a given duration at the specified volume level.

Returns STATUS.

Function Prototype
STATUS EXPORTED0 OSTone(
    U16 frequency,   // in Hertz
    U16 duration,    // in milliseconds
    U16 volumeLevel  // 0 for off; 1 for on
);
OSThisWinDev
Passes back the windowing device for this application.
Returns OBJECT.

Function Prototype
OBJECT EXPORTED OSThisWinDev(void);

OSWinDevPoke
Stores the windowing device for the specified process.
Returns nothing.

Function Prototype
void EXPORTED OSWinDevPoke(
    OS_TASK_ID process,  // owner of application
    OBJECT winDev       // Window device object
);

OSTaskProcess
Returns the process id for the task specified.
Returns OS_TASK_ID.

Function Prototype
OS_TASK_ID EXPORTED OSTaskProcess(
    OS_TASK_ID task
);

Comments
If the task parameter is invalid, the routine will return osNullTaskId.

OSTaskInstallTerminate
Notifies tasks waiting on OSProgramInstall that the instance is finished.
Returns nothing.

Function Prototype
void EXPORTED OSTaskInstallTerminate(
    BOOLEAN wait
);

Comments
If the parameter is set the TRUE, then the caller will go into an infinite wait state in order to keep the
task and it’s allocated resources alive.

OSMemInfo
Returns information on memory usage for a specified task.
Returns STATUS.

Function Prototype
STATUS EXPORTED OSMemInfo (  
    SIZEOF memBufSize,        // size of the info buffer (in bytes)
    P_OS_MEM_INFO pMemInfo    // Out: info buffer
);

OSMemUseInfo
Returns information on memory usage for a specified task.
Returns STATUS.

Function Prototype
STATUS EXPORTED OSMemUseInfo (  
    SIZEOF memBufSize,        // size of the info buffer (in bytes)
    P_OS_MEM_USE_INFO pMemInfo // Out: info buffer
);
**OSMemAvailable**
Return amount of swappable memory available (to caution zone).
Returns STATUS.

Function Prototype
```c
STATUS EXPORTED OSMemAvailable (P_U32 pAvailable);
```

**OSSystemInfo**
Passes back information on the system configuration.
Returns STATUS.

Function Prototype
```c
STATUS EXPORTED OSSystemInfo (SIZEOF bufSize, P_OS_SYSTEM_INFO pSystemInfo);
```

**osPrintBufferRoutine**
Function variable print routine.
Returns nothing.

Function Prototype
```c
extern void FunctionPtr(osPrintBufferRoutine)(P_CHAR pStr, SIZEOF len);
```

Comments
All debug out (Debugf, DPrintf, printf, etc) flows through this function.
This file describes the heap memory management routines.

Heaps are used to allocate local and shared memory efficiently.

The functions described in this file are contained in PENPOINT.LIB.

Introduction

Heaps allocate regions of virtual memory and manage the allocation and freeing of smaller blocks within those regions.

Heaps have many different characteristics which are specified when the heap is created (see OSHeapCreate). For example, heaps can be shared (i.e. put in the shared memory space) or local.

A heap is identified by a heap handle. PenPoint pre-defines two heap handles for each process, as described below. OSHeapCreate also returns the handle of a new heap. Most heap routines take the heap handle as a parameter to identify the heap.

Pre-defined Heaps

PenPoint pre-defines two heaps for every process. These heaps can be used without calling OSHeapCreate.

osProcessHeapId is the handle for the pre-defined local heap in each process.

osProcessSharedHeapId is the handle for the shared heap. The shared heap behavior is the same as the local heap except that the shared heap resides in shared memory. Blocks allocated from the shared heap are accessible from any process.

Quick Start

Many clients call only the following functions, using one of the two pre-defined heaps.

- OSHeapBlockAlloc
- OSHeapBlockFree

Clients who need to create their own heaps also call the following functions:

- OSHeapCreate
- OSHeapDelete

Debugging Flags

Heap Manager debugging flag set is '*'. Defined flags are:

1: Validate heap before OSHeapBlockAlloc and before OSHeapBlockFree
2: Display message for each heap block allocate and free
4: Display message for each heap create and delete10: Validate heap after OSHeapBlockAlloc and after OSHeapBlockFree 20: Display messages about internal region operation (private)
1000 Display messages about the internal workings (private)
8000 Enter the debugger after printing warnings.

**Memory Overhead**

A heap consists of the memory allocated by the client plus the structures needed by the heap manager itself to maintain the heap. This section describes the overhead imposed by these structures.

A heap is constructed as a collection of REGIONS. The overhead for a region is 36 bytes. By default, regions are 16Kb long; however, a request larger than ~16K causes the creation of a special region whose size is a multiple of 4K and large enough to handle the request.

Each region have any number of allocated blocks within it. The overhead of an allocated block (beyond the size requested) is 4 bytes, plus 0-3 bytes as necessary to pad the whole block up the nearest 32-bit boundary.

```c
#ifndef OS
#define OS
ifndef OSHEAP_INCLUDED
#define OSHEAP_INCLUDED
ifndef GO_INCLUDED
#include <go.h>
#endif
ifndef OSTYPES_INCLUDED
#include <ostypes.h>
#endif
#endif

**Common #defines and typedefs**

Heap attributes for OSHeapCreate

```c
Enum16(OS_HEAP_MODE) {  
osHeapLocal = 0,        // heap is local to the owning process  
osHeapShared = flag0,  // heap is accessible by all processes  
osHeapReadWrite = 0,   // heap is writable  
osHeapReadOnly = flag1, // heap is only readable  
osHeapOptSpace = 0,    // heap is optimized for space  
osHeapOptTime = flag2, // heap is optimized for speed  
osHeapWaitForMem = 0,  // wait for memory to become available  
osHeapOutOfMemErrOK = flag3 // doesn't wait, returns out-of-memory error  
    // flags 5-10 reserved as supervisor flags  
};
```

Heap information

```c
typedef struct OS_HEAP_BLOCK_INFO {  
    SIZEOF numBlocks;  // total number of blocks  
    SIZEOF totalSize;  // total # bytes in all blocks  
    SIZEOF minSize;    // # bytes in smallest block  
    SIZEOF maxSize;    // # bytes in largest block  
} OS_HEAP_BLOCK_INFO, *P_OS_HEAP_BLOCK_INFO;

typedef struct OS_HEAP_INFO {  
    OS_HEAP_BLOCK_INFO alloc;  // info on a given heap  
    OS_HEAP_BLOCK_INFO free;   // info for allocated blocks  
    U32 numRegions;            // # regions in heap  
    U32 committedSize;        // # bytes committed  
    U32 decommittedSize;      // # bytes decommitted  
    U32 reservedSize;         // # bytes reserved  
    U32 numOwners;            // # tasks which have heap open  
    OS_HEAP_MODE heapMode;    // Mode used in heap creation  
} OS_HEAP_INFO, *P_OS_HEAP_INFO;
```

```c
#define OSTaskSharedHeapId(t) ((OS_HEAP_ID)OSTaskProcess(t))
```
Functions

OSHHeapCreate

Creates a heap.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED OSHHeapCreate(
    OS_HEAP_MODE mode,  // heap create mode
    SIZEOF size,         // initial region size
    P_OS_HEAP_ID pHeapId // Out: heap id
);
```

Comments

The size of the initial region allocated by the heap manager is a parameter to OSHHeapCreate. If the amount of memory required by the heap is more than the size of the initial region, the heap manager allocates additional regions of 16K or the last request size, whichever is larger. An initial region size of 0 will default to 16K.

Return Value

- **stsOSRequestTooBig**: The requested size is greater than maxS32.
- **stsOutOfMem**: The heap cannot be created because there is not enough memory available within the system.
- **stsBadParam**: The mode parameter specified an illegal mode.

See Also

- OSHHeapDelete

OSHHeapDelete

Deletes a heap. Frees all memory allocated by clients and by the heap manager.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED OSHHeapDelete(
    OS_HEAP_ID heapId // heap id of heap to delete
);
```

Comments

Even heap blocks that are still allocated are deleted.

If other tasks have opened the heap (using OSHHeapOpen), the heap is not actually deleted until all tasks that have opened the heap have closed it (using OSHHeapClosed). Note that this routine is similar to calling OSHHeapClose with the current task.

Return Value

- **stsOSInvalidHeapId**: The heapId was invalid or inaccessible.

See Also

- OSHHeapCreate

OSHHeapAllowError

Changes the "out of memory" behavior of heap block allocation.

Returns OS_HEAP_ID.

```c
#define OSHeapAllowError(heap) \  ((OS_HEAP_ID)((U32)(heap)|osHeapIdOutOfMemErrOKBit))
#define osHeapIdOutOfMemErrOKBit flag0
```

Comments

Normally when a heap block is requested, the heap manager returns only when the memory is available. Calling OSHHeapAllowError changes the heap so that if the system has insufficient memory the heap manager returns immediately with stsOutOfMem.
**OSHeapClear**

Clears a heap. Deletes all the allocated heap blocks but not the heap.

Returns STATUS.

```c
STATUS EXPORTED OSHeapClear(
    OS_HEAP_ID heapId,       // heap id of heap to clear
);
```

**Return Value**

stsOSHeapOpen  Heap has multiple owners and cannot be cleared.

stsOSInvalidHeapId  The heapId was invalid or inaccessible.

**See Also**

OSHeapDelete

---

**OSHeapBlockAlloc**

Allocates a block within the heap.

Returns STATUS.

```c
STATUS EXPORTED OSHeapBlockAlloc(
    OS_HEAP_ID heapId,       // heap id
    SIZEOF size,            // size of block to allocate
    P_UNKNOWN ppHeapBlock   // Out: pointer to new heap block
);
```

**Comments**

The memory for the heap block is obtained from the list of regions in the heap. If a heap allocate request is larger than the available space in the region, a new region is allocated for the request.

The newly allocated block is at least as large as the requested length. Sometimes, the heap manager allocates a block larger than the requested size. Heap blocks are always allocated on 32-bit boundaries.

Heap blocks are allocated on behalf of the creator of the heap. Even if the allocate occurs in a different task than the creator, the new memory is owned by the creator of the heap.

**WARNING.** This function expects a valid heap identifier. Using an invalid heap identifier can cause unpredictable results (including a page fault). A heapId for a heap that has been deleted is considered to be invalid.

**See Also**

OSHeapBlockFree

**Return Value**

stsOSRequestTooBig  The requested block size greater than maxS32.

stsOutOfMem  The heap cannot grow any bigger because the system is out of memory.

stsOSInvalidHeapId  The heapId given is invalid.

stsOSHeapIntegrityError  The heap has been corrupted (heap flag 1).

---

**OSHeapBlockFree**

Frees a heap block.

Returns STATUS.

```c
STATUS EXPORTED OSHeapBlockFree(
    P_UNKNOWN pHeapBlock,  // pointer to heap block
);
```

**Comments**

**WARNING.** This function expects a valid heap block. Using an invalid heap block can cause unpredictable results (including a page fault).
OSHeapBlockAlloc

stsOSInvalidHeapId
The heapId given is invalid.

stsOSHeapIntegrityError
The heap has been corrupted (heap flag 1) or heap block pointer is bad (debug only).

stsBadParam
The heap block pointer is bad (debug only).

OSHeapBlockResize
Resizes a heap block.

Returns STATUS.

Function Prototype
STATUS EXPORTED OSHeapBlockResize(
    SIZEOF newSize,   // new size to allocate
    PP_UNKNOWN ppHeapBlock   // Out: New pointer is returned here.
);

Comments
The heap block is resized to the new size. This may be slightly faster than allocating a new block and copying the original block's contents.

After the call the heap block may be identified with a new pointer value, which is returned in *ppHeapBlock.

The actual size of the new heap block may be slightly larger than the request.

WARNING. This function expects a valid heap block. Using an invalid heap block can cause unpredictable results (including a page fault).

OSHeapId
Passes back the heap id from which a heap block has been allocated.

Returns OS_HEAP_ID.

Function Prototype
OS_HEAP_ID EXPORTED OSHeapId(
    PP_UNKNOWN pHeapBlock   // pointer to a heap block
);

Comments
WARNING. This function expects a valid heap block. Using an invalid heap block can cause unpredictable results (including a page fault).

OSHeapBlockSize
Passes back the size of the heap block.

Returns STATUS.

Function Prototype
STATUS EXPORTED OSHeapBlockSize(
    PP_UNKNOWN pHeapBlock,   // pointer to the heap block
    P_SIZEOF pSize           // Out: size of the heap block
);

Comments
The size returned is the actual size of the heap block. This may be slightly larger than the requested size.

WARNING. This function expects a valid heap block. Using an invalid heap block can cause unpredictable results (including a page fault).

See Also
OSHeapBlockAlloc
OSHeapPoke
Stores 32 bits of client info in the heap header.
Returns STATUS.

Function Prototype
```c
STATUS EXPORTED OSHeapPoke(
    OS_HEAP_ID heapId,
    P_UNKNOWN info        // uninterpreted pointer stored in heap header
);
```

Comments
The client info is not interpreted by the heap manager.
There is only client info field per heap; if more than one call is made to OSHeapPoke, the most recent caller determines the value stored.

WARNING. This function expects a valid heap identifier. Using an invalid heap identifier can cause unpredictable results (including a page fault). An heapId for a heap that has been deleted is considered to be invalid.

OSHeapPeek
Passes back the client info previously set via OSHeapPoke().
Returns STATUS.

Function Prototype
```c
STATUS EXPORTED OSHeapPeek(
    OS_HEAP_ID heapId,
    PP_UNKNOWN pInfo        // Out: pointer stored by OSHeapPoke
);
```

Comments
WARNING. This function expects a valid heap identifier. Using an invalid heap identifier can cause unpredictable results (including a page fault). A heapId for a heap that has been deleted is considered to be invalid.

OSHeapInfo
Passes back information on a heap.
Returns STATUS.

Function Prototype
```c
STATUS EXPORTED OSHeapInfo(
    OS_HEAP_ID heapId,
    S_SIZEOF heapInfoSize,    // size of heap info buffer
    P_OS_HEAP_INFO pHeapInfo  // Out: heap info buffer
);
```

Return Value
- stsOSInvalidHeapId: The heapId was invalid or inaccessible.
- stsOSHeapIntegrityError: The heap has been corrupted. Under debug version additional info is printed.

OSHeapOpen
Adds the specified task as an owner of the specified heap.
Returns STATUS.

Function Prototype
```c
STATUS EXPORTED OSHeapOpen(
    OS_HEAP_ID heapId,
    OS_TASK_ID taskId        // task to add as an owner
);
```
Heaps are owned by the task that creates them. When the task is destroyed the heap is automatically destroyed. If one task wants to access another task's heap, the heap should be opened. Opening a heap is not required, but if the task owning the heap is destroyed while the second task is accessing the heap, the second task will fault.

Memory resources allocated in the heap are not actually destroyed until the last owner of the heap deletes the heap. Note that if the heap is opened multiple times by the same owner, a corresponding OSHeapClose or OSHeapDelete must occur for each before resources are deallocated.

The kernel automatically destroys heap resources when all of the owners of the heap have terminated.

The heap is automatically opened on the behalf of the creator during an OSHeapCreate.

Return Value

stsOSInvalidHeapId The heap must be a shared heap to be opened, the heapId was invalid or inaccessible.

See Also

OSHeapCreate

OSHeapClose

Remove the specified task as an owner of the specified heap.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED OSHeapClose(
    OS_HEAP_ID heapId,  // heap id
    OS_TASK_ID taskId   // task to remove as an owner
);
```

Comments

When the heap has been closed by the last owner, the heap is automatically deleted.

Return Value

stsOSInvalidHeapId The heapId was invalid or inaccessible.

See Also

OSHeapClose

OSHeapEnumerate

Enumerates all the heaps in the given process.

Returns STATUS.

Function Prototype

```c
typedef STATUS FunctionPtr(P_OS_HEAPENUMERATE) {
    OS_HEAP_ID heapId,  // next heap
    OS_HEAP_MODE heapMode,  // mode of heap
    P_UNKNOWN clientData  // client data of OSHeapEnumerate
};
```

Function Prototype

```c
STATUS EXPORTED OSHeapEnumerate(
    P_OS_HEAP_ENUMERATE pEnumProc,
    P_UNKNOWN clientData  // passed EnumProc on each call
);
```

Comments

For each heap in the current process, OSHeapEnumerate calls the supplied callback procedure. This routine is supplied with a heapId and its mode.

OSHeapEnumerate continues until it has exhausted all the heaps in the current process or the callback routine returns an error status. If the callback procedure returns an error status, processing is terminated and the error status is returned to the caller of OSHeapEnumerate.

Return Value

stsOSInvalidHeapId The heapId was invalid or inaccessible.

See Also

OSHeapWalk
**OSHeapWalk**

Traverses the given heap.

Returns STATUS.

**Arguments**

```c
typedef struct OS_HEAP_WALK_INFO {
    P_UNKNOWN pBlock;  // address of heap block
    U32 size;          // size of block
    BOOLEAN inUse;     // true if the block is allocated
    P_UNKNOWN clientData;  // set to the client data of OSHeapWalk
    // The following fields are only supported by a debugging version of
    // PenPoint's kernel. Changing their value modifies the heap block.
    BOOLEAN marked;     // true if the block was marked w/OSHeapMark
    OS_TASK_ID owner;   // last task to allocate or free this block
    P_UNKNOWN caller;   // address of the last OSHeapBlockAlloc/Free
} OS_HEAP_WALK_INFO, *P_OS_HEAP_WALK_INFO;
```

**Function Prototype**

```c
STATUS EXPORTED OSHeapWalk(
    OS_HEAP_ID heapId,      // heap to walk
    P_OS_HEAP_WALK pWalkProc, // procedure to call for each heap block
    P_UNKNOWN clientData     // passed directly to pWalkProc
);
```

**Comments**

For each allocated block in the given heap, calls the supplied callback routine, providing the address and size of the block. OSHeapWalk continues until it has exhausted all allocated blocks in the heap or the callback routine returns an error status. If the callback procedure returns an error status, processing is terminated and the error status is immediately returned to the caller of OSHeapWalk.

**Return Value**

stsOSInvalidHeapId  The heapId was invalid or inaccessible.

**See Also**

OSHeapEnumerate

---

**OSHeapMark**

Marks all the allocated blocks in given heap.

Returns STATUS.

**Function Prototype**

```c
STATUS EXPORTED OSHeapMark(
    OS_HEAP_ID heapId,      // heap to mark
)
```

**Comments**

Combining OSHeapMark with OSHeapWalk provides a simple means to track down storage leaks. For example:

```c
// Program is in a known state
OSHeapMark(myHeap);

// Lots of OSHeapBlockAlloc/Free calls
OSHeapBlockAlloc(myHeap, xx, &blk);
OSHeapBlockFree(blk);

// Program is back to the known state.
// Any unmarked heap blocks probably indicate a storage leak
OSHeapWalk(myHeap, MyHeapWalker);
```

**Return Value**

stsOSInvalidHeapId  The heapId was invalid or inaccessible.

**See Also**

OSHeapWalk
OSHeapPrint

Prints debugging info about the given heap.

Returns STATUS.

typedef enum OS_HEAP_PRINT_FLAGS {
    osHeapSuppressFree = flag0, // Don't print the free blocks
    osHeapSuppressInUse = flag1, // Don't print the allocated blocks
    osHeapSuppressMarked = flag2, // Don't print the marked blocks
    osHeapSuppressUnmarked = flag3, // Don't print the unmarked blocks
    osHeapSuppressSummary = flag4, // Don't print the heap summary
    osHeapDisplayRegions = flag5, // Print regions in heap
    osHeapPrintAll = 0, // Display summary and all blocks
    osHeapPrintSummaryOnly = osHeapSuppressFree|osHeapSuppressInUse|
    osHeapSuppressMarked|osHeapSuppressUnmarked,
    // Show blocks created since the last call to OSHeapMark
    osHeapPrintActiveBlocks = osHeapSuppressFree|osHeapSuppressMarked
} OS_HEAP_PRINT_FLAGS;

STATUS EXPORTED OSHeapPrint(OS_HEAP_ID heapId, OS_HEAP_PRINT_FLAGS suppress);

OSHeapPrint is only available in a debugging version of the PenPoint kernel. This request is not
supported in production versions of Penpoint.

OSHeapPrint assumes the heap is not corrupted; in other words, OSHeapPrint does not duplicate any
of the integrity tests done by OSHeapInfo.

stsOSInvalidHeapId  The heapId was invalid or inaccessible.

Flags for OSHeapPrint
OSPRIV.H

This include file describes the prototypes for supervisor privilege PenPoint routines. The functions described in this file are contained in PENPOINT.LIB.

```c
#ifndef OSPRIV_INCLUDED
#define OSPRIV_INCLUDED
#ifndef OS_INCLUDED
#include <os.h>
#endif
#endif
```

**Common #defines and typedefs**

The following are heap modes for supervisor level clients

```c
#define osHeapSupervisor flag5 // heap memory access is limited to supervisor
#define osHeapNoSwap flag6     // heap memory is never swapped
#define osHeapSystem flag10    // heap is owned by the system not a process
```

Special heap defines for supervisor level clients

```c
#define osGlobalHeapId ((OS_HEAP_ID)10) // predefined heap for sys clients
```

Physical address types

```c
typedef U32 OS_PHYS_ADDR;    // physical mem address
typedef OS_PHYS_ADDR * P_OS_PHYS_ADDR;
```

Program region information

```c
typedef struct OS_PROGRAM_REGION_INFO {
P_MEM base;
SIZEOF length;
} OS_PROGRAM_REGION_INFO, *P_OS_PROGRAM_REGION_INFO;
```

**Functions**

**OSIntMask**

Sets the interrupt mask for a given interrupt.

Returns STATUS.

```c
STATUS EXPORTED OSIntMask (OS_INTERRUPT_ID intNum, P_BOOLEAN pEnable);
```

**Comments**

Note: OR in the flag osIntNumIsHardwareLevel if intNum is a hardware interrupt level (vs a MIL logical device id). The flag is defined in ostypes.h.

Warning!!! Supervisor privilege only.
**OSIntEOI**

Send an EOI request to the interrupt controller device.

Returns STATUS.

**Function Prototype**

```c
STATUS EXPORTED OSIntEOI ( 
    OS_INTERRUPT_ID intNum // MIL device id or hw interrupt level 
); 
```

**Comments**

Note: OR in the flag osIntNumIsHardwareLevel if intNum is a hardware interrupt level (vs a MIL logical device id). The flag is defined in ostypes.h.

Warning!!! Supervisor privilege only.

**OSProgramRegionInfo**

Passes back region information for the debugger.

Returns STATUS.

**Function Prototype**

```c
STATUS EXPORTED OSProgramRegionInfo ( 
    OS_PROG_HANDLE progHandle, // program handle 
    P_U32 pNRegions, // Out: number of regions 
    P_OS_PROGRAM_REGION_INFO pRI // Out: region information 
); 
```

**Comments**

Warning!!! Supervisor privilege only.

**OSSysSemaRequest**

Requests access to a system semaphore.

Returns STATUS.

**Function Prototype**

```c
STATUS EXPORTED OSSysSemaRequest ( 
    OS_SEMA_ID sema // the semaphore to lock 
); 
```

**Comments**

System semaphores are regular semaphores with a little more protection. If a task owns a system semaphore, then that task cannot be terminated or suspended by another task until the system semaphore is relinquished. With this feature, tasks can be sure that any system critical data structures will be completely updated.

If the task terminates itself while it owns a system semaphore, then the next task that acquires the system semaphore will get the warning **stsOSSemaLockBroken**.

**OSSysSemaClear** should be used to relinquish the system semaphore. The function **OSSemaCreate** is used to create the system semaphore. Any semaphore can become a system semaphore simply by calling this routine. System semaphores are only used for critical section management. Do NOT use system semaphores for event handling.

Like regular semaphores, system semaphores are counting semaphores.

Warning!!! Supervisor privilege only.

**Return Value**

**stsOSSemaLockBroken**  Previous locker of semaphore died without clearing the semaphore

**See Also**

OSSemaCreate
**OSSysSemaClear**

Releases access to the system semaphore.

Returns STATUS.

*Function Prototype*

```c
STATUS EXPORTED OSSysSemaClear ( 
   OS_SEMA_ID sema  // the semaphore to unlock 
 );
```

*Comments*

System semaphores are regular semaphores with a little more protection. If a task owns a system semaphore, then that task cannot be terminated or suspended by another task until the system semaphore is relinquished. With this feature, tasks can be sure that any system critical data structures will be completely updated.

If the task terminates itself while it owns a system semaphore, then the next task that acquires the system semaphore will get the warning `stsOSSemalockBroken`.

OSSysSemaClear should be used to relinquish the system semaphore. The function OSSemaCreate is used to create the system semaphore. Any semaphore can become a system semaphore simply by calling OSSysSemaRequest/ OSSysSemaClear. System semaphores are only used for critical section management. Do NOT use system semaphores for event handling.

Like regular semaphores, system semaphores are counting semaphores.

Warning!!! Supervisor privilege only.

*See Also*

OSSysSemaRequest

---

**OSSupervisorCall**

Performs a privilege transition to supervisor privilege.

Returns U32.

```
#if defined(_WATCOMC_) && defined(_386_)
#pragma aux OSSupervisorCall parm [eax] [edx] [ecx] modify [gs];
#endif
```

*Function Prototype*

```c
U32 __far OSSupervisorCall ( 
   P_UNKNOWN pFunction, 
   P_UNKNOWN pStackParms, 
   U32 nStackParms 
 );
```

*Comments*

The function passed into the routine will be called by OSSupervisorCall in supervisor privilege. This function will check to verify that the routine passed in is actually a supervisor level routine.

OSSupervisorCall will work correctly if called in supervisor level.

---

**OSTaskAddressInfo**

Passes back task and system memory information.

Returns STATUS.

*Function Prototype*

```c
STATUS EXPORTED OSTaskAddressInfo ( 
   P_MEM pAddr, 
   OS_TASK_ID owner, 
   SIZEOF statBufSize, 
   P_OS_ADDRESS_INFO pAddrInfo 
 );
```

*Comments*

Warning!!! Supervisor privilege only.
Data structures used by OSResourcesAvailable

```c
Enum16 (OS_RESOURCE_ZONE) {
    osResourceZoneNormal,    // Normal: plenty of resource
    osResourceZoneCaution,   // Caution: resource is getting low
    osResourceZoneWarning,   // Warning: resource is low
    osResourceZoneDanger,    // Danger: resource is really low
    osResourceZoneCritical   // Critical: PenPoint will reboot
};
#define numResourceZones 5
typedef struct OS_RESOURCE_AVAILABLE {
    OS_RESOURCE_ZONE currentZone;
    U32 resourceAvailable;
    U32 zoneLimits[numResourceZones];
} OS_RESOURCE_AVAILABLE, *P_OS_RESOURCE_AVAILABLE;
```

OSResourcesAvailable

Returns info on the available resources in the system.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED OSResourcesAvailable (  
    SIZEOF bufSize, // size of the info buffer (in bytes)  
    P_OS_RESOURCES_INFO pInfo // Out: info buffer  
);
```

OSMemMapAlloc

Allocates linear memory for memory mapped hardware

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED OSMemMapAlloc (  
    U32 physAddr, // address of memory mapped area  
    U32 length,  // length of memory to allocate  
    PP_MEM ppMem // Out: return ptr to the memory  
);
```

Comments

Creates a guard page after the memory. The memory is created with the attributes: read/write data, system privilege, owned by system TId.

Note: the physical address passed in physAddr must be within the first 16MB of physical memory.

Warning!!! Supervisor privilege only.

OSMemMapFree

Frees memory which was allocated by OSMemMapAlloc

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED OSMemMapFree (  
    PP_MEM pMem // ptr to memory to free  
);
```

Comments

Warning!!! Supervisor privilege only.
**OSDMAMemAlloc**
Allocates linear memory that is DMA-able
Returns STATUS.

Function Prototype
```c
STATUS EXPORTED OSDMAMemAlloc ( 
    U32 length, 
    OS_TASK_ID owner, 
    PP_MEM ppMem  
); 
```

Comments
Creates a guard page after the memory. The memory is created with the following attributes:
- read/write access
- supervisor privilege

Not swappable (every page locked).

All pages are mapped in and are physically contiguous in memory. For machines that have DMA boundary conditions (e.g. can't cross 64k physical boundary), the memory allocated in this region is guaranteed to honor those conditions. Memory will be allocated on system page size boundaries and all allocations will be a minimum of the processor page size.

Warning!!! Supervisor privilege only.

**OSDMAMemFree**
Frees memory which was allocated by OSDMAMemAlloc
Returns STATUS.

Function Prototype
```c
STATUS EXPORTED OSDMAMemFree ( 
    P_MEM pMem, 
    OS_TASK_ID owner 
); 
```

Comments
Warning!!! Supervisor privilege only.

**OSTaskMemInfo**
Provides memory info for the system.
Returns STATUS.

Function Prototype
```c
STATUS EXPORTED OSTaskMemInfo ( 
    OS_TASK_ID taskId, 
    SIZEOF memBufSize, 
    P_OS_MEM_INFO pMemInfo 
); 
```

Comments
Warning!!! Supervisor privilege only.

**OSVirtToPhys**
Translates a virtual address into a physical address.
Returns U32.

Function Prototype
```c
U32 EXPORTED OSVirtToPhys ( 
    P_UNKNOWN pMem 
); 
```

Comments
Warning!!! Supervisor privilege only.
**OSMemLock**

Locks pages in memory.

Returns STATUS.

**Function Prototype**

```c
STATUS EXPORTED OSMemLock (  
P_MEM pMem,          // pointer to memory  
SIZEOF length      // length in bytes of memory to lock  
);
```

**Comments**

Locked pages will not be paged out of the system. If the page is paged out before this call, then the page will be brought into memory and then locked.

A counter is maintained to keep track of multiple locks on a given page.

Warning!!! Supervisor privilege only.

---

**OSMemUnlock**

Unlocks pages in memory.

Returns STATUS.

**Function Prototype**

```c
STATUS EXPORTED OSMemUnlock (  
P_MEM pMem,          // pointer to memory  
SIZEOF length      // length in bytes of memory to unlock  
);
```

**Comments**

When the page is unlocked, it may be paged out by the memory manager.

A counter is maintained to keep track of multiple locks on a given page. When the counter goes to 0 then the page will be unlocked.

Warning!!! Supervisor privilege only.
Module Description: This include file describes types for the Penpoint kernel.

```c
#ifndef OSTYPES_INCLUDED
#define OSTYPES_INCLUDED
#endif
#ifndef GO_INCLUDED
#include <go.h>
#endif
```

### Defines

- **Status values: errors**

```c
#define stsOSBadPointer MakeStatus(clsOS, 1)
#define stsOSOutOfMem stosOutOfMem
#define stsOSNoMoreOwners MakeStatus(clsOS, 4)
#define stsOSInvalidPath MakeStatus(clsOS, 5)
#define stsOSNoSemaExists MakeStatus(clsOS, 6)
#define stsOSTimeOut MakeStatus(clsOS, 7)
#define stsOSInvalidPath MakeStatus(clsOS, 8)
#define stsOSInvalidPath MakeStatus(clsOS, 9)
#define stsOSTransactionInvalid MakeStatus(clsOS, 10)
#define stsOSRequestTooBig MakeStatus(clsOS, 11)
#define stsOSInvalidHeapId MakeStatus(clsOS, 12)
#define stsOSInvalidOperationForTask MakeStatus(clsOS, 13)
#define stsOSHeapIntegrityError MakeStatus(clsOS, 14)
#define stsOSSegmentDiscarded MakeStatus(clsOS, 15)
#define stsOSFlashEraseFailure MakeStatus(clsOS, 16)
#define stsOSFlashProgramFailure MakeStatus(clsOS, 17)
#define stsOSHeapOpen MakeStatus(clsOS, 18)
#define stsOSHeapNotOpen MakeStatus(clsOS, 19)
#define stsOSInstallInternalError MakeStatus(clsOS, 20)
#define stsOSMissingEntryName MakeStatus(clsOS, 21)
#define stsOSMissingEntryOrdinal MakeStatus(clsOS, 22)
#define stsOSInitiateInternalError MakeStatus(clsOS, 23)
#define stsOSInitiateStackOverflow MakeStatus(clsOS, 24)
#define stsOSProgInstallError MakeStatus(clsOS, 25)
#define stsOSTooManySelectors MakeStatus(clsOS, 26)
#define stsOSTooManyInstances MakeStatus(clsOS, 27)
#define stsOSDependenciesExist MakeStatus(clsOS, 28)
#define stsOSTooManyRequireds MakeStatus(clsOS, 29)
#define stsOSPathTooLong MakeStatus(clsOS, 30)
#define stsOSModuleNotFound MakeStatus(clsOS, 31)
#define stsOSBadDLCFormat MakeStatus(clsOS, 32)
#define stsOSMissingDependency MakeStatus(clsOS, 33)
#define stsOSInvalidProgramHandle MakeStatus(clsOS, 34)
#define stsOSHeapOpen MakeStatus(clsOS, 35)
#define stsOSHeapNotOpen MakeStatus(clsOS, 36)
```

- **Status values: warnings**

```c
#define stsOSSemalockBroken MakeWarning(clsOS, 1)
```
# Misc defines

```c
#define osNullTaskId  ((OS_TASK_ID)NULL)
define osNullOpenSema  ((OS_SEMA_ID)NULL)
define osInvalidHandle  ((OS_HANDLE)NULL)
define osInfiniteTime  0xFFFFFFFF
define maxModNameLength  32
```

- **Well known heap ids**

```c
#define osInvalidHeapId  ((OS_HEAP_ID)0)
define osProcessHeapId  ((OS HEAP_ID)&OSProcessHeapValue)
define osProcessSharedHeapId ((OS-basket)OSThisProcess())
```

- **Filters**

```c
#define osAnyITMessage  0xFFFFFFFF
#define osStartupCommandLineFilter  flag0
define osClsmgrSend  flag0
define osClsmgrReply  flag1
define osMILFilter  flag2
define osAppSend  flag3
define osAppReply  flag4
define osTestManagerFilter  flag5
define osClsmgrPost  flag6
define osInstallWaitingFilter  flag30
define osTerminatedTaskFilter  flag31
```

**NOTE:** flag25 - flag29 reserved for users

```c
#define userDefinedFilters  (flag25 I flag26 I flag27 I flag28 I flag29)
define objSendFilter  ((OS_ITMSG_FILTER)osClsmgrSend)
define objReplyFilter  ((OS_ITMSG_FILTER)osClsmgrReply)
```

Used to treat the intNum field as a hardware interrupt level (vs a MIL logical device id) in the routines OSSetInterrupt, OSIntMask and OSIntEOI.

```c
#define osIntNumIsHardwareLevel  flag15
```

## Typedefs

```c
typedef P UNKNOWN P_MEM;  // Pointer to memory
typedef U32  OS_HANDLE;  // Handle to an object
typedef U16  OS_TASK_ID;  // Task Id
typedef U16  OS_SEMA_ID;  // Open semaphore Id
typedef P UNKNOWN OS_PROG_HANDLE;  // Loaded program handle
typedef P UNKNOWN OS_ITMSG_ID;  // message identifier
typedef U32  OS_ITMSG_FILTER;  // Inter-task msg filter
typedef U16  OS_INTERRUPT_ID;  // logical interrupt ID
typedef U32  OS_MILLISECONDS;  // number of milliseconds
typedef P_MEM* PP_MEM;  // Pointer to memory
typedef P OS HANDLE* P_OS_HANDLE;  // Pointer to memory
typedef P OS TASK_ID* P_OS_TASK_ID;  // Pointer to memory
```

```c
typedef P OS_SEMA_ID* P_OS_SEMA_ID;
typedef P OS_PROG_HANDLE* P_OS_PROG_HANDLE;
typedef P OS_ITMSG_ID* P_OS_ITMSG_ID;
typedef P OS_ITMSG_FILTER* P_OS_ITMSG_FILTER;
typedef P OS_TASK_ERROR* P_OS_TASK_ERROR;
typedef P UNKNOWN OS_HEAP_ID, * P OS_HEAP_ID;
typedef enum OS TASK MODE {
osThisTaskOnly,  // "act" on this task only
osTaskFamily,  // "act" on all tasks in the task family
osAllTasks  // "act" on all tasks in the system
} OS TASK MODE, * P OS TASK MODE;
```
typedef enum OS_PRIORITY_CLASS {
    osDefaultClass,        // use existing class
    osHighPriority,        // the class is "high priority"
    osMedHighPriority,     // the class is "med high priority"
    osMedLowPriority,      // the class is "med low priority"
    osLowPriority          // the class is "low priority"
} OS_PRIORITY_CLASS, * P_OS_PRIORITY_CLASS;

typedef struct OS_ITMSG_INFO* P_OS_ITMSG_INFO;

### Public Functions

**OSThisProcess**

Passes back the task id of this task's process.

Returns OS_TASK_ID.

Function Prototype

```c
OS_TASK_ID EXPORTED0 OSThisProcess(void);
```

Comments

Note: This function is defined here (instead of in os.h) to satisfy the definition for osProcessSharedHeapId above.
Interfaces to sorting routines.
This file contains the API definition for the quicksort sorting algorithm.

NOTE: qsort can be found in stdlib.h

```
#ifndef SORT_INCLUDED
#define SORT_INCLUDED

Version 1.0

Public Functions

quicksort
Sorts a linked list of records using the "quicksort" algorithm.
Returns pointer.

extern void ** quicksort(void **head, int (*comp)(void **, void **));

Usage:

struct record *head;
int comp (struct record *p, struct record *q);

head = quicksort (head, comp);
```

The routine "quicksort" takes an argument "head", which is a pointer to the first record of a linked list. It also takes an argument "comp", which is the name of a user-supplied routine for comparing two list records. The routine "comp" must take as its arguments a pointer to each of two list records, and must return an integer, either (-1) if the first record is "smaller than" the second, (0) if the first record is "equal to" the second, or (+1) if the first record is "larger than" the second.

After sorting, "quicksort" returns a pointer to the new first record of the linked list (i.e., the new "head" of the list).

The structure of the linked list records is as follows. The first field of each list record must be the "next" pointer. The actual data in the list records may be of variable size.

```
+--------+ +--------+ +--------+ +--------+ +--------+
| head | ----->| next | ----->| next | ----->| next | ----->| next | ----->pNull
+--------+ +--------+ +--------+ +--------+ +--------+
| data | | data | | data | | data |
| .... | | .... | | .... | | .... |
| .... | | .... | | .... | | .... |
+--------+ +--------+ +--------+ +--------+
```

The "quicksort" algorithm is fast. However, it is recursive. When there are N records in the list, the maximum recursion depth will average around (ln N) calls. Each recursion puts about 30 bytes on the stack.
This file contains the API definition for clsTimer.

Notes:
"theTimer" is a well known object that provides timer and alarm support.

clsTimer inherits from clsObject.

```c
#ifndef TIMER_INCLUDED
#define TIMER_INCLUDED

Include file dependencies for this include file

#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif
#ifndef OS_INCLUDED
#include <os.h>
#endif
#endif
```

### Class Timer Messages

#### msgTimerRegister

Registers a request for notification.

Takes P_TIMER_REGISTER_INFO, returns STATUS.

```c
#define msgTimerRegister MakeMsg(clsTimer, 1)
```

**Arguments**

```c
typedef struct TIMER_REGISTER_INFO {
  OBJECT client; // client object to notify
  OS_MILLISECONDS time; // waiting period before msg is sent
  P_UNIQUE clientData; // Uninterpreted client data
  OS_HANDLE transactionHandle; // Out: transaction handle
} TIMER_REGISTER_INFO, * P_TIMER_REGISTER_INFO;
```

**Comments**

Sent by the client to the timer object for notification after a specified time period has elapsed. At that time, msgTimerNotify will be sent (via ObjectPost) to the client. See that message for details.

When the machine is turned off, the time period will stop counting down until the machine is turned back on.

To stop the timeout message, use msgTimerStop.

The use of ObjectPost to send the msgTimerNotify message means that it will be synchronous with input events.

**Return Value**

- **stsBadObject** The client field cannot be a local object.
msgTimerRegisterAsync

Registers a request for notification.

Takes P_TIMER_REGISTER_INFO, returns STATUS.

```c
#define msgTimerRegisterAsync MakeMsg(clsTimer, 9)
```

```c
typedef struct TIMER_REGISTER_INFO {
    OBJECT client;  // client object to notify
    OS_MILLISECONDS time;  // waiting period before msg is sent
    P.Unknown clientData;  // Uninterpreted client data
    OS_HANDLE transactionHandle;  // Out: transaction handle
} TIMER_REGISTER_INFO, *P_TIMER_REGISTER_INFO;
```

Comments

Sent by the client to the timer object for notification after a specified time period has elapsed. At that time, msgTimerNotify will be sent (via ObjectPostAsync) to the client. See that message for details.

When the machine is turned off, the time period will stop counting down until the machine is turned back on.

To stop the timeout message, use msgTimerStop.

The use of ObjectPostAsync to send the msgTimerNotify message means that it will NOT be synchronous with input events.

msgTimerRegisterDirect

Registers a request for notification.

Takes P_TIMER_REGISTER_INFO, returns STATUS.

```c
#define msgTimerRegisterDirect MakeMsg(clsTimer, 12)
```

```c
typedef struct TIMER_REGISTER_INFO {
    OBJECT client;  // client object to notify
    OS_MILLISECONDS time;  // waiting period before msg is sent
    P.Unknown clientData;  // Uninterpreted client data
    OS_HANDLE transactionHandle;  // Out: transaction handle
} TIMER_REGISTER_INFO, *P_TIMER_REGISTER_INFO;
```

Comments

Sent by the client to the timer object for notification after a specified time period has elapsed. At that time, msgTimerNotify will be sent (via ObjectPostDirect) to the client. See that message for details.

When the machine is turned off, the time period will stop counting down until the machine is turned back on.

To stop the timeout message, use msgTimerStop.

The use of ObjectPostDirect to send the msgTimerNotify message means that it will NOT be synchronous with input events.

msgTimerRegisterInterval

Registers a request for interval notification.

Takes P_TIMER_INTERVAL_INFO, returns STATUS.

```c
#define msgTimerRegisterInterval MakeMsg(clsTimer, 2)
```

```c
typedef struct TIMER_INTERVAL_INFO {
    OBJECT client;  // client object to notify
    OS_MILLISECONDS interval;  // waiting interval before msg is sent
    P.Unknown clientData;  // Uninterpreted client data
    OS_HANDLE transactionHandle;  // Out: transaction handle
} TIMER_INTERVAL_INFO, *P_TIMER_INTERVAL_INFO;
```
Class Timer Messages

Sent by the client to the timer for a notification message on a specified time interval. After each time interval, `msgTimerNotify` will be posted (via ObjectPost) to the client.

When the machine is turned off, the time period will stop counting down until the machine is turned back on.

To stop the interval messages, use `msgTimerStop`.

The use of ObjectPost to send the `msgTimerNotify` message means that it will be synchronous with input events.

Return Value

- `stsBadObject` The client field cannot be a local object.

---

**msgTimerStop**

Stops a timer transaction.

Takes OS_HANDLE, returns STATUS.

```c
#define msgTimerStop MakeMsg(clsTimer, 11)
```

---

**msgTimerTransactionValid**

Determines if a timer transaction is valid.

Takes OS_HANDLE, returns STATUS.

```c
#define msgTimerTransactionValid MakeMsg(clsTimer, 10)
```

---

**msgTimerNotify**

Notifies the client that the timer request has elapsed.

Takes P_TIMER_NOTIFY, returns nothing. Category: advisory message.

```c
#define msgTimerNotify MakeMsg(clsTimer, 3)
```

---

**msgTimerAlarmRegister**

Registers a request for alarm notification.

Takes P_TIMER_ALARM_INFO, returns STATUS.

```c
#define msgTimerAlarmRegister MakeMsg(clsTimer, 5)
```

---

**typedef struct TIMER_NOTIFY {**

- P_UNKNOWN clientData; // client data returned
- OS_HANDLE transactionHandle; // transaction handle

```c
typedef struct TIMER_NOTIFY, * P_TIMER_NOTIFY;
```

---

**Comments**

Sent by the timer object to the client.

---

**typedef struct TIMER_ALARM_INFO {**

- OBJECT client; // client object to notify
- OS_DATE_TIME alarmTime; // alarm time
- P_UNKNOWN clientData; // Uninterpreted client data
- OS_HANDLE transactionHandle; // Out: transaction handle
- TIMER_ALARM_MODE alarmMode;

```c
} TIMER_ALARM_INFO, * P_TIMER_ALARM_INFO;
```
Alarms differ from timer requests in that a time and date specifies when an alarm is to occur. The timer will ObjectPost msgTimerAlarmNotify to the client when the alarm goes off. See that message for details.

Alarms will alarm within a minute of the alarm time.

When the machine is turned off, the alarm is still active. An alarm will turn the machine on.

To stop the alarm, use the message msgTimerAlarmStop.

stsBadObject The client field cannot be a local object.

**msgTimerAlarmStop**

Stops a pending alarm request.

Takes OS_HANDLE, returns STATUS.

```c
#define msgTimerAlarmStop MakeMsg(clsTimer, 6)
```

**msgTimerAlarmNotify**

Notifies the client that the alarm request has elapsed.

Takes P_ALARM_NOTIFY, returns nothing. Category: advisory message.

```c
#define msgTimerAlarmNotify MakeMsg(clsTimer, 7)
```

**Arguments**

```c
typedef struct ALARM_NOTIFY {
P_UNKNOWN clientData;  // client data returned
OS_HANDLE transactionHandle;  // transaction handle
BOOLEAN alarmCausedPoweron;  // power up occurred due to alarm
} ALARM_NOTIFY, * P_ALARM_NOTIFY;
```

**Comments**

Sent by the timer object to the client.
Part 9 / Utility Classes
This file contains the API definition for clsDVBookshelf.

clsDVBookshelf inherits from clsIconWin.

It provides a view of bookshelves on external disks.

```c
#ifndef BKSHELF_INCLUDED
#define BKSHELF_INCLUDED

#ifndef APPWIN_INCLUDED
#include <appwin.h>
#endif

#ifndef ICONWIN_INCLUDED
#include <iconwin.h>
#endif

Common #defines and typedefs

typedef struct BOOKSHELF_METRICS {
    U32 spare1; // Spare: reserved.
    U32 spare2; // Spare: reserved.
} BOOKSHELF_METRICS, *P_BOOKSHELF_METRICS;

msgNew

Creates a new bookshelf viewer.

Takes P_BOOKSHELF_NEW, returns STATUS. Category: class message.

Arguments

typedef struct BOOKSHELF_NEW_ONLY {
    BOOKSHELF_METRICS metrics; // Initial metrics setting.
    OBJECT rootDH; // Dir handle of volume for this bkshelf.
    OBJECT win; // Window for move/copy.
    U32 reserved1;
    U32 reserved2;
} BOOKSHELF_NEW_ONLY, *P_BOOKSHELF_NEW_ONLY;

#define bookshelfNewFields
\iconWinNewFields
BOOKSHELF_NEWONLY bookshelf;

typedef struct BOOKSHELF_NEW {
    bookshelfNewFields
} BOOKSHELF_NEW, *P_BOOKSHELF_NEW;

msgBookshelfGetMetrics

Gets current metrics setting.

Takes P_BOOKSHELF_METRICS, returns STATUS.

#define msgBookshelfGetMetrics
MakeMsg(clsDVBookshelf, 1)

Arguments

typedef struct BOOKSHELF_METRICS {
    U32 spare1; // Spare: reserved.
    U32 spare2; // Spare: reserved.
} BOOKSHELF_METRICS, *P_BOOKSHELF_METRICS;
msgBookshelfSetMetrics

Sets current metrics setting.
Takes P_BOOKSHELF_METRICS, returns STATUS.

#define msgBookshelfSetMetrics MakeMsg(clsDVBookshelf, 2)

typedef struct BOOKSHELF_METRICS {
    U32 spare1;  // Spare: reserved.
    U32 spare2;  // Spare: reserved.
} BOOKSHELF_METRICS, *P_BOOKSHELF_METRICS;

Miscellaneous

// "-- Empty --" label tag
#define tagBookshelfEmpty MakeTag(clsDVBookshelf, 1)
#define hlpBKBookshelfEmpty MakeTag(clsDVBookshelf, 100)
This file contains the API definition for clsBrowser.

clsBrowser inherits from clsScrollWin.

clsBrowser provides the UI for viewing and manipulating notebooks and disks.

clsBrowser provides both the Table Of Contents view of "live" data in the notebook and the Disk Viewer view of "dead" data on disk. clsBrowser functions include displaying notebook and disk items, navigating the notebook or file system hierarchy, move/copy of documents, export of notebook documents to disk, import of files from disks into the notebook, deleting notebook and disk items, and creating notebook and disk items.

clsBrowser is useful to applications that need to allow users to select sections or documents in the notebook, or items from disk.

Some messages apply only to the TOC view or to the disk view. Disk View only messages are labeled DskView only, TOC view only messages are labeled TOC only.

Many browser messages are sent to self allowing subclasses to modify browser behavior.

Move/Copy Conventions

See embedwin.h for move/copy protocol.

When the source of a move/copy, the browser responds to msgXferGetList with:

XferName can xfer the name of the selection

XferFullPathName can xfer the full path name of the selection

XferFlatLocator can xfer the flat locator of the selection

clsFileSystem can xfer as a file or directory

clsEmbeddedWin can xfer as "live" data notebook, section, or document

clsExport If source is TOC and export mode is in effect then do export instead of copy. (see export.h for details)

If the destination is the disk and the xferList contains clsExport then do export instead of move/copy.

If not an export, and the xferList contains clsEmbeddedWin then let the embedded win superclass will handle the move/copy.

If the destination is the TOC and source is not a clsEmbeddedWin then invoke the import code.

Otherwise, if the source is clsFileSystem do a file system move or copy.

 ifndef BROWSER_INCLUDED
 define BROWSER_INCLUDED
 ifndef GO_INCLUDED
 ifndef GO_INCLUDED
 include <go.h>
 endif
 endif
Common #defines and typedefs

Sort Types

Defines the order the browser will sort display items by.

```
Enum16 ( SORT_BY ) {
    browserSortByName = 1,
    browserSortBySize = 2,
    browserSortByDate = 3,
    browserSortByPage = 4,
    browserSortByType = 5
};
```

These are tags for the icons used by clsBrowser

```
#define tagBrowserSmallFileIcon MakeTag(clsBrowser,1)
#define tagBrowserBigFileIcon MakeTag(clsBrowser,2)
#define tagBrowserSmallClosedDirIcon MakeTag(clsBrowser,3)
#define tagBrowserBigClosedDirIcon MakeTag(clsBrowser,4)
#define tagBrowserSmallOpenDirIcon MakeTag(clsBrowser,5)
#define tagBrowserBigOpenDirIcon MakeTag(clsBrowser,6)
#define tagBrowserSmallClosedSectIcon MakeTag(clsBrowser,7)
#define tagBrowserBigClosedSectIcon MakeTag(clsBrowser,8)
#define tagBrowserSmallOpenSectIcon MakeTag(clsBrowser,9)
#define tagBrowserBigOpenSectIcon MakeTag(clsBrowser,10)
#define tagBrowserSmallDefaultDocIcon MakeTag(clsBrowser,11)
#define tagBrowserBigDefaultDocIcon MakeTag(clsBrowser,12)
```

These are the help ID's used for the various browser items.

```
#define hlpBrowser MakeTag(clsBrowser,170) // Generic TOC
#define hlpBrowserIcon MakeTag(clsBrowser,169) // TOC
#define hlpBrowserName MakeTag(clsBrowser,171) // TOC
#define hlpBrowserPage MakeTag(clsBrowser,172) // TOC
#define hlpBrowserType MakeTag(clsBrowser,173) // TOC
#define hlpBrowserDate MakeTag(clsBrowser,174) // TOC
#define hlpBrowserTime MakeTag(clsBrowser,175) // TOC
#define hlpBrowserSize MakeTag(clsBrowser,176) // TOC
#define hlpBrowserBookmark MakeTag(clsBrowser,177) // TOC
#define hlpBrowserColumn MakeTag(clsBrowser,178) // TOC
```

DskViewer help tags

```
#define hlpBrowserDV MakeTag(clsBrowser,180) // Generic DSKVIEW
```
Common #defines and typedefs

```c
#define hlpBrowNameDV MakeTag(clsBrowser, 181) // DSKVIEW
#define hlpBrowTypeDV MakeTag(clsBrowser, 183) // DSKVIEW
#define hlpBrowDateDV MakeTag(clsBrowser, 184) // DSKVIEW
#define hlpBrowTimeDV MakeTag(clsBrowser, 185) // DSKVIEW
#define hlpBrowSizeDV MakeTag(clsBrowser, 186) // DSKVIEW

#define tagBrowNameColumn MakeTag(clsBrowser, 191)
#define tagBrowPageColumn MakeTag(clsBrowser, 192)
#define tagBrowTypeColumn MakeTag(clsBrowser, 193)
#define tagBrowDateColumn MakeTag(clsBrowser, 194)
#define tagBrowTimeColumn MakeTag(clsBrowser, 195)
#define tagBrowSizeColumn MakeTag(clsBrowser, 196)
#define tagBrowBookmarkColumn MakeTag(clsBrowser, 197)
#define tagBrowUserColumn0 MakeTag(clsBrowser, 198)
#define tagBrowUserColumn1 MakeTag(clsBrowser, 199)
#define tagBrowUserColumn2 MakeTag(clsBrowser, 200)
#define tagBrowUserColumn3 MakeTag(clsBrowser, 201)
```

Messages

**msgNewDefaults:**

Initializes the BROWSER_NEW structure to default values.

Takes P_BROWSER_NEW, returns STATUS. Category: class message.

Comments

Zeros out pNew->browser.

**msgNew:**

Creates a new browser object.

Takes P_BROWSER_NEW, returns STATUS. Category: class message.

Arguments

```c
typedef struct BROWSER_NEW_ONLY {
    FS_LOCATOR base;   // Points to where the browser will display.
    OBJECT client;     // Note: This UID must not be an absolute path!
    U16   tocView;     // UID of client.
    U8    spare[8];   // TRUE for TOC view, FALSE for disk view.
} BROWSER_NEW_ONLY, *P_BROWSER_NEW_ONLY;
#define    browserNewFields    \  
    scrollWinNewFields         \  
    BROWSER_NEW_ONLY          \  
    browser;                  

typedef struct BROWSER_NEW {
    browserNewFields
} BROWSER_NEW, *P_BROWSER_NEW;
```

Comments

Creates a browser which will display the file system within the specified base directory. If the browser will be looking at "live" notebook sections and documents set tocView to true; If the browser will be looking at "dead" directories, files, or documents and sections on disk then set tocView to false.

**msgBrowserCreateDir**

Creates a directory at the selection.

Takes nothing, returns STATUS.

```c
#define msgBrowserCreateDir MakeMsg(clsBrowser, 1)
```

Comments

If nothing is selected, this message creates a directory at the top level of the disk. DskView message only. Usually sent from menu.
msgBrowserByName
Sorts by name order.
Takes nothing, returns STATUS.
#define msgBrowserByName MakeMsg(clsBrowser, 2)
Comments Displays all displayed items sorted by name order. Usually sent from menu.

msgBrowserByType
Sorts by type order.
Takes nothing, returns STATUS.
#define msgBrowserByType MakeMsg(clsBrowser, 40)
Comments Displays all displayed items sorted by type order. Usually sent from menu.

msgBrowserBySize
Sorts by size order.
Takes nothing, returns STATUS.
#define msgBrowserBySize MakeMsg(clsBrowser, 3)
Comments Displays all displayed items sorted by size order. Usually sent from menu.

msgBrowserByDate
Sorts by date order.
Takes nothing, returns STATUS.
#define msgBrowserByDate MakeMsg(clsBrowser, 4)
Comments Displays all displayed items sorted by date order. Usually sent from menu.

msgBrowserExpand
Expands sections or directories.
Takes nothing or P_FS_FLAT_LOCATOR, returns STATUS.
#define msgBrowserExpand MakeMsg(clsBrowser, 5)
Comments If pArgs is P_FS_FLAT_LOCATOR, expands P_FS_FLAT_LOCATOR otherwise if pArgs is pNull and the browser has the selection, the selection is expanded. Otherwise, every displayed closed selection is expanded.

msgBrowserCollapse
Collapses sections or directories.
Takes nothing or P_FS_FLAT_LOCATOR, returns STATUS.
#define msgBrowserCollapse MakeMsg(clsBrowser, 6)
Comments If pArgs is P_FS_FLAT_LOCATOR, collapses P_FS_FLAT_LOCATOR otherwise if pArgs is pNull and the browser has the selection, the selection is collapsed; otherwise, every open selection is collapsed.
**msgBrowserRefresh**

Refreshes the disk image the browser is displaying.

Takes nothing, returns STATUS.

```c
#define msgBrowserRefresh MakeMsg(clsBrowser, 15)
```

**msgBrowserDelete**

Deletes selection if `pNull` or `P_FS_FLAT_LOCATOR` otherwise.

Takes nothing or `P_FS_FLAT_LOCATOR`, returns STATUS.

```c
#define msgBrowserDelete MakeMsg(clsBrowser, 22)
```

**Comments**

Sent to self to allow subclass to override.

**msgBrowserRename**

 Renames browser items.

Takes nothing or `P_FS_FLAT_LOCATOR`, returns STATUS.

```c
#define msgBrowserRename MakeMsg(clsBrowser, 23)
```

**Comments**

Pops up rename dialog box for the selection if `pNull`; otherwise the item pointed to by `P_FS_FLAT_LOCATOR` is renamed. Sent to self to allow subclass to override.

**msgBrowserConfirmDelete**

Sets a flag whether to confirm deletions within a browser.

Takes BOOLEAN, returns STATUS.

```c
#define msgBrowserConfirmDelete MakeMsg(clsBrowser, 24)
```

**msgBrowserExport**

Puts the selection into export mode.

Takes nothing, returns STATUS.

```c
#define msgBrowserExport MakeMsg(clsBrowser, 118)
```

**Comments**

After this message is received by TOE the selected item is highlighted with the copy box. Then if notebook item is dragged to the DiskViewer, it will be exported, not copied. The export mode is cancelled when the selection is cancelled or the export completes. TOE only.

**msgBrowserByPage**

Sorts by page number.

Takes nothing, returns STATUS.

```c
#define msgBrowserByPage MakeMsg(clsBrowser, 25)
```

**Comments**

TOE only.
msgBrowserWriteState

Writes the current browser expanded/collapsed state to a file.
Takes nothing, returns STATUS.
#define msgBrowserWriteState MakeMsg(clsBrowser, 26)

Comments
This message saves the name of each expanded section or directory to a disk file. By using msgBrowserSetSaveFile clients or subclasses can set which file this information is stored in. By default the state file ends up in the OSThisApp's directory in a file named BROWSTAT.

msgBrowserReadState

Reads the browser expanded/collapsed state from a disk file.
Takes nothing, returns STATUS.
#define msgBrowserReadState MakeMsg(clsBrowser, 27)

Comments
This message restores the state of the browser view of the notebook or file system. By using msgBrowserSetSaveFile clients or subclasses can set which file this information is stored in. By default the state file ends up in the OSThisApp's dir in a file named browstate.

msgBrowserSetSaveFile

Sets the file that the browser will save open/close state to.
Takes P_FS_LOCATOR, returns STATUS.
#define msgBrowserSetSaveFile MakeMsg(clsBrowser,148)

msgBrowserGetMetrics

Gets browser metrics.
Takes P_BROWSER_METRICS, returns STATUS.
#define msgBrowserGetMetrics MakeMsg(clsBrowser, 28)

SubClass-definable Column Type

Defines attributes of the subclass definable browser columns. Subclasses can control up to browUserColumns (4) columns.

User Columns are columns of checkboxes or text, that subclasses of clsBrowser can control. The subclass can supply the header above the column and whether or not the boxes appear next to sections or documents or both.

User columns are enabled by setting pMetrics->userColumn.showUserColumn.

The browser sends msgBrowserUserColumnQueryState to subclasses to determine the initial state of the columns.

When a column is tapped, msgBrowserUserColumnChanged notifies subclasses that the checkbox has toggled.
#define browDefaultColumns 7 // Number of default columns.
#define browUserColumns 4 // Maximum number of user columns.
Display justifications

Enum16 (BROWSER_JUSTIFY) {
    browserLeftJustify = 0, // Left justification.
    browserRightJustify = 1, // Right justification.
    browserCenterJustify = 2, // Center justification.
    browserUserJustify = 3 // Miscellaneous justification.
};

User column type

Enum16 (USER_COLUMN_TYPE) {
    browserButtonType = 0, // Button user column.
    browserTextType = 1, // Text user column.
    browserUserType = 2 // User defined user column.
};

typedef struct {
    BROWSER_JUSTIFY headerJustify; // Justification of header.
    BROWSER_JUSTIFY columnJustify; // Justification of column.
    CHAR columnHeader[nameBufLength]; // Text for column.
} BROWSER_DEF_COLUMN, *P_BROWSER_DEF_COLUMN;

typedef struct {
    U16 showUserColumn : 1; // Must be set to TRUE for the
    // following fields to apply.
    U16 userColumnOnSections : 1; // Show userColumn next to sections.
    U16 userColumnOnDocs : 1; // Show userColumn next to documents.
    USER_COLUMN_TYPE userColumnType; // Type of field if user column.
    CHAR checkedChar; // Character to show when checked.
    CHAR uncheckedChar; // Character to show when unchecked.
    BROWSER_JUSTIFY headerJustify; // Justification of header.
    BROWSER_JUSTIFY columnJustify; // Justification of column.
    U8 spare[4]; // Spare: reserved.
} BROWSER_COLUMN, *P_BROWSER_COLUMN;

typedef struct BROWSER_METRICS {
    U16 showIcon : 1; // Show icons.
    U16 showType : 1; // Show type field.
    U16 showSize : 1; // Show size field.
    U16 showDate : 1; // Show date field.
    U16 showBookmark : 1; // Show bookmark field. (TOC only)
    U16 showHeader : 1; // Show column header.
    U16 computeRecursiveSize : 1; // Computes recursive size
    // for directories.
    U16 showIconButton : 1; // Show page turn buttons
    // instead of icons. (TOC only)
    SORT_BY sortBy; // Field by which to sort items.
    BROWSER_COLUMN userColumn[browUserColumns]; // Subclass-definable columns
    BROWSER_DEF_COLUMN defaultColumn[browDefaultColumns]; // Default columns
    U8 spare[40]; // Spare: reserved.
} BROWSER_METRICS, *P_BROWSER_METRICS;

msgBrowserSetMetrics

Sets browser metrics.

Takes P_BROWSER_METRICS, returns STATUS.

#define msgBrowserSetMetrics MakeMsg(clsBrowser, 29)
msgBrowserUserColumnGetState

Do nothing.

Takes P_BROWSER_USER_COLUMN, returns STATUS.

#define msgBrowserUserColumnGetState MakeMsg(clsBrowser, 62)

typedef struct {
    BOOLEAN changed; // TRUE if this column has changed.
    BOOLEAN state; // State of item checkbox.
    CHAR text[bufLength]; // Text of field for item.
    BOOLEAN shown; // TRUE if this column is shown.
    BOOLEAN active; // TRUE if this column is active
} BROWSER_COLUMN_STATE;

typedef struct {
    FS_FLAT_LOCATOR flat; // Locator of browser item.
    BROWSER_COLUMN_STATE column[browUserColumns]; // Column information.
    U8 spare[12]; // Spare: reserved.
} BROWSER_USER_COLUMN, *P_BROWSER_USER_COLUMN;

msgBrowserUserColumnSetState

Sets the user column states in the browser for columns that are marked changed.

Takes P_BROWSER_USER_COLUMN, returns STATUS.

#define msgBrowserUserColumnSetState MakeMsg(clsBrowser, 63)

typedef struct {
    FS_FLAT_LOCATOR flat; // Locator of browser item.
    BROWSER_COLUMN_STATE column[browUserColumns]; // Column information.
    U8 spare[12]; // Spare: reserved.
} BROWSER_USER_COLUMN, *P_BROWSER_USER_COLUMN;

msgBrowserUserColumnStateChanged

Notifies subclass when user checks a user column checkbox.

Takes P_BROWSER_USER_COLUMN, returns STATUS.

#define msgBrowserUserColumnStateChanged MakeMsg(clsBrowser, 68)

If the changed BOOLEAN is set, the user column state will be set. Does not generate a msgBrowserUserColumnStateChanged. The entire BROWSER_USER_COLUMN structure must be cleared before setting the fields that are changing.
 typedef struct {
   FS_FLAT_LOCATOR flat;   // Locator of browser item.
   BROWSER_COLUMN_STATE column[browUserColumns]; // Column information.
   U8 spare[12];           // Spare: reserved.
) BROWSER_USER_COLUMN, *P_BROWSER_USER_COLUMN;

The changed field is true for the column that was tapped.

**msgBrowserUserColumnQueryState**

Gets the user column state from subclass.

Takes P_BROWSER_USER_COLUMN, returns STATUS.

#define msgBrowserUserColumnQueryState MakeMsg(clsBrowser, 69)

This message is sent to self when the browser needs to know the user column states for a notebook item. The FS_FLAT_LOCATOR points to the file system item the browser needs to know the state of. The subclass should pass back the state or the text of each user column for the file system item.

**msgBrowserShowIcon**

Controls icon field display.

Takes BOOLEAN, returns STATUS.

#define msgBrowserShowIcon MakeMsg(clsBrowser, 100)

**msgBrowserShowButton**

Controls button field display.

Takes BOOLEAN, returns STATUS.

#define msgBrowserShowButton MakeMsg(clsBrowser, 99)

**msgBrowserShowSize**

Controls size field display.

Takes BOOLEAN, returns STATUS.

#define msgBrowserShowSize MakeMsg(clsBrowser, 102)

**msgBrowserShowDate**

Controls date field display.

Takes BOOLEAN, returns STATUS.

#define msgBrowserShowDate MakeMsg(clsBrowser, 103)

**msgBrowserShowType**

Controls type field display.

Takes BOOLEAN, returns STATUS.

#define msgBrowserShowType MakeMsg(clsBrowser, 33)
**msgBrowserShowBookmark**

Controls bookmark field display.

Takes BOOLEAN, returns STATUS.

```c
#define msgBrowserShowBookmark MakeMsg(clsBrowser, 104)
```

Comments

TOC only.

**msgBrowserShowHeader**

Controls column header display.

Takes BOOLEAN, returns STATUS.

```c
#define msgBrowserShowHeader MakeMsg(clsBrowser, 39)
```

**msgBrowserGoto**

Takes true to goto, false to bringto the selection.

Takes BOOLEAN, returns STATUS.

```c
#define msgBrowserGoto MakeMsg(clsBrowser, 105)
```

Comments

TOC only. Used by menu.

**msgBrowserGotoBringto**

Takes P_BROWSER_GOTO. If pFlat is pNull, applies to selection.

Takes P_BROWSER_GOTO, returns STATUS.

```c
#define msgBrowserGotoBringto MakeMsg(clsBrowser, 134)
```

Arguments

```c
typedef struct {
    BOOLEAN doGoto; // TRUE - Goto document.
    // FALSE - Bringto document.
    // (Goto if bringto is disabled.)
    FS_FLAT_LOCATOR flat; // Document to goto-bringto.
} BROWSER_GOTO, *P_BROWSER_GOTO;
```

Comments

Sent to self to allow subclass to override. TOC only.

**msgBrowserUndo**

Does nothing yet...

Takes nothing, returns STATUS.

```c
#define msgBrowserUndo MakeMsg(clsBrowser, 106)
```

**msgBrowserSetSelection**

Causes browser/TOC to select and display the given file system item.

Takes P_FS_FLAT_LOCATOR, returns STATUS.

```c
#define msgBrowserSetSelection MakeMsg(clsBrowser, 32)
```

Comments

As long as the locator points to an item within the browser's base directory subtree, the browser will open directories and scroll the display as necessary to display the selected item.
msgBrowserSetClient
Sets the target of the browser client messages.
Takes OBJECT, returns STATUS.
#define msgBrowserSetClient MakeMsg(clsBrowser, 108)

This message controls who gets the various browser client messages.

msgBrowserGetClient
Passes back the target of the browser client messages.
Takes P_OBJECT, returns STATUS.
#define msgBrowserGetClient MakeMsg(clsBrowser, 64)

msgBrowserGetBaseFlatLocator
Passes back the directory the browser is looking at.
Takes P_FS_FLAT_LOCATOR, returns STATUS.
#define msgBrowserGetBaseFlatLocator MakeMsg(clsBrowser, 65)

Passes back the root directory within which the browser is looking.

typedef struct {
    CHAR path[fsMaxPathLength];
} BROWSER_PATH, *P_BROWSER_PATH;

Also responds to msgXferGet with id XferFullPathName to get the selections path. Note: If possible use msgBrowserSelection with flat locators to avoid duplicate volume name confusion.

msgBrowserSelectionPath
Passes back the full path of the selection.
Takes P_BROWSER_PATH, returns STATUS.
#define msgBrowserSelectionPath MakeMsg(clsBrowser, 109)

Also responds to msgXferGet with id XferFullPathName to get the selections path.

msgBrowserSelection
Passes back the flat locator of the selection.
Takes P_FS_FLAT_LOCATOR, returns STATUS.
#define msgBrowserSelection MakeMsg(clsBrowser, 79)

Also responds to msgXferGet with id XferFlatLocator to get the selections path.

msgBrowserSelectionUUID
Passes back the UUID of the selection.
Takes P_UUID, returns STATUS.
#define msgBrowserSelectionUUID MakeMsg(clsBrowser, 117)
### msgBrowserSelectionDir

Passes back the flat locator of the directory the selection is in.

Takes `P_FS_FLAT_LOCATOR`, returns `STATUS`.

```c
#define msgBrowserSelectionDir MakeMsg(clsBrowser, 110)
```

### msgBrowserSelectionName

Returns the name of the selection.

Takes `P_CHAR`, returns `STATUS`.

```c
#define msgBrowserSelectionName MakeMsg(clsBrowser, 111)
```

**Comments**

Also responds to `msgXferGet` with id `XferName` to get the selection's name.

### msgBrowserSelectionOn

Notifies client when a selection is made inside the browser.

Takes nothing, returns `STATUS`.

```c
#define msgBrowserSelectionOn MakeMsg(clsBrowser, 112)
```

### msgBrowserSelectionOff

Notifies client when selection is yielded by the browser.

Takes nothing, returns `STATUS`.

```c
#define msgBrowserSelectionOff MakeMsg(clsBrowser, 113)
```

### msgBrowserBookmark

Notifies client that the bookmark specified by locator has toggled.

Takes `P_BROWSER_BOOKMARK`, returns `STATUS`.

```c
#define msgBrowserBookmark MakeMsg(clsBrowser, 107)
```

**Arguments**

```c
typedef struct {
    FS_LOCATOR loc;
} BROWSER_BOOKMARK, *P_BROWSER_BOOKMARK;
```

### msgBrowserCreateDoc

Creates a directory.

Takes `P_BROWSER_CREATE_DOC`, returns `STATUS`.

```c
#define msgBrowserCreateDoc MakeMsg(clsBrowser, 152)
```

**Arguments**

```c
typedef struct {
    CLASS docClass;
    P_CHAR pName;
    BOOLEAN atSelection;
    XY32 xy;
} BROWSER_CREATE_DOC, *P_BROWSER_CREATE_DOC;
```

**Comments**

The directory is created at the selection if there is one. If not, the directory is created at the top level shown. DiskView only.
**msgBrowserGetBrowWin**

Passes back the browser's internal display window.

Takes pObject, returns STATUS.

```c
#define msgBrowserGetBrowWin MakeMsg(clsBrowser, 149)
```

**Comments**
The browser's internal display window is the selected object for any selection based operations.

**msgBrowserGesture**

Sends to self gesture and which file it landed on.

Takes P_BROWSER_GESTURE, returns STATUS.

```c
#define msgBrowserGesture MakeMsg(clsBrowser, 59)
```

**Arguments**

```c
typedef struct {
    MESSAGE gesture; // Gesture that occurred.
    P_FS_FLAT_LOCATOR pFlat; // Item on which to apply the gesture.
    P_GWIN_GESTURE pGest; // Original gesture struct.
    TAG columnTag; // Tag of column on which to apply the gesture.
    U32 info; // Internal browser information.
    U32 spare[2]; // Spare: reserved.
} BROWSER_GESTURE, *P_BROWSER_GESTURE;
```

**Comments**
Allows subclasses to respond to gestures targeted at browser items. If the status returned by the subclass is \( \geq \) stsOK the gesture will NOT be sent to browser superclass. So subclasses should ignore this message or return stsOK to signify it has been handled.
This file contains the API definition for the ByteArray interface. The functions described in this file are contained in MISC.LIB.

A ByteArray implements a growing and shrinking array of bytes, indexed from 0 to ByteArrayLength()-1. A ByteArray grabs and releases memory as needed.

The ByteArray implementation is optimized for highly localized series of insertions and deletions.

```c
#ifndef BYTARRAY_INCLUDED
#define BYTARRAY_INCLUDED $Revision: 1.17 $
#endif

#ifdef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif

#ifdef DEBUG_INCLUDED
#include <debug.h>
#endif

#ifdef OS_HEAP_INCLUDED
#include <osheap.h>
#endif

typedef struct BYTE_ARRAY *
P_BYTE_ARRAY;

#define stsBAMaxExceeded MakeStatus(clsMisc, 255)

typedef U32 BYTE_INDEX, *
P_BYTE_INDEX;

#define SIZE_OF_BYTE_INDEX 4
#define maxBYTE_INDEX maxU32

typedef struct BYTE_ARRAY {
    BYTE_INDEX length;
    BYTE_INDEX bufferLength;
    P_U8 firstPart;
    BYTE_INDEX firstPartLength;
    P_U8 secondPart;
    UI6 mode;
} BYTE_ARRAY;

_BYTE_INDEX ByteArrayGapLength(p) 
   (p)->bufferLength - (p)->length)
```

**Types and Constants**

```c
typedef struct BYTE_ARRAY * P_BYTE_ARRAY;
#define stsBAMaxExceeded MakeStatus(clsMisc, 255)

typedef U32 BYTE_INDEX, *
P_BYTE_INDEX;
#define SIZE_OF_BYTE_INDEX 4
#define maxBYTE_INDEX maxU32
```

**Private**

```c
typedef struct BYTE_ARRAY {
    BYTE_INDEX length;
    BYTE_INDEX bufferLength;
    P_U8 firstPart;
    BYTE_INDEX firstPartLength;
    P_U8 secondPart;
    UI6 mode;
} BYTE_ARRAY;
```

**ByteArrayGapLength**

Returns the size of the byte array's gap.

Returns BYTE_INDEX.

```c
#define ByteArrayGapLength(p) 
   ((p)->bufferLength - (p)->length)
```
ByteArrayPrint
Prints the content of the byte array.
Returns void.

```
#ifdef DEBUG
void EXPORTED ByteArrayPrint (  
    P_BYTE_ARRAY p,  
    P_STRING charFmt, 
    int charWidth);  
#endif // DEBUG
```

Exported Functions and Macros

ByteArrayFindByte
Gets address of byte n from ByteArray p.
Returns P_U8.

```
#define ByteArrayFindByte(p,n) ( 
    (n) < (p)->firstPartLength 
    ? &((p)->firstPart[(n)]) 
    : &((p)->secondPart[(n)]))
```

Comments
Warning 1: n is evaluated twice, so it should not be an expression with an auto-increment or decrement!
Warning 2: to be as fast as possible, ByteArrayFindByte does no error checking!

ByteArrayFindIndex
Determines the index from address addr of byte in ByteArray p.
Returns BYTE_INDEX.

```
#define ByteArrayFindIndex(p,addr) ( 
    (addr) < &((p)->firstPart[(p)->firstPartLength]) 
    ? (BYTE_INDEX) (addr - (p)->firstPart) 
    : (BYTE_INDEX) (addr - (p)->secondPart))
```

Comments
This is the inverse of ByteArrayFindByte.
Warnings from ByteArrayFindByte apply here also.

ByteArrayGetByte
Get byte n from ByteArray p
Returns U8.

```
#define ByteArrayGetByte(p,n) ( 
    (n) < (p)->firstPartLength 
    ? (p)->firstPart[(n)] 
    : (p)->secondPart[(n)])
```

Comments
Warnings from ByteArrayFindByte apply here also.
**ByteArrayCreate**

Creates a byte array.

Returns STATUS.

STATUS EXPORTED

Function Prototype

```c
ByteArrayCreate(
    P_BYTE_ARRAY * pp,
    U16 mode,
    BYTE_INDEX length);
```

Comments

Only the osHeapLocal/osHeapShared flags of mode are meaningful. The initial length doesn't matter very much, since the byte array grows or shrinks as needed. However, if length is approximately correct, then early insertions will be quicker. If length<=0, a length of 1 is assumed.

Returns stsOK if able to create the byte array, in which case *pp will be the created byte array, otherwise *pp will be Nil(P_BYTE_ARRAY).

The mode parameter is really of type OS_HEAP_MODE.

**ByteArrayDestroy**

Destroys a byte array.

Returns void.

void EXPORTED

Function Prototype

```c
ByteArrayDestroy(
    P_BYTE_ARRAY p);
```

**ByteArrayGetMany**

Gets one or more characters from contiguous positions in the byte array.

Returns STATUS.

STATUS EXPORTED

Function Prototype

```c
ByteArrayGetMany(
    P_BYTE_ARRAY p,
    BYTE_INDEX pos,
    P_U8 buf,
    BYTE_INDEX buflen);
```

Comments

Retrieves up to buflen characters in p from positions [pos..MIN(pos+bufLen,ByteArrayLength(p))]. Client should insure that buf != Nil(P_U8). Returns count of bytes placed in buf.

**ByteArrayReplace**

Replaces zero or more characters in the byte array.

Returns STATUS.

STATUS EXPORTED

Function Prototype

```c
ByteArrayReplace(
    P_BYTE_ARRAY p,
    BYTE_INDEX pos,
    BYTE_INDEX len,
    P_U8 buf,
    BYTE_INDEX buflen);
```
Replaces len characters in p at positions \([pos..pos+len)\) by \(buf\) len characters from buf. Client should insure that \(pos+len \leq \text{ByteArrayLength}(p)\).

Returns:
- \(\text{stsOutOfMem}\) if no memory available, or
- \(\text{stsBadParam}\) if range \([pos..pos+len)\) is invalid, or
- \(\text{stsBAMaxExceeded}\) if the maximum ByteArray length is exceeded, or
- number bytes taken from buf otherwise.

**ByteArrayInsert**

Inserts \(buf\) len characters from buf into p at position pos.

Returns STATUS.

```c
#define ByteArrayInsert(p, pos, buf, bufLen) \
ByteArrayReplace((p), (pos), 0, (buf), (bufLen))
```

Comments

This routine does no error checking. Client should insure that: \(pos \leq \text{ByteArrayLength}(p)\).

See ByteArrayReplace for possible return values.

**ByteArrayDelete**

Delete n characters from p starting at pos.

Returns void.

```c
#define ByteArrayDelete(p, pos, len) \ 
(void) ByteArrayReplace((p), (pos), (len), Nil(P_U8), 0)
```

Comments

This routine does no error checking. Client should insure that: \(pos+len \leq \text{ByteArrayLength}(p)\).

**ByteArrayLength**

Returns the number of bytes currently stored in the BYTE_ARRAY.

Returns BYTE_INDEX.

```c
#define ByteArrayLength(p) ((p)->length)
```

**ByteArrayHeapMode**

Returns the heap mode the BYTE_ARRAY was created with.

Returns OS_HEAP_MODE.

```c
#define ByteArrayHeapMode(p) ((p)->mode)
```

**ByteArrayReserve**

Reserves space in byte array (without actually initializing it).

Returns STATUS.

```
STATUS EXPORTED
ByteArrayReserve( 
    P_BYTE_ARRAY p, 
    BYTE_INDEX pos, 
    BYTE_INDEX len);
```
Reserves len characters in p at position pos, but does not initialize them. (The gap is guaranteed to not break the reserved range.) Client should insure that pos <= ByteArrayLength(p).

Returns:

- stsOutOfMem if no memory available, or
- stsBadParam if pos is invalid, or
- stsBAMaxExceeded if the maximum ByteArray length is exceeded, or
- stsOK otherwise.

**ByteArrayWrite**

Writes the content of the byte array to the specified file.

Returns STATUS.

**Function Prototype**

```c
STATUS ByteArrayWrite(
    P_BYTE_ARRAY p,
    OBJECT file);
```

The file parameter must act like a FILE_HANDLE object.

**ByteArrayRead**

Reads previously saved content of a byte array from the specified file.

Returns STATUS.

**Function Prototype**

```c
STATUS ByteArrayRead(
    P_BYTE_ARRAY * pp,
    OBJECT file,
    OS_HEAP_MODE mode);
```

The file parameter must act like a FILE_HANDLE object.

**BAFileWriteString**

Debugging utility routine to write a string to a file.

Returns STATUS.

```c
#define DEBUG

BAFileWriteString(
    OBJECT file,
    P_U8 str);

#endif
```

Comments

Useful when initially writing filing code to insert helpful strings into the file and to then skip over the strings when reading the file.

This routine takes an exception if it encounters an error. Also, it will only work with a string whose length is MAX_STR_LENGTH or less.

The file parameter must act like a FILE_HANDLE object.
BAFileReadString

Debugging utility routine to read a string from a file.

Returns STATUS.

#ifdef DEBUG
STATUS EXPORTED

Function Prototype

BAFileReadString(

OBJECT  file,

P_U8    str);

#endif

Comments

Useful when initially writing filing code to skip over strings written with BAFileWriteString.

This routine takes an exception if it encounters an error. Also, it will only work with a string whose length is MAX_STR_LENGTH or less.

The file parameter must act like a FILE_HANDLE object.
This file contains the API definition for clsByteBuf.

clsByteBuf inherits from clsObject.

clsByteBuf provides a facility to store uninterpreted byte strings. Each object of clsByteBuf stores a single buffer. This class provides convenient object filing of the buffer data. Storage for each object’s buffer is allocated out of the creator’s shared process heap using OSHapBlockAlloc.

Clients who want to store null terminated strings should use clsString (see strobj.h).

ifndef BYTEBUF INCLUDED
#define BYTEBUF INCLUDED

#include <go.h>
#include <clsmgr.h>

typedef OBJECT BYTEBUF, *P_BYTEBUF;

typedef struct BYTEBUF_DATA {
    U16 buflen; // In/Out: Length (in bytes) of the pBuf buffer.
    P U8 pBuf; // In/Out: Object buffer.
} BYTEBUF_DATA, *P_BYTEBUF_DATA;

---

**Class Messages**

### msgNew

Creates a new buffer object.

Takes P_BYTEBUF_NEW, returns STATUS. Category: class message.

Arguments

```c
typedef struct BYTEBUF_NEW_ONLY {
    BOOLEAN allowObservers; // In: Send clsByteBuf observer messages
    // to the object's observers?
    BYTEBUF_DATA data; // In/Out: Buffer data.
} BYTEBUF_NEW_ONLY, *P_BYTEBUF_NEW_ONLY;
```

```c
#define byteBufNewFields
    objectNewFields
    BYTEBUF_NEW_ONLY bytebuf;

typedef struct BYTEBUF_NEW {
    byteBufNewFields
} BYTEBUF_NEW, *P_BYTEBUF_NEW;
```

Comments

This message allocates shared heap storage for the specified buffer.

`allowObservers` indicates whether the object will send the clsByteBuf observer messages (See `msgByteBufChanged`). Only clsByteBuf messages are affected by this option. Adding and removing observers is not affected by this option.

### msgNewDefaults

Initializes the BYTEBUF_NEW structure to default values.

Takes P_BYTEBUF_NEW, returns STATUS. Category: class message.
typedef struct BYTEBUF_NEW {
    byteBufNewFields
} BYTEBUF_NEW, *P_BYTEBUF_NEW;

Sets

pNew->bytebuf.allowObservers = true;
pNew->bytebuf.data.bufLen = 0;
pNew->bytebuf.data.pBuf = pNull;

allowObservers indicates whether the object will send the clsByteBuf observer messages. (See msgByteBufChanged)

### Object Messages

**msgByteBufGetBuf**

Passes back the object’s buffer.

Takes P_BYTEBUF_DATA, returns STATUS.

```c
#define msgByteBufGetBuf MakeMsg(clsByteBuf, 1)
```

**Message**

typedef struct BYTEBUF_DATA {
    U16 bufLen; // In/Out: Length (in bytes) of the pBuf buffer.
    P_U8 pBuf; // In/Out: Object buffer.
} BYTEBUF_DATA, *P_BYTEBUF_DATA;

**Comments**

The pointer passed back references the object’s global storage. Clients must not modify or free this storage.

**msgByteBufSetBuf**

Copies the specified buffer data into the object’s buffer.

Takes P_BYTEBUF_DATA, returns STATUS.

```c
#define msgByteBufSetBuf MakeMsg(clsByteBuf, 2)
```

**Message**

typedef struct BYTEBUF_DATA {
    U16 bufLen; // In/Out: Length (in bytes) of the pBuf buffer.
    P_U8 pBuf; // In/Out: Object buffer.
} BYTEBUF_DATA, *P_BYTEBUF_DATA;

**Comments**

Previously retrieved bytebuf pointers will be invalid after this operation. Clients must call msgByteBufGetBuf to retrieve a pointer to the valid object buffer.

### Observer Messages

**msgByteBufChanged**

Sent to observers when the object data changes.

Takes OBJECT, returns nothing. Category: observer notification.

```c
#define msgByteBufChanged MakeMsg(clsByteBuf, 3)
```

**Comments**

The message argument is the UID of the clsByteBuf object that changed.

This message is not sent if the creator did not specify allowObservers during msgNew.
This file contains the API definition for clsDiskViewWin. clsDiskViewWin inherits from clsCustomLayout. It is the view window for a multi-volume disk viewer.

**Overview**

The Disk Viewer also defines clsDVBrowBar, clsDVTabButton, clsDVIcon, and clsDVForward. These are internal classes which must be well-known uids, since the Disk Viewer component is shared.

The Disk Viewer component implements the heart of the Disk Manager. It is consists of two panels: an icon panel and a browser panel. Each known filesystem volume (connected and disconnected) is represented by an icon in the icon window. Each open volume is represented by a browser card in the browser panel. A browser card is a frame with a menu bar and control tab as decoration and an instance of clsBrowser in the view (see browser.h for details).

The icon panel is only as big as it needs to be to fit the known volumes. The browser panel takes up the rest of the space. The open browser cards equally divide up the browser panel.

Clients will typically put the Disk Viewer component inside of a frame. The frame must not be shrink-wrapped; the Disk Viewer must be told what size it should be.

clsDiskViewWin understands the following clsBrowser's messages:

*msgBrowserCreateDir*

The browser messages that deal with the selection are sent to the browser which has the current selection. Messages that do not deal with the selection or make sense if there is no selection are sent to all browsers in the Disk Viewer.

The Disk Viewer client is made the client of all the open browsers. The client will get all the messages that browsers send to their clients.

The Disk Viewer takes care of setting up browser state files in a directory off the current working directory. The Disk Viewer ensures that the state files for each volume is unique; it handles duplicate volume names.

The Disk Viewer understands msgSave and msgRestore. It will reopen volumes that were open when it was saved, and restore as much volume state (which directories were expanded) as possible.

```c
#ifndef DSKVIEW_INCLUDED
#define DSKVIEW_INCLUDED
#ifndef CLAYOUT_INCLUDED
#include <clayout.h>
#endif
#ifndef BROWSER_INCLUDED
#include <browser.h>
#endif
```
**Common #defines and typedefs**

Illegal volume name error.

```c
#define svcDVIllegalVolumeName MakeStatus(clsDiskViewWin, 0)
```

Directory where state files go, relative to the `WorkingDir`.

```c
#define pDVStateDir "diskViewState"
```

Trigger point for going over to 'K' size notation

```c
#define dvKSizeUnit 1024
```

**Icon Panel Style**

```c
#define dvShowIcons 0  // Show icons.
#define dvShowHelpText 1 // Show informative message about each
                      // view category.
#define dvShowClientWin 2 // Client sets contents via
                       // msgDVSetIconPanel.
```

**Icon Style**

```c
#define dvBigPictTitleUnder 0  // Big icon, title under picture.
#define dvBigPictTitleRight 1  // Big icon, title to right of picture.
#define dvSmallPictTitleUnder 2 // Small icon, title under picture.
#define dvSmallPictTitleRight 3 // Small icon, title to right of picture.
```

**Disk Viewer Style**

```c
typedef struct DV_STYLE {
    U16 displayRamVolume: 1,  // Display the RAM volume. Used for debugging.
    autoOpen : 1,             // Disk Viewer app sets this if /DB0800 is on.
    enableBookshelf : 1,      // If there is only one volume, open it.
    enableDirectoryView : 1,  // Should directory view be enabled?
    showVolumeMenu : 1,       // Should the volume menu be shown?
    showEditMenu : 1,         // Should the edit menu be shown?
    showViewMenu : 1,         // Should the view menu be shown?
    showOptionsMenu : 1,      // Should the options menu be shown?
    iconPanelStyle : 3,       // What should be shown in the icon panel?
    iconStyle : 3,            // Initial icon look, only used if
                              // iconPanelStyle == dvShowIcons.
    unused1 : 2;
    U16 spare1;
    U16 spare2;
} DV_STYLE, *P_DV_STYLE;
```

**Array Element For Volume Name Array**

```c
typedef struct NAME {
    US pName[nameBufLength];
} NAME, *P_NAME;
```

```c
typedef struct DV_NEW_ONLY {
    DV_STYLE style;
    P_STRING pBasePath;    // Path offset for each volume;
                           // pNull for no offset.
    OBJECT client;         // Client. Note: client is *not* saved at
                           // msgSave time. Client must restore with
                           // msgBrowserSetClient.
    U16 numOpenVols;       // Number of volumes to pre-open.
    P_NAME pOpenVols;      // Array of volume names.
    TAG displayType;       // Default display type for new cards.
```
CLASS browserClass; // Class of browser to mutate volume
// default browsers to. objNull says
// no mutation.

CLASS bookshelfClass; // Class of bookshelf viewer to mutate
// volume default bookshelf viewers.
// objNull says no mutation.

U8 spare[24]; // Spare: reserved.

typedef struct DV_NEW {
    diskViewWinNewFields
    customLayoutNewFields
    DV_NEWONLY diskViewWin;
} DV_NEW, *P_DV_NEW;
typedef struct DV_STYLE {
    U16 displayRamVolume: 1,    // Display the RAM volume. Used for debugging.
             autoOpen : 1,    // Disk Viewer app sets this if /DB0800 is on.
             enableBookshelf : 1,    // Should bookshelf viewing be enabled?
             enableDirectoryView : 1,    // Should the directory view be enabled?
             showVolumeMenu : 1,    // Should the volume menu be shown?
             showEditMenu : 1,    // Should the edit menu be shown?
             showViewMenu : 1,    // Should the view menu be shown?
             showOptionsMenu : 1,    // Should the options menu be shown?
             iconPanelStyle : 3,    // What should be shown in the icon panel?
             iconStyle : 3,    // Initial icon look, only used if
             unused1 : 2;
    U16 spare1;
    U16 spare2;
} DV_STYLE, *P_DV_STYLE;

msgDVSetStyle

Sets style setting.

Takes P_DV_STYLE, returns STATUS.

#define msgDVSetStyle MakeMsg(clsDiskViewWin, 2)

typedef struct DV_STYLE {
    U16 displayRamVolume: 1,    // Display the RAM volume. Used for debugging.
             autoOpen : 1,    // If there is only one volume, open it.
             enableBookshelf : 1,    // Should bookshelf viewing be enabled?
             enableDirectoryView : 1,    // Should the directory view be enabled?
             showVolumeMenu : 1,    // Should the volume menu be shown?
             showEditMenu : 1,    // Should the edit menu be shown?
             showViewMenu : 1,    // Should the view menu be shown?
             showOptionsMenu : 1,    // Should the options menu be shown?
             iconPanelStyle : 3,    // What should be shown in the icon panel?
             iconStyle : 3,    // Initial icon look, only used if
             unused1 : 2;
    U16 spare1;
    U16 spare2;
} DV_STYLE, *P_DV_STYLE;

msgDVGetBasePath

Passes back the current base path.

Takes P_STRING, returns STATUS.

#define msgDVGetBasePath MakeMsg(clsDiskViewWin, 3)

The argument must point to a string buffer that is at least fsPathBufLength in size.

msgDVGetIconPanel

Passes back the current icon panel window.

Takes P_WIN, returns STATUS.

#define msgDVGetIconPanel MakeMsg(clsDiskViewWin, 4)
msgDVSetIconPanel
Sets the icon panel window.
Takes P_WIN, returns STATUS.
#define msgDVSetIconPanel MakeMsg(clsDiskViewWin, 5)
Comments
This message is only relevant if style.iconPanelStyle is set to dvShowHelpText or dvShowClientWin.

msgDVGetOpenVols
Passes back the names of all the currently open volumes.
Takes P_DV_GET_OPEN_VOLS, returns STATUS.
#define msgDVGetOpenVols MakeMsg(clsDiskViewWin, 7)
Arguments
typedef struct DV_GET_OPEN_VOLS {
  U16    numOpenVolSi; // Number of open volumes.
  P_NAME pOpenVolsi; // Out: Array of volume names.
    // must be OSHeapBlockFreed.
  U8    spare[24];
} DV_GET_OPEN_VOLS, *P_DV_GET_OPEN_VOLSi;
Comments
This message allocates a heap block on the process local stack (pOpenVols). THE CALLER MUST FREE THIS BLOCK WHEN DONE.
If there are no open volumes then pOpenVols is set to pNull and nothing is allocated.

msgDVSetOptionVolume
Sets the current volume for our option sheet.
Takes OBJECT, returns STATUS.
#define msgDVSetOptionVolume MakeMsg(clsDiskViewWin, 8)

msgDVCardPopupChanged
Option card's quick installer popup button has changed.
Takes BOOLEAN, returns STATUS.
#define msgDVCardPopupChanged MakeMsg(clsDiskViewWin, 9)

msgDVOptionMenuNeed
Sent to the disk view client as notification that the option menu is being provided.
Takes nothing, returns STATUS.
#define msgDVOptionMenuNeed MakeMsg(clsDiskViewWin, 10)

msgDVOpenVolume
Opens the disk browser of the volume specified by the given name.
Takes P_CHAR, returns STATUS.
#define msgDVOpenVolume MakeMsg(clsDiskViewWin, 11)
msgDVCloseVolume
Closes the disk browser of the volume specified by the given name.
Takes P_CHAR, returns STATUS.
#define msgDVCloseVolume MakeMsg(clsDiskViewWin, 12)

msgDVConnectToVolume
Connects a network volume specified in pArgs.
Takes P_CONNECTIONS_MENU_ITEM, returns STATUS.
#define msgDVConnectToVolume MakeMsg(clsDiskViewWin, 13)

Menu Messages
#define msgDVOpenClose MakeMsg(clsDVForward, 1)
#define msgDVDuplicate MakeMsg(clsDVForward, 2)
#define msgDVAddQuickInstall MakeMsg(clsDVForward, 3)
#define msgDVRemoveQuickInstall MakeMsg(clsDVForward, 4)
#define msgDVEjectRemember MakeMsg(clsDVForward, 5)
#define msgDVEjectForget MakeMsg(clsDVForward, 6)
#define msgDVFormat MakeMsg(clsDVForward, 7)
#define msgDVrename MakeMsg(clsDVForward, 10)
#define msgDVviewAll MakeMsg(clsDVForward, 20)
#define msgDVviewBookshelf MakeMsg(clsDVForward, 21)
#define msgDVdisplayInstaller MakeMsg(clsDVForward, 22)
#define msgDVlayoutOptions MakeMsg(clsDVForward, 30)
#define msgDVdiskOptions MakeMsg(clsDVForward, 31)
#define msgDVoptionsIcon MakeMsg(clsDVForward, 41)
#define msgDVoptionsType MakeMsg(clsDVForward, 42)
#define msgDVoptionsDate MakeMsg(clsDVForward, 43)
#define msgDVoptionsSize MakeMsg(clsDVForward, 44)
#define msgDVoptionsDirSize MakeMsg(clsDVForward, 45)
#define msgDVoptionsVersion MakeMsg(clsDVForward, 46)
#define msgDVoptionsInstall MakeMsg(clsDVForward, 47)
#define msgDVsortByName MakeMsg(clsDVForward, 50)
#define msgDVsortByDate MakeMsg(clsDVForward, 51)
#define msgDVsortBySize MakeMsg(clsDVForward, 52)
#define msgDVsortByType MakeMsg(clsDVForward, 53)
// Note: clsDVForward messages 100 and above are used internally.

Tags
#define tagDVvolumeMenu MakeTag(clsDiskViewWin, 1)
#define tagDVeditMenu MakeTag(clsDiskViewWin, 2)
#define tagDVviewMenu MakeTag(clsDiskViewWin, 3)
#define tagDVoptionsMenu MakeTag(clsDiskViewWin, 4)
#define tagDVtabButton MakeTag(clsDiskViewWin, 7)
#define tagDVopenClose MakeTag(clsDiskViewWin, 10)
#define tagDVduplicate MakeTag(clsDiskViewWin, 11)
#define tagDVEjectRemember MakeTag(clsDiskViewWin, 12)
#define tagDVEjectForget MakeTag(clsDiskViewWin, 13)
#define tagDVrefresh MakeTag(clsDiskViewWin, 14)
#define tagDVquickInstall MakeTag(clsDiskViewWin, 15)
#define tagDVformat MakeTag(clsDiskViewWin, 16)
#define tagDVrename MakeTag(clsDiskViewWin, 17)
#define tagDVcreateDir MakeTag(clsDiskViewWin, 18)
#define tagDVviewChoice MakeTag(clsDiskViewWin, 20)
#define hlpDVMove MakeTag(clsDiskViewWin, 130)
#define hlpDVCopy MakeTag(clsDiskViewWin, 131)
#define hlpDVDelete MakeTag(clsDiskViewWin, 132)
#define hlpDVRename MakeTag(clsDiskViewWin, 133)
#define hlpDVCreateDir MakeTag(clsDiskViewWin, 134)
#define hlpDVViewAll MakeTag(clsDiskViewWin, 140)
#define hlpDVViewBookshelf MakeTag(clsDiskViewWin, 141)
#define hlpDVDisplayInstaller MakeTag(clsDiskViewWin, 142)
#define hlpDVExpand MakeTag(clsDiskViewWin, 143)
#define hlpDVCollapse MakeTag(clsDiskViewWin, 144)
#define hlpDVILayoutOptionMenu MakeTag(clsDiskViewWin, 145)
#define hlpDVIExpand MakeTag(clsDiskViewWin, 146)
#define hlpDVViewBookshelf MakeTag(clsDiskViewWin, 147)
#define hlpDVDisplayInstaller MakeTag(clsDiskViewWin, 148)
#define hlpDVExpand MakeTag(clsDiskViewWin, 149)
#define hlpDVViewBookshelf MakeTag(clsDiskViewWin, 150)
#define hlpDVDisplayInstaller MakeTag(clsDiskViewWin, 151)
#define hlpDVExpand MakeTag(clsDiskViewWin, 152)
#define hlpDVViewBookshelf MakeTag(clsDiskViewWin, 153)
#define hlpDVDisplayInstaller MakeTag(clsDiskViewWin, 154)
#define hlpDVExpand MakeTag(clsDiskViewWin, 155)
#define hlpDVViewBookshelf MakeTag(clsDiskViewWin, 156)
#define hlpDVDisplayInstaller MakeTag(clsDiskViewWin, 157)
#define hlpDVExpand MakeTag(clsDiskViewWin, 158)
#define hlpDVViewBookshelf MakeTag(clsDiskViewWin, 159)
#define hlpDVDisplayInstaller MakeTag(clsDiskViewWin, 160)
#define hlpDVExpand MakeTag(clsDiskViewWin, 161)
#define hlpDVViewBookshelf MakeTag(clsDiskViewWin, 162)
#define hlpDVDisplayInstaller MakeTag(clsDiskViewWin, 163)
#define hlpDVExpand MakeTag(clsDiskViewWin, 164)
#define hlpDVViewBookshelf MakeTag(clsDiskViewWin, 165)
#define hlpDVDisplayInstaller MakeTag(clsDiskViewWin, 166)
#define hlpDVExpand MakeTag(clsDiskViewWin, 167)
#define hlpDVViewBookshelf MakeTag(clsDiskViewWin, 168)
#define hlpDVDisplayInstaller MakeTag(clsDiskViewWin, 169)
#define hlpDVExpand MakeTag(clsDiskViewWin, 170)
#define hlpDVViewBookshelf MakeTag(clsDiskViewWin, 171)
#define hlpDVDisplayInstaller MakeTag(clsDiskViewWin, 172)
#define hlpDVExpand MakeTag(clsDiskViewWin, 173)
#define hlpDVViewBookshelf MakeTag(clsDiskViewWin, 174)
#define hlpDVDisplayInstaller MakeTag(clsDiskViewWin, 175)
#define hlpDVExpand MakeTag(clsDiskViewWin, 176)
#define hlpDVViewBookshelf MakeTag(clsDiskViewWin, 177)
#define hlpDVDisplayInstaller MakeTag(clsDiskViewWin, 178)
#define hlpDVExpand MakeTag(clsDiskViewWin, 179)
#define hlpDVViewBookshelf MakeTag(clsDiskViewWin, 180)
#define hlpDVDisplayInstaller MakeTag(clsDiskViewWin, 181)
#define hlpDVExpand MakeTag(clsDiskViewWin, 182)
#define hlpDVViewBookshelf MakeTag(clsDiskViewWin, 183)
#define hlpDVDisplayInstaller MakeTag(clsDiskViewWin, 184)
#define hlpDVExpand MakeTag(clsDiskViewWin, 185)
#define hlpDVViewBookshelf MakeTag(clsDiskViewWin, 186)
#define hlpDVDisplayInstaller MakeTag(clsDiskViewWin, 187)
#define hlpDVExpand MakeTag(clsDiskViewWin, 188)
#define hlpDVViewBookshelf MakeTag(clsDiskViewWin, 189)
#define hlpDVDisplayInstaller MakeTag(clsDiskViewWin, 190)
#define hlpDVNameColumn MakeTag(clsDiskViewWin, 191)
#define hlpDVTypeColumn MakeTag(clsDiskViewWin, 192)
#define hlpDVDateColumn MakeTag(clsDiskViewWin, 193)
#define hlpDVTimeColumn MakeTag(clsDiskViewWin, 194)
#define hlpDVSizeColumn MakeTag(clsDiskViewWin, 195)
#define hlpDVVersionColumn MakeTag(clsDiskViewWin, 196)
#define hlpDVInstallColumn MakeTag(clsDiskViewWin, 197)
This file contains the API definition for clsExport.

clsExport inherits from clsObject.

clsExport is the abstract class defining the API for exporting data to external disks.

The clsExport API provides a common mechanism for documents to translate themselves into foreign file formats and place the file on external disks.

**Overview**

The export protocol is initiated from the move/copy protocol (see embedwin.h). All moves/copies from the TOC to non-bookshelf views of the DiskViewer are implicitly exports.

More specifically, export happens after msgSelCopySelection reaches the DiskViewer, which is the destination of the copy, and the source of the copy includes clsExport as an item in the list returned by msgXferList. Anything moveable/copyable can potentially invoke export. (See xfer.h and sel.h for information on PenPoint’s move/copy protocol and selection management.)

The DiskViewer will send the source of the copy (the selection) msgExportGetFormats. The source should pass back an array of possible export formats. From the information in msgExportGetFormats clsApp generates the export dialog box. If the user selects the external export format and taps the Move/Copy button, the export class sends msgExport to the appropriate translator specified in msgExportGetFormats. If user selects the PenPoint format and taps the Move/Copy button, the move/copy is equivalent to msgAppMgrMove/msgAppMgrCopy (see appmgr.h).

If the source of the export is in the TOC, the DiskViewer activates the source document and sends it msgExportGetFormats.

**How to Be an Exporting Application**

Any application that wants to export must have its subclass of clsApp respond to msgExportGetFormats and msgExport.

```c
#ifndef EXPORT_INCLUDED
#define EXPORT_INCLUDED
#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef UID_INCLUDED
#include <uid.h>
#endif
#ifndef FS_INCLUDED
#include <fs.h>
#endif
#endif
```
Common #defines and typedefs

Status codes

```c
#define stsExportActivateSource MakeWarning(clsExport, 1)
#define stsExportFailed MakeWarning(clsExport, 2)
#define stsExportFailedUserNotified MakeWarning(clsExport, 3)
```

Messages

msgExportGetFormats

Passes back the export format array from from the source of the export.

Takes P_EXPORT_LIST, returns STATUS. Category: client responsibility.

```c
#define msgExportGetFormats MakeMsg(clsExport, 1)
```

Arguments

```c
typedef struct {
    TAG        documentType;   // Source document type.
    TAG        exportType;     // Export destination type.
    OBJECT     translator;     // Object which to send msgExport.
    CHAR       exportName[nameBufLength]; // Name of export type for
                                               // display in dialog box.
} EXPORT_FORMAT, *P_EXPORT_FORMAT;
```

Comments

The DiskViewer sends this message to the selection.
The recipient should allocate global memory to hold the EXPORT_FORMAT array which is passed back to the DiskViewer in the format field. The sender of msgExportGetFormats must free the memory.

If the source returns stsExportActivateSource, the DiskViewer will treat the source as an inactive document (This is how the TOC behaves when it is the source of export). The source will be activated using msgAppMgrActivate and the activated doc will be sent msgExportGetFormats.

msgExport

Initiates export by the translator.

Takes P_EXPORT_DOC, returns STATUS. Category: client responsibility.

```c
#define msgExport MakeMsg(clsExport, 2)
```

Arguments

```c
typedef struct {
    exportType; // Corresponds to exportType from
    source; // msgExportGetFormats EXPORT_FORMAT.
    destination; // Source document or null if
                  // source is not a document.
    path[fsPathBufLength]; // Destination file handle.
    charNAME[documentType]; // Source path.
    fileHandle; // If you don’t want to export to
                 // this file, use msgFSGetPath to
                 // retrieve the destination and
                 // destroy this file handle.
    path[fsPathBufLength]; // Source path.
    spare1; // Spare: reserved
    spare2; // Spare: reserved
} EXPORT_DOC, *P_EXPORT_DOC;
```
Comments

This message is sent to the translator specified in EXPORT_FORMAT. The translator is passed an open file handle to which the translator can write exported data or the translator can get the path of the file, destroy the file and replace it with its own file structure.

If the export fails, it is the exporter's responsibility for removing invalid and/or partial files created during the failed export. The minimum the client should do is send msgFSDelete to pArgs->destination to remove the file created for the exportation.

If the exporter wishes to put their custom dialog box to query the user for more information, the exporter should do this in response to msgExport. If the custom dialog allows the user to cancel the export operation, then the exporter should return stsExportFailedUserNotified which will cause PenPoint to suppress any error of the aborted export.

msgExportName

Passes back a possibly modified destination name from the translator.

Takes P_EXPORT_FORMAT, returns STATUS.

```
#define msgExportName

MakeMsg(clsExport, 3)
```

```
tdef struct { 
    TAG  documentType;  // Source document type. 
    TAG  exportType;    // Export destination type. 
    OBJECT translator;  // Object which to send msgExport. 
    CHAR  exportName[nameBufLength];  // Name of export type for 
    // display in dialog box. 
} EXPORT_FORMAT, *P_EXPORT_FORMAT;
```

Comments

This message is sent to the translator specified in EXPORT_FORMATS whenever the user chooses a new export type in the dialog box. When the translator receives the message, export name is set to the source document name. The translator should set export name exportName should be set to the "correct" destination file name. For instance the extension '.RTF' or '.WKS' may be appended to the name.

If the translator ignores this message the destination name will remain unchanged (so this message can safely be ignored).

Miscellaneous

Help tags

These are help tags on various pieces of the standard export dialog box.

```
#define hlpExportSheet    MakeTag(clsExport, 50)
#define hlpExportName     MakeTag(clsExport, 51)
#define hlpExportNewName  MakeTag(clsExport, 52)
#define hlpExportChoice   MakeTag(clsExport, 53)
```
This file contains the API definition for clsGestureMargin.

clsGestureMargin inherits from clsScrollWin.

clsGestureMargin adds a margin to the scroll win on the opposite side from the scroll bar. Gestures made in the margin are forwarded to the client win.

clsGestureMargin is used in PenPoint by the MiniNote application. MiniNote uses the gesture margin in lieu of a scroll win. When MiniNote is in writing mode, the margin is gray. In gesture mode, the margin is white.

Gesture mode is intended to indicate a "safe" mode in which the 11 core gestures can be used. In ink mode, some gestures do not work and may be interpreted as some other type of data (e.g. ink).

```c
#ifndef GMARGIN_INCLUDED
#define GMARGIN_INCLUDED
#endif

#include <swin.h>

#define clsGestureMargin MakeGlobalWKN(2572,1)
#define clsGestureMarginInnerWin MakeGlobalWKN(2573,1)

typedef struct GESTURE_MARGING_STYLE {
    U16 gestureMargin : 1, // gesture margin on/off
    wideGestureMargin : 1, // make the gesture margin wide
    // (not implemented)
    maskGestureMargin : 1, // mask out gestureMargin
    inkMode : 1, // margin is gray for if in ink mode
    reserved : 12;
} GESTURE_MARGING_STYLE, *P_GESTURE_MARGING_STYLE;

typedef struct {
    GESTURE_MARGING_STYLE style;
    S32 spares[4];
} GESTURE_MARGING_NEW_ONLY, *P_GESTURE_MARGING_NEW_ONLY;

#define gestureMarginNewFields \
    scrollWinNewFields \
    GESTURE_MARGING_NEW_ONLY gestureMargin;

typedef struct {
    gestureMarginNewFields \
} GESTURE_MARGING_NEW, *P_GESTURE_MARGING_NEW;
```

### Messages

**msgGestureMarginGetStyle**

Passes back the receiver's current style values.

Takes P_GESTURE_MARGING_STYLE, returns STATUS.

```c
#define msgGestureMarginGetStyle MakeMsg(clsGestureMargin, 1)
```
typedef struct GESTURE_MARGIN_STYLE {
    U16 gestureMargin : 1,  // gesture margin on/off
    wideGestureMargin : 1,  // make the gesture margin wide
    // (not implemented)
    maskGestureMargin : 1,  // mask out gestureMargin
    inkMode : 1,  // margin is gray for if in ink mode
    reserved :12;
} GESTURE_MARGIN_STYLE, *P_GESTURE_MARGIN_STYLE;

msgGestureMarginSetStyle
Sets the receiver's style values.
Takes P_GESTURE_MARGIN_STYLE, returns STATUS.
#define msgGestureMarginSetStyle MakeMsg(clsGestureMargin, 2)

typedef struct GESTURE_MARGIN_STYLE {
    U16 gestureMargin : 1,  // gesture margin on/off
    wideGestureMargin : 1,  // make the gesture margin wide
    // (not implemented)
    maskGestureMargin : 1,  // mask out gestureMargin
    inkMode : 1,  // margin is gray for if in ink mode
    reserved :12;
} GESTURE_MARGIN_STYLE, *P_GESTURE_MARGIN_STYLE;

msgGestureMarginSetInkMode
Sets margin to be either ink or gesture mode.
Takes BOOLEAN, returns STATUS.
#define msgGestureMarginSetInkMode MakeMsg(clsGestureMargin, 3)
Introduction

This package implements hash tables. Hash tables offer relatively fast key-based random access to data at the expense of some memory. The performance improvement over linear searching is substantial.

The defaults supplied by this package are probably fine for most data. However, hash table performance depends on both a good hash function and proper size parameters. If your data's keys are unevenly distributed then consider writing your own hash function. Try to get the hash table's initial size close to the number of expected entries divided by the fill percentage. You can vary the fill percentage to meet your tradeoffs between space and time.

Creating a Hash Table

To create a hash table:

- Allocate space for the hash table (either on the stack or in a heap block)
- Call HashInitDefaults()
- Optionally customize the HASH_INFO structure
- Call HashInit()

Examples

Here's some sample code based on a 32 bit key. (The package has built-in Hash and Compare functions for 32 bit keys; see section "Hash and Compare Functions.")

```c
typedef struct {
    U32 data;
    U32 key;
} YOUR_DATA, *P_YOUR_DATA;

P HASH_INFO pHashInfo;
P_YOUR_DATA pMD;
U32 key;

// Create table.
OSHeapBlockAlloc(osProcessHeapId, sizeof(*pHashInfo), &pHashInfo);
HashInitDefaults(pHashInfo);
// Optionally customize between calls to HashInitDefaults() and
// HashInit(). For instance, if you have 16 bit keys, you
// might do the following:
// pHashInfo->pHashFunction = HashFunction16;
// pHashInfo->pHashCompare = HashCompare16;
HashInit(pHashInfo, offsetof(YOUR_DATA, key));
```
// Add entry to hash table
OSHeapBlockAlloc(osProcessHeapId, SizeOf(YOUR_DATA), &pMD);
pMD->key = 25;
pMD->data = someData;
HashAddEntry(pHashInfo, pMD);

// Find entry in hash table. Returns stsNoMatch if not found.
key = 25;
HashFindData(pHashInfo, &key, &pMD);
Debugf("Data for key %d is %d", key, pMD->data);

// Delete entry in hash table without freeing client data.
// Returns stsNoMatch if not found.
key = 25;
HashDeleteEntry(pHashInfo, &key, &pMD, false);
OSHeapBlockFree(pMD);

// Delete entry in hash table and free the client data.
// Returns stsNoMatch if not found.
key = 25;
HashDeleteEntry(pHashInfo, &key, &pMD, true);

// Free hash table, and call OSHeapBlockFree() on all client data.
HashFree(pHashInfo, true);
OSHeapBlockFree(pHashInfo);

Enumerating Hash Table Entries

All of the entries in a hash table can be enumerated by examining the entries field of the HASH_INFO structure. Empty entries are null. Note that there are numEntries slots, numFilled of which are non-null.

```
P_HASH_INFO pHashInfo;
P_HASH_ENTRY pEntries;
pEntries = pHashInfo->entries;
for (i = 0; i < pHashInfo->numEntries; i++) {
    if (pEntries[i].pData) {
        // Do something with entry
    }
}
```

Hash and Compare Functions

The package includes good Hash and Compare functions for the following types of keys:

- 16 bit numbers
- 32 bit numbers
- 64 bit numbers
- null-terminated strings

Clients with other key types need to provide their own Hash and Compare functions. Sophisticated clients may want to provide their own Hash and Compare functions even if they have keys with one of the above types.
Replacement Hash and Compare functions should look like the following:

```c
typedef struct {
    U8 major;
    U16 minor;
} MY_KEY, *P_MY_KEY;

typedef struct {
    MY_KEY key;
    P_UNKNOWN pData;
} MY_DATA, *P_MY_DATA;

U32 EXPORTED
MyKeyHashFunction(
    P_HASH_KEY pKey)
{
    P_MY_KEY pMyKey = (P_MY_KEY)pKey;
    U32 hash;
    hash = pMyKey->major * 9551; // 9551 is prime
    hash += pMyKey->minor * 113; // 113 is prime
    return hash;
}

BOOLEAN EXPORTED MyKeyHashCompare(
    P_HASH_KEY pKey1,
    P_HASH_KEY pKey2)
{
    P_MY_KEY pMyKey1 = (P_MY_KEY)pKey1;
    P_MY_KEY pMyKey2 = (P_MY_KEY)pKey2;
    return (pMyKey1->major == pMyKey2->major) AND
           (pMyKey1->minor == pMyKey2->minor));
}
```

**Space / Time Tradeoff**

The following table show the space / time tradeoff for a variety of percentFull values, normalized to 80%. This table is a gross simplification. Among other things, it assumes well distributed keys.

<table>
<thead>
<tr>
<th>full percentage</th>
<th>relative speed</th>
<th>relative memory use</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2.8</td>
<td>8.0</td>
</tr>
<tr>
<td>20</td>
<td>2.7</td>
<td>4.0</td>
</tr>
<tr>
<td>30</td>
<td>2.5</td>
<td>2.7</td>
</tr>
<tr>
<td>40</td>
<td>2.3</td>
<td>2.0</td>
</tr>
<tr>
<td>50</td>
<td>2.0</td>
<td>1.6</td>
</tr>
<tr>
<td>60</td>
<td>1.7</td>
<td>1.3</td>
</tr>
<tr>
<td>70</td>
<td>1.4</td>
<td>1.2</td>
</tr>
<tr>
<td>80</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>90</td>
<td>.6</td>
<td>.9</td>
</tr>
<tr>
<td>95</td>
<td>.3</td>
<td>.8</td>
</tr>
</tbody>
</table>

```c
#ifndef HASH_INCLUDED
#define HASH_INCLUDED
#include <string.h>
#endif

#ifndef GO_INCLUDED
#define GO_INCLUDED
#include <go.h>
#endif

#ifndef OSTYPES_INCLUDED
#define OSTYPES_INCLUDED
#include <ostypes.h>
#endif

#ifndef OSHEAP_INCLUDED
#define OSHEAP_INCLUDED
#include <osheap.h>
#endif

#include <stddef.h>
```
Common #defines and typedefs

Default values

#define minHashTableInitialSize 15 // minimum initial size
#define minHashTableExpandSize 16  // minimum expand increment
#define hashTableMaxFillPct 80     // expand when the table gets this percentage full.

Key and Data Pointer Types

typedef void * P_HASH_KEY;
typedef void * P_HASH_DATA;

Type for Hash function

Function Prototype typedef U32 FunctionPtr(HASH_FUNCTION) (P_HASH_KEY pKey);

Type for Compare function. Function should return true if pKey1 and pKey2 point to keys with identical values.

Function Prototype typedef BOOLEAN FunctionPtr(HASH_COMPARE)(P_HASH_KEY pKey1, P_HASH_KEY pKey2);

A hash table entry.

typedef struct HASH_ENTRY {
P_HASH_DATA pData; // Points to user data
} HASH_ENTRY, * P_HASH_ENTRY, ** PP_HASH_ENTRY;

The hash table itself. Space for the table is allocated by the client. Space for the entries is allocated by hash table functions and is freed via a call to HashFree().

The debugging version of the hash table gathers statistics.

typedef struct HASH_INFO {
U32   numEntries;    // number of entries allocated.
        // Should be prime!
U32   numFilled;     // number of entries in use. Not too small or table will expand too often. Should be even.
U32   expandNumber;  // number of entries to expand by
U32   percentFull;   // max percentage full at expand time.
        // Performance falls off rapidly if table allowed to get much fuller
        // than 80%.
U16   keyOffset;     // offset of key in P_HASH_DATA
OS_HEAP_ID heap;     // heap to expand into
P_HASH_ENTRY entries; // points to hash table array.
        // Array can be indexed sequentially to find all the entries in the table. Empty slots are null.
HASH_FUNCTION pHashFunction;  // Hash function
HASH_COMPARE pHashCompare;   // Compare function
        // Statistics maintained for DEBUG version
U32   numProbes;      // Counts number of hash probes
U32   numProbeMisses; // Counts number of probe retries
U32   numAdds;       // Counts number of adds
U32   numDeletes;    // Counts number of deletes
} HASH_INFO, * P_HASH_INFO;
Functions

HashFindData
Given a key, passes back a P_HASH_DATA.
Returns STATUS.

Function Prototype
STATUS EXPORTED HashFindData ( 
    P_HASH_INFO    pInfo,
    P_HASH_KEY     pKey,
    P_HASH_DATA    * ppData);

Return Value
stsNoMatch  the key is not in the table. *ppData is undefined.
stsOK  the key is in the table.

See Also
HashFindTableEntry

HashFindTableEntry
Given a key, passes back a pointer to client data.
Returns STATUS.

Function Prototype
STATUS EXPORTED HashFindTableEntry ( 
    P_HASH_INFO    pInfo,
    P_HASH_KEY     pKey,
    PP_HASH_ENTRY  ppEntry);

Return Value
stsNoMatch  the key is not in the table. *ppEntry is undefined.
stsOK  the key is in the table.

See Also
HashFindData

HashAddEntry
Adds an entry to a hash table.
Returns STATUS.

Function Prototype
STATUS EXPORTED HashAddEntry ( 
    P_HASH_INFO    pInfo,
    P_HASH_DATA    pData);

Comments
The hash table expands if adding this entry causes the table to exceed the expand threshold.

Return Value
stsFailed  the key is already in the table

HashDeleteEntry
Deletes entry from hash table.
Returns STATUS.

Function Prototype
STATUS EXPORTED HashDeleteEntry ( 
    P_HASH_INFO    pInfo,
    P_HASH_KEY     pKey,
    P_HASH_DATA    * ppData,
    BOOLEAN        freeClientData);

Comments
If freeClientData is true then the client data is deallocated using ppData is undefined. Otherwise
*ppData contains the pointer to client data.
Freeing entries does not cause the table to shrink.

Return Value
stsNoMatch  the key is not in the table.
HashInitDefaults
Initializes hash table parameters.
Returns STATUS.

Function Prototype
STATUS EXPORTED HashInitDefaults(
P_HASH_INFO pInfo);

Comments
Warning: HashInitDefaults() MUST be called before HashInit. See the section "Examples."

Default values:
memset(pInfo, 0, sizeof(HASH_INFO));
pInfo->numEntries = 31;
pInfo->expandNumber = 24;
pInfo->heap = osProcessHeapId;
pInfo->pHashFunction = HashFunction32; // Default 32 bit key
pInfo->pHashCompare = HashCompare32; // Default 32 bit key
pInfo->percentFull = 80;

HashInit
Causes the hash table to allocate internal tables.
Returns STATUS.

Function Prototype
STATUS EXPORTED HashInit(
P_HASH_INFO pInfo,
U32 keyOffset); // offset of key in client data.

Comments
The client must call this function after calling HashInitDefaults() and performing any optional customization.

Example:
HashInitDefaults(pInfo);
HashInit(pInfo, offsetof(YOUR_DATA, key));

HashFree
Frees internal hash table memory. Optionally deallocates any remaining user data blocks.
Returns STATUS.

Function Prototype
STATUS EXPORTED HashFree(
P_HASH_INFO pInfo,
BOOLEAN freeAllEntries);

Comments
If freeAllEntries is true, then the hash table calls OSHeapBlockFree() on each remaining piece of client data.

If the client is going to call HashFree() with freeAllEntries false, the client must free all client data beforehand.

Note that this function does NOT free the HASH_INFO structure. If the client allocated it before calling HashInit() then the client should free the table after calling HashFree().
Built-in Hash and Compare Functions

The functions in this section are useful default hash and compare functions for common key types. The 64 bit, 32 bit, and 16 bit functions work equally well for signed or unsigned values.

64 bit keys

Function Prototype: U32 EXPORTED HashFunction64(P_HASH_KEY pKey);
BOOLEAN EXPORTED HashCompare64(P_HASH_KEY pKey1, P_HASH_KEY pKey2);

32 bit keys

Function Prototype: U32 EXPORTED HashFunction32(P_HASH_KEY pKey);
BOOLEAN EXPORTED HashCompare32(P_HASH_KEY pKey1, P_HASH_KEY pKey2);

16 bit keys

Function Prototype: U32 EXPORTED HashFunction16(P_HASH_KEY pKey);
BOOLEAN EXPORTED HashCompare16(P_HASH_KEY pKey1, P_HASH_KEY pKey2);

String keys

Function Prototype: U32 EXPORTED HashFunctionString(P_HASH_KEY pKey);
BOOLEAN EXPORTED HashCompareString(P_HASH_KEY pKey1, P_HASH_KEY pKey2);
IMPORT.H

This file contains the API definition for clsImport.

clsImport inherits from clsObject.

clsImport is the abstract class defining the API for importing foreign files from external disks into notebook documents.

Overview

The import protocol is triggered when the TOC receives msgSelMoveSelection or msgSelCopySelection the TOC, and the source of the move/copy includes clsFileSystem as an item in the list returned by msgXferList, then the TOC initiates the import protocol. (See xfer.h and sel.h for information on PenPoint's move/copy protocol and selection management.)

The import protocol sends msgImportQuery, as a class message, to each installed application class to determine the set of applications that can import the file.

Once every installed application has been queried, clsApp will put up an import dialog box. An instance of the application is created on the destination and msgImport is sent. If the import succeeds, the importer should return stsOK. If an error occurs and the user has not been notified of the failure, the importer should return stsImportFailed. If an error occurs and the user has been notified, the importer should return stsImportFailedUsNotified.

How to Be an Importing Application

Any application that wants to import must handle msgImportQuery and msgImport.

The import protocol sends msgImportQuery as a class message. (See clsmgr.h for more general information about class messages.) For your app to receive a class message you must have an entry something like this in your application class's method table:

```c
    MSG_INFO myAppMethods [] = {
    ...
    msgImportQuery, "MyAppImportQuery", objClassMessage,
    ...
    0
    },
```

The 'ImportQueryHandler' method can look at the contents or the name of the imported file to determine if that file can be imported by the app. If the app can import the file, the 'ImportQueryHandler' method sets the pArgs->canImport boolean to true (the default is false) and returns stsOK. The TOC will then add the application's name to the list of possible import destinations for the import dialog.

#ifndef IMPORTINCLUDED
#define IMPORTINCLUDED
#ifndef GOINCLUDED
#include <go.h>
#endif
#ifndef UIDINCLUDED
#include <uid.h>
#endif
#ifndef FSINCLUDED
#include <fs.h>
#endif
#endif
Common #defines and typedefs

Status codes

Importing applications should re stslImportFailedUserNotified if the importer detected an error during the importation and notified the user of the error. This allows the importer to give a more detailed error message to the user.

```c
#define stslImportFailed
#define stslImportFailedUserNotified
#define stslImportInvalidFormat
```

Messages

msgImportQuery

Queries each app class to see if it is capable of importing the file.

Takes P_IMPORT_QUERY, returns STATUS. Category: client responsibility.

```c
#define msgImportQuery
typedef struct {
  FILE_HANDLE file; // Open file handle to imported file.
  TAG fileType; // File type if it exists.
  CHAR fileName[nameBuf Length]; // Source file name.
  BOOLEAN canImport; // Out: TRUE if app can import the file.
  U16 suitabilityRating; // Out: 0 - lowest
                        // 50 - average (default)
                        // 100 - highest
  U8 spare[64]; // Spare: reserved.
} IMPORT_QUERY, *P_IMPORT_QUERY;
```

Arguments

This message is sent by the browser to each application class. The application should pass back `pArgs->canImport` set to true if it can import the file. `pArgs->suitabilityRating` is the relative rating of how suitable the application is to importing the file. This rating determines the ordering within the list of applications in the import dialog box displayed by PenPoint.

msgImport

Initiates the import.

Takes P_IMPORT_DOC, returns STATUS. Category: client responsibility.

```c
#define msgImport
typedef struct {
  FILE_HANDLE file; // Open file handle to file.
  TAG fileType; // File type if exists.
  U8 fileName[nameBufLength]; // Source file name.
  U32 sequence; // Sequence number for dest.
  DIRHANDLE destHandle; // Dir handle to dest section.
} IMPORT_DOC, *P_IMPORT_DOC;
```

Arguments

This message is sent by clsApp to a newly created instance of the destination application. The application should import the data from the file and return stsOK. If this message returns an error status the newly created app instance will be deleted.
Help tags

These are help tags on various pieces of the standard export dialog box.

#define hlpImportSheet        MakeTag(clsImport, 50)
#define hlpImportName         MakeTag(clsImport, 51)
#define hlpImportNewName      MakeTag(clsImport, 52)
#define hlpImportChoice       MakeTag(clsImport, 53)
This file contains the API definition for clsList.

clsList inherits from clsObject.

Lists are a simple ordered collections of items.

```c
#ifndef LIST_INCLUDED
#define LIST_INCLUDED
#endif

#ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif

Common #defines and typedefs

typedef OBJECT LIST, *P_LIST;
typedef P_UNKNOWN LIST_ITEM, *P_LIST_ITEM;

LIST_ENTRY is used in many messages. In general, the fields are treated as follows:

- position. An item’s location. Locations are zero-based. The first item is 0 and the last item is number of items - 1. When used as an In parameter, position specifies the position of the item to operate on. For adding operations, maxU16 means beyond the last item. For other operations, maxU16 means the last item in the list. Values beyond the size of the list but less than maxU16 are not recommended. When used as an Out parameter, position contains the actual position of the item. maxU16 is never passed back even if passed in.
- item. When used as an In parameter, item identifies the item to operate on. If the same item added to the list more than once, then all operations work only the first appearance of the item. When used as an Out parameter, item contains the item operated on.

typedef struct LIST_ENTRY {
    U16 position;
    LIST_ITEM item;
} LIST_ENTRY, *P_LIST_ENTRY;

typedef struct LIST_NOTIFY {
    MESSAGE msg;
    P_ARGS pArgs;
    SIZEOF lenSend;
} LIST_NOTIFY, *P_LIST_NOTIFY;

Status Codes

#define stsListFull
#define stsListEmpty

MakeStatus(clsList, 1)
MakeStatus(clsList, 2)
**Messages Defined by Other Classes**

**msgNew**

Creates a new empty list.

Takes `P_LIST_NEW`, returns `STATUS`. Category: class message.

**Arguments**

typedef struct LIST_STYLE {
U16 reserved:16;
} LIST_STYLE, *P_LIST_STYLE;

List filing behavior.

typedef enum LIST_FILE_MODE {
listFileItemsAsData,
listFileItemsAsObjects,
listDoNotFileItems
} LIST_FILE_MODE, *P_LIST_FILE_MODE;

typedef struct LIST_NEW_ONLY {
LIST_STYLE style;
LIST_FILE_MODE fileMode;
U32 reserved[4]; // Reserved
} LIST_NEW_ONLY, *P_LIST_NEW_ONLY;

#define listNewFields
objectNewFields \ LIST_NEW_ONLY list;

typedef struct LIST_NEW {
listNewFields
} LIST_NEW, *P_LIST_NEW;

**Comments**

If the heap specified in `pArgs->object.heap` is null, the process heap is used.

**msgNewDefaults**

Initializes the LIST_NEW structure to default values.

Takes `P_LIST_NEW`, returns `STATUS`. Category: class message.

**Message**

typedef struct LIST_NEW {
listNewFields
} LIST_NEW, *P_LIST_NEW;

**Comments**

Zeroes out `pNew->list` and sets:

`pArgs->list.fileMode = listFileItemsAsObjects`

**msgSave**

Defined in clsmgr.h

Takes `P_OBJ_SAVE`, returns `STATUS`.

**Comments**

In response to this message, the list saves itself. Then, based on the list's `fileMode`, it may save the item information. See the commentary with the type `LIST_FILE_MODE` for more information.
msgRestore
Defined in clsmgr.h
Takes P_OBJ_RESTORE, returns STATUS.

Comments
In response to this message, the list restores itself. Then, based on the list's fileMode, it may restore
the items information. See the commentary with the type LIST_FILE_MODE for more information.

List Manipulation Messages

msgListFree
Frees a list according to mode.
Takes P_LIST_FREE, returns STATUS.

#define msgListFree
MakeMsg(clsList, 1)

Arguments
typedef enum LIST_FREE_MODE {
    listFreeItemsAsData, // Ignore the item's value. Simply destroy
    listFreeItemsAsObjects, // Treat items as objects. Send each item
        // msgDestroy Nil(key) before destroying
    listDoNotFreeItems, // Obsolete. Do not use.
} LIST_FREE_MODE, *P_LIST_FREE_MODE;
typedef struct LIST_FREE {
    OBJ_KEY key; // Key for freeing the list object.
    LIST_FREE_MODE mode; // Key for freeing the list object.
} LIST_FREE, *P_LIST_FREE;

Comments
In response to this message, the list destroys itself AND all of its items.
Use msgDestroy to destroy the list without affecting the list's items. For both messages, observers are
sent msgListNotifyEmpty.

msgListAddItem
Adds an item to the end of a list.
Takes LIST_ITEM, returns STATUS.

#define msgListAddItem
MakeMsg(clsList, 2)

Comments
Observers are sent msgListNotifyAddition.

msgListAddItemAt
Adds an item to a list at pArgs->position.
Takes P_LIST_ENTRY, returns STATUS.

#define msgListAddItemAt
MakeMsg(clsList, 10)

Message Arguments
typedef struct LIST_ENTRY {  
    U16 position;
    LIST_ITEM item;
} LIST_ENTRY, *P_LIST_ENTRY;

Comments
If the list is empty, pArgs->position is treated as if it were 0. If pArgs->position is maxU16, the item is
inserted at the end of the list.
If necessary, list items move to make room for the new item.

Observers are sent msgListNotifyAddition.

Return Value

\textbf{stsOK} \quad \text{item added. pArgs->position contains the actual position of the new item.}

\textbf{msgListRemoveItem}

The list searches for pArgs in the list and removes the item if found.

Takes LIST_ITEM, returns STATUS.

\texttt{#define msgListRemoveItem} \quad \text{MakeMsg(clsList, 11)}

Comments

If the argument is in the list more than once, only the first instance of it is removed.

Observers are sent msgListNotifyDeletion.

Return Value

\textbf{stsListEmpty} \quad \text{the list was empty}

\textbf{stsNoMatch} \quad \text{item was not found}

\textbf{msgListRemoveItemAt}

Removes the item in the list at pArgs->position.

Takes P_LIST_ENTRY, returns STATUS.

\texttt{#define msgListRemoveItemAt} \quad \text{MakeMsg(clsList, 3)}

Message Arguments

typedef struct LIST_ENTRY {
  U16 \quad \text{position;}
  LIST_ITEM \quad \text{item;}
} LIST_ENTRY, *P_LIST_ENTRY;

Comments

Observers are sent msgListNotifyDeletion.

Return Value

\textbf{stsListEmpty} \quad \text{the list was empty}

\textbf{stsOK} \quad \text{item removed. pArgs->position contains the position of the removed item.}

\textbf{msgListReplaceItem}

Replaces the item in the list at pArgs->position.

Takes P_LIST_ENTRY, returns STATUS.

\texttt{#define msgListReplaceItem} \quad \text{MakeMsg(clsList, 4)}

Message Arguments

typedef struct LIST_ENTRY {
  U16 \quad \text{position;}
  LIST_ITEM \quad \text{item;}
} LIST_ENTRY, *P_LIST_ENTRY;

Comments

If pArgs->position is \texttt{maxU16}, the last item in the list is replaced.

Observers are sent msgListNotifyReplacement.

Return Value

\textbf{stsListEmpty} \quad \text{the list was empty}

\textbf{stsOK} \quad \text{item was replaced. pArgs->item contains the old item and pArgs->position contains its old position.}
msgListGetItem
Gets the item in the list at pArgs->position.
Takes P_LIST_ENTRY, returns STATUS.
#define msgListGetItem MakeMsg(clsList, 5)

typedef struct LIST_ENTRY {
  U16 position;
  LIST_ITEM item;
} LIST_ENTRY, *P_LIST_ENTRY;

Message
Arguments
Comments
Return Value
If pArgs->position is maxU16, the last item in the list is returned.
stsListEmpty the list was empty.
stsOK item found. pArgs->position contains the position of the item.

msgListFindItem
Searches for pArgs->item in the list.
Takes P_LIST_ENTRY, returns STATUS.
#define msgListFindItem MakeMsg(clsList, 6)

typedef struct LIST_ENTRY {
  U16 position;
  LIST_ITEM item;
} LIST_ENTRY, *P_LIST_ENTRY;

Message
Arguments
Return Value
stsNoMatch item was not found.
stsOK item was found. pArgs->position contains the position of the item.

msgListNumItems
Passes back the number of items in a list.
Takes P_U16, returns STATUS.
#define msgListNumItems MakeMsg(clsList, 7)

msgListRemoveItems
Removes all of the items in a list.
Takes no arguments, returns STATUS.
#define msgListRemoveItems MakeMsg(clsList, 8)

Comments
The list's items are not affected in any way.
Observers are sent msgListNotifyEmpty.

msgListEnumItems
Enumerates the items in a list.
Takes P_LIST_ENUM, returns STATUS.
#define msgListEnumItems MakeMsg(clsList, 9)

typedef struct LIST_ENUM {
  U16 max;
  U16 count;
  P_LIST_ITEM pItems;
  P_UNKNOWN pNext;
} LIST_ENUM, *P_LIST_ENUM;
This copies successive items from the list into an array. There are two approaches a client can use:

1. Let the list do all the work in one call. The list allocates an array of items which is passed back in pArgs->pItems. You must free this array when you are done with a call to OSHeapBlockFree. LIST_ENUM Should be filled in as follows:
   - **max**: On input, should be 0. On output, will be the number of items in the allocated block.
   - **count**: On input, should be **maxU16**. On output will be the same as max.
   - **pItems**: On input, should be null. On output, will be the pointer to the allocated block.
   - **pNext**: On input, should be null.

2. Go through the items, a chunk at a time. Repeatedly call msgListEnumItems with the same LIST_ENUM structure and processes successive groups of items. The call that returns **stsEndOfData** indicates that the enumeration is finished (there are no more items to process). LIST_ENUM is used as follows:
   - **max**: On input and output, the number of items your block can hold.
   - **count**: On input, the same as max. On output, will be the number of items returned in block. (This will be less than max the last time through.)
   - **pItems**: On input, a pointer to a block that can hold at least max items.
   - **pNext**: On input for first call, should be null. Do not modify thereafter.

**msgListGetHeap**

Passes back the heap used by the list.

Takes P_OS_HEAP_ID, returns STATUS.

```c
#define msgListGetHeap MakeMsg(clsList, 12)
```

**Forwarding Messages**

clsList responds to these messages by sending the specified message to each item in the list in turn. clsList ignores the values returned by sending this message and always returns **stsOK**.

**msgListCall**

Sends a message to each object in the list using ObjectCall.

Takes P_LIST_NOTIFY, returns STATUS.

```c
#define msgListCall MakeMsg(clsList, 13)
```

**Message Arguments**

```c
typedef struct LIST_NOTIFY {
    MESSAGE msg;          // In: message to send/post
    P_ARGS pArgs;         // In: pArgs for message
    SIZEOF lenSend;       // In: length of pArgs
} LIST_NOTIFY, *P_LIST_NOTIFY;
```
msgListSend
Sends a message to each object in the list using ObjectSend.
Takes P_LIST_NOTIFY, returns STATUS.
#define msgListSend
MakeMsg(clsList, 14)
typedef struct LIST_NOTIFY {
  MESSAGE msg; // In: message to send/post
  P_ARGS pArgs; // In: pArgs for message
  SIZEOF lenSend; // In: length of pArgs
} LIST_NOTIFY, * P_LIST_NOTIFY;

Observer Notifications
A list uses msgPostObservers to deliver all of its notification messages. (See clsmgr.h for more information.)

msgListNotifyAddition
Notifies observers that an item has been added to the list.
Takes P_LIST_NOTIFY_ADDITION, returns STATUS.
#define msgListNotifyAddition
MakeMsg ( clsList, 16 )
typedef struct LIST_NOTIFY_ADDITION {
  LIST list; // the affected list
  LIST_ITEM listItem; // the affected list item
  U16 count; // new number of entries
  U8 reserved[40];
} LIST_NOTIFY_ADDITION, * P_LIST_NOTIFY_ADDITION;

msgListNotifyDeletion
Notifies observers that an item has been deleted from the list.
Takes P_LIST_NOTIFY_DELETION, returns STATUS.
#define msgListNotifyDeletion
MakeMsg ( clsList, 17 )
typedef struct LIST_NOTIFY_DELETION {
  LIST list; // the affected list
  LIST_ITEM listItem; // the affected list item
  U16 count; // new number of entries
  U8 reserved[40];
} LIST_NOTIFY_DELETION, * P_LIST_NOTIFY_DELETION;

Observer Notifications
Sends a message to each object in the list using ObjectSend.
Takes P_LIST_NOTIFY, returns STATUS.
#define msgListPost
MakeMsg(clsList, 15)
typedef struct LIST_NOTIFY {
  MESSAGE msg; // In: message to send/post
  P_ARGS pArgs; // In: pArgs for message
  SIZEOF lenSend; // In: length of pArgs
} LIST_NOTIFY, * P_LIST_NOTIFY;

Observer Notifications
Notifies observers that an item has been added to the list.
Takes P_LIST_NOTIFY_ADDITION, returns STATUS.
#define msgListNotifyAddition
MakeMsg ( clsList, 16 )
typedef struct LIST_NOTIFY_ADDITION {
  LIST list; // the affected list
  LIST_ITEM listItem; // the affected list item
  U16 count; // new number of entries
  U8 reserved[40];
} LIST_NOTIFY_ADDITION, * P_LIST_NOTIFY_ADDITION;

Observer Notifications
Notifies observers that an item has been deleted from the list.
Takes P_LIST_NOTIFY_DELETION, returns STATUS.
#define msgListNotifyDeletion
MakeMsg ( clsList, 17 )
typedef struct LIST_NOTIFY_DELETION {
  LIST list; // the affected list
  LIST_ITEM listItem; // the affected list item
  U16 count; // new number of entries
  U8 reserved[40];
} LIST_NOTIFY_DELETION, * P_LIST_NOTIFY_DELETION;
**msgListNotifyReplacement**

Notifies observers that an item in the list has been replaced.

Takes P_LIST_NOTIFY_REPLACEMENT, returns STATUS.

```c
typedef struct LIST_NOTIFY_REPLACEMENT {
    LIST list; // the affected list
    LIST_ITEM newItem; // the new list item
    LIST_ITEM oldListItem; // the replaced list item
    U16 index; // index of replace item
    U8 reserved[40];
} LIST_NOTIFY_REPLACEMENT, *P_LIST_NOTIFY_REPLACEMENT;
#define msgListNotifyReplacement MakeMsg ( clsList, 18 )
```

**msgListNotifyEmpty**

Notifies observers that a list is now empty.

Takes P_LIST_NOTIFY_EMPTY, returns STATUS.

```c
typedef struct LIST_NOTIFY_EMPTY {
    LIST list; // the affected list
    U8 reserved[40];
} LIST_NOTIFY_EMPTY, *P_LIST_NOTIFY_EMPTY;
#define msgListNotifyEmpty MakeMsg ( clsList, 19 )
```
This file contains the API definition for clsNotePaper. clsNotePaper inherits from clsView.

NotePaper is the view class for PenPoint's ink-management or note-taking building block. Most of the code for the MiniNote application actually resides in the building block. Other classes of the building block are clsNPData (the data class), clsNPItem (the generic data item), clsNPScribbleItem (the ink item), clsNPTextItem (the text data item), and clsGestureMargin (the subclass of clsScrollWin that implements MiniNote's gesture margin).

NotePaper provides standard PenPoint functionality including embedding, undo, move/copy, import, export, option sheets, and marks. (Supporting marks means that search and replace, spell, proof, and reference buttons are all supported.)

NotePaper displays (and alters) the contents of an NPData object. For PenPoint 1.0, NotePaper keeps all of the items in its data object in a coordinate system with (0,0) its upper-left corner. As a result, all the items in the data object have a negative y coordinate. This means that as the NotePaper window grows in width and height, its contents remain relative to the top-left corner of the page.

A sample application (called npapp or "NotePaper App") demonstrating the use of the ink building block is included in the SDK. The ink building block is distributed as part of the SDK as a distributed DLL. The DLL and all resources used by the ink building block are included in the SDK in the DLL\NOTEPAPR directory. The resources in that directory include:

- notepaper.res: contains all resources used by NotePaper
- paper.res: contains the 8 bitmaps representing paper styles
- pen.res: contains the 4 bitmaps representing pen styles
- strings.rc: contains the source for quick help, error text, and undo strings

```
#ifndef NOTEPAPR_INCLUDED
#define NOTEPAPR_INCLUDED
#endif

#ifndef VIEW_INCLUDED
#include <view.h>
#endif

#ifndef SYSFONT_INCLUDED
#include <sysfont.h>
#endif

#ifndef ITOGGLE_INCLUDED
#include <itoggle.h>
#endif
```

### Types and Constants

```
#define clsNotePaper MakeGlobalWKN(2567,1)

#define stsNotePaperNoHit MakeWarning(clsNotePaper, 0)
#define stsNotePaperTreatAsInk MakeWarning(clsNotePaper, 1)

Enum16 (NP_PAPER_STYLE) {
  npPaperRuled = 0,
  npPaperRuledLeftMargin = 1,
  npPaperRuledCenterMargin = 2,
  npPaperRuledLegalMargin = 7,
  npPaperBlank = 3,
  npPaperLeftMargin = 4,
  npPaperCenterMargin = 6,
  npPaperGrid = 5,
};
```
typedef struct NOTE_NP_PAPER_STYLE {
    U16 bEditMode : 1, // writing/ink vs. gesture/edit mode
    bAutoGrow : 1, // auto grow height as user enters data?
    bWidthOpt : 1, // include page widths in option sheet
    bHideTopRule : 1, // don't paint the top ruling line for
    bVirtualHeight : 1, // the npPaperRuledxxx paper style
    reserved : 11; // always set to 0
} NOTE_NP_PAPER_STYLE;

typedef struct NOTE_NP_PAPER_METRICS {
    NOTE_NP_PAPER_STYLE style;
    SYSDC_FONT_SPEC paperFont; // defines the font for the paper
    NP_PAPER_STYLE paperStyle; // one of the NP_PAPER_STYLE values
    COORD16 lineSpacing; // (in points) determines font size and
                         // vertical spacing
    U8 penStyle; // use the NPPenStyle() macro
} NOTE_NP_PAPER_METRICS;

NOTE: in NPPenStyle, color is one of: bsInkBlack, bsInkGrayXX, or bsInkWhite

NOTE: in NPPenStyle, weight is one of: 1 = bold, 0 = normal

#define NPPenStyle(color, weight) (((color & 0x7) | ((weight & 0x1) << 3))
#define NPPenColor(style) (style & 0x7)
#define NPPenWeight(style) ((style & 0x8) >> 3)

The following definitions are included for convenience only.

#define npPenFineBlack NPPenStyle(bsInkBlack, 0)
#define npPenFineGray NPPenStyle(bsInkGray50, 0)
#define npPenBoldBlack NPPenStyle(bsInkBlack, 1)
#define npPenBoldGray NPPenStyle(bsInkGray50, 1)

---

### Messages

Next up: none; Recycle: 11-51 53 58-101 103 106 120-127

#### msgNewDefaults

Initialize pArgs.

Takes P_NOTE_PAPER_NEW, returns STATUS.

```c
typedef struct {
    NOTE_PAPER_STYLE style; // as in NOTE_PAPER_METRICS
    NP_PAPER_STYLE paperStyle; // as in NOTE_PAPER_METRICS
    SYSDC_FONT_SPEC paperFont; // as in NOTE_PAPER_METRICS
    COORD16 lineSpacing; // as in NOTE_PAPER_METRICS
    U8 penStyle; // as in NOTE_PAPER_METRICS
    S32 spares[6];
} NOTE_PAPER_NEW_ONLY, *P_NOTE_PAPER_NEW_ONLY;
#define notePaperNewFields \
    viewNewFields \
    NOTE_PAPER_NEW_ONLY notePaper;
```
Zeroes out pArgs->notePaper and sets:

- pArgs->notePaper.style.bEditMode = false;
- pArgs->notePaper.style.bAutoGrow = false;
- pArgs->notePaper.style.bWidthOpts = false;
- pArgs->notePaper.style.bHideTopRule = false;
- pArgs->notePaper.style.bVirtualHeight = false;
- pArgs->notePaper.paperStyle = npPaperRuled;
- pArgs->notePaper.paperFont = current user font preference
- pArgs->notePaper.penStyle = NPPenStyle(bsInkBlack, 1);
- pArgs->notePaper.lineSpacing = 24; // 24 point

Various gWin and win flags are set and should only be modified by the fearless!

- pArgs->gWin.style.gestureEnable = true;
- pArgs->gWin.style.gestureForward = true;
- pArgs->win.flags.input &= -inputInkThrough;
- pArgs->win.flags.input |= inputInk;
- pArgs->win.flags.style |= wsSendGeometry;
- pArgs->win.flags.style |= wsGrowBottom;
- pArgs->win.flags.style |= wsGrowRight;
- pArgs->win.flags.style |= wsCaptureGeometry;

**msgNotePaperGetMetrics**

Passes back receiver's metrics.

Takes P_NOTE_PAPER_METRICS, returns STATUS.

```c
#define msgNotePaperGetMetrics MakeMsg(clsNotePaper, 101)
```

```c
typedef struct NOTE_PAPER_METRICS {
    NOTE_PAPER_STYLE style;
    SYSDC_FONT_SPEC paperFont;  // defines the font for the paper
    NP_PAPER_STYLE paperStyle;  // one of the NP_PAPER_STYLE values
    COORD16 lineSpacing;  // (in points) determines font size and
                         // vertical spacing
    U8 penStyle;  // use the NPPenStyle() macro
} NOTE_PAPER_METRICS, *P_NOTE_PAPER_METRICS;
```

**msgNotePaperGetDcInfo**

Passes back the drawing contexts used by receiver.

Takes P_NOTE_PAPER_DC_INFO, returns STATUS.

```c
#define msgNotePaperGetDcInfo MakeMsg(clsNotePaper, 4)
```

```c
typedef struct {
    U32 units;  // currently, msgDcUnitsTwips
    OBJECT dc;  // transformed dc in "units"
    OBJECT dcPen;  // transformed dc in pen units
    U32 reserved[4];
} NOTE_PAPER_DC_INFO, *P NOTE_PAPER_DC_INFO;
```

**msgNotePaperGetSelType**

Passes back information about the types of items selected in receiver.

Takes P_NOTE_PAPER_SEL_TYPE, returns STATUS.

```c
#define msgNotePaperGetSelType MakeMsg(clsNotePaper, 116)
```
typedef struct NOTE_PAPER_SEL_TYPE {
    BOOLEAN bScribble;  // selection contains a scribble
    BOOLEAN bTranslated; // selection contains untranslatable text
    BOOLEAN bReserved1;
    BOOLEAN bReserved2;
} NOTE_PAPER_SEL_TYPE, * P_NOTE_PAPER_SEL_TYPE;

msgNotePaperSetEditMode
Sets receiver to either gesture/edit (true) or writing/ink (false) mode.
Takes BOOLEAN, returns STATUS.
#define msgNotePaperSetEditMode MakeMsg(clsNotePaper, 102)

msgNotePaperSetPaperAndPen
Sets paperStyle, lineSpacing, penColor, and penWeight.
Takes P_NOTE_PAPER_METRICS, returns STATUS.
#define msgNotePaperSetPaperAndPen MakeMsg(clsNotePaper, 104)

typedef struct NOTE_PAPER_METRICS {
    NOTE_PAPER_STYLE style;
    SYSDC_FONT_SPEC paperFont;  // defines the font for the paper
    NP_PAPER_STYLE paperStyle;   // one of the NP_PAPER_STYLE values
    COORD16 lineSpacing;  // (in points) determines font size and
                           // vertical spacing
    U8 penStyle;  // use the NPPenStyle() macro
} NOTE_PAPER_METRICS, * P_NOTE_PAPER_METRICS;

This message does not affect the pen style for selected items.

msgNotePaperSetPenStyle
Sets the pen style for selected items as well as the default for new items.
Takes U32, returns STATUS.
#define msgNotePaperSetPenStyle MakeMsg(clsNotePaper, 109)

msgNotePaperGetPenStyle
Gets the pen style for selected items (or the default if nothing selected).
Takes U32, returns STATUS.
#define msgNotePaperGetPenStyle MakeMsg(clsNotePaper, 112)

msgNotePaperSetStyle
Sets the receiver's style values.
Takes P_NOTE_PAPER_STYLE, returns STATUS.
#define msgNotePaperSetStyle MakeMsg(clsNotePaper, 2)
typedef struct NOTE_NP_PAPER_STYLE {
    U16  bEditMode : 1, // writing/ink vs. gesture/edit mode
    bAutoGrow : 1, // auto grow height as user enters data?
    bWidthOpts : 1, // include page widths in option sheet
    bHideTopRule : 1, // don't paint the top ruling line for
    bVirtualHeight : 1, // if set, NotePaper grows itself into
    reserved : 11; // a long thin window and responds to
    U16  reserved1;
} NOTE_PAPER_STYLE, *P_NOTE_PAPER_STYLE;

msgNotePaperGetStyle
Passes back the receiver's style values.
Takes P_NOTE_PAPER_STYLE, returns STATUS.
#define msgNotePaperGetStyle MakeMsg(clsNotePaper, 3)

msgNotePaperTranslate
Translates untranslated scribbles in the selection.
Takes P_NULL, returns STATUS.
#define msgNotePaperTranslate MakeMsg(clsNotePaper, 113)

msgNotePaperUntranslate
Untranslates translated scribbles in the selection.
Takes P_NULL, returns STATUS.
#define msgNotePaperUntranslate MakeMsg(clsNotePaper, 114)

msgNotePaperEdit
Edits text and translates and edits scribbles in the selection.
Takes P_NULL, returns STATUS.
#define msgNotePaperEdit MakeMsg(clsNotePaper, 115)

msgNotePaperTidy
Tidies the selection by normalizing the spacing of items each line.
Takes P_NULL, returns STATUS.
#define msgNotePaperTidy MakeMsg(clsNotePaper, 105)

Comments
The inter-item spacing is determined by sending msgNPItemGetWordSpacing to each item to be tidied.
**msgNotePaperCenter**
Centers the entire selection.
Takes P_NULL, returns STATUS.
#define msgNotePaperCenter MakeMsg(clsNotePaper, 107)
Comments
The selection is centered on the page as a whole, not line by line.

**msgNotePaperAlign**
Aligns the selection according to pArgs.
Takes U32, returns STATUS.
#define msgNotePaperAlign MakeMsg(clsNotePaper, 108)
#define npAlignLeft 1
#define npAlignRight 2
Comments
Alignment takes place relative to the bounding box of the selection.

**msgNotePaperMerge**
Joins scribbles and text in the selection.
Takes P_NULL, returns STATUS.
#define msgNotePaperMerge MakeMsg(clsNotePaper, 110)
Comments
Consecutive scribble items are combined into a single scribble item. Adjacent text items are combined into a single text item. Any subclass of clsNPltem that can respond to msgNPltemCanJoin and msgNPltemJoin can determine its own merging behavior.

**msgNotePaperSplit**
Splits scribbles and text.
Takes P_NULL, returns STATUS.
#define msgNotePaperSplit MakeMsg(clsNotePaper, 111)
Comments
First msgNotePaperSplitAsWords is self-sent. If stsRequestDenied is returned, then msgNotePaperSplitAsAtoms is self-sent.

**msgNotePaperAddMenus**
Modifies the passed in menu bar and appends standard NotePaper menus.
Takes OBJECT, returns STATUS.
#define msgNotePaperAddMenus MakeMsg(clsNotePaper, 117)

**msgNotePaperAddModeCtrl**
Adds the standard NotePaper mode icon to the passed in menu bar.
Takes OBJECT, returns STATUS.
#define msgNotePaperAddModeCtrl MakeMsg(clsNotePaper, 118)
msgNotePaperClear
Deletes all items in receiver.
Takes pNull, returns STATUS.

#define msgNotePaperClear MakeMsg(clsNotePaper, 119)

msgNotePaperClearSel
Deletes all selected items in receiver.
Takes pNull, returns STATUS.

#define msgNotePaperClearSel MakeMsg(clsNotePaper, 11)

msgNotePaperInsertLine
Inserts a blank line above the selection.
Takes P_NULL, returns STATUS.

#define msgNotePaperInsertLine MakeMsg(clsNotePaper, 5)

msgNotePaperSelectRect
Selects items within rect in the receiver's data.
Takes P_RECT32, returns STATUS.

#define msgNotePaperSelectRect MakeMsg(clsNotePaper, 1)

Return Value
stsNotePaperNoHit Returned if nothing selected.

msgNotePaperSelectLine
Selects items whose baselines intersect rect in the receiver's data.
Takes P_RECT32, returns STATUS.

#define msgNotePaperSelectLine MakeMsg(clsNotePaper, 6)

Return Value
stsNotePaperNoHit Returned if nothing selected.

msgNotePaperDeselectLine
Deselects items whose baselines intersect rect in the receiver's data.
Takes P_RECT32, returns STATUS.

#define msgNotePaperDeselectLine MakeMsg(clsNotePaper, 7)

Return Value
stsNotePaperNoHit Returned if nothing deselected.

msgNotePaperDeleteLine
Deletes items whose baselines intersect rect in the view's data.
Takes P_RECT32, returns STATUS.

#define msgNotePaperDeleteLine MakeMsg(clsNotePaper, 8)

Return Value
stsNotePaperNoHit Returned if nothing deleted.
**msgNotePaperScribble**
Handles scribble (including creating and insert object into view's data).
Takes OBJECT, returns STATUS.

```c
#define msgNotePaperScribble MakeMsg(clsNotePaper, 9)
```

Comments
The passed scribble's origin should be relative to the lower-left corner of the receiver.

**msgGWinGesture**
Self-sent to process the gesture.
Takes P_GWIN_GESTURE, returns STATUS.

```c
#define msgGWinGesture MakeMsg(clsGWin, 2)
```

Comments
The standard behavior of this gesture is defined in gwin.h. In addition, subclasses can return `stsNotePaperTreatAsInk` if they want the gesture to be treated as ink. In that case, an instance of `clsNPScribbleItem` will be created from the gesture's strokes.

`clsNotePaper`'s response to the various gestures is described in the MiniNote quick reference card. In gesture mode, gesture can be made anywhere in the window. However, any unrecognized gesture of more than two strokes will be treated as ink. In writing mode, most drawing is treated as ink (unless it is drawn over the selection). However, the following gestures are allowed even in writing mode:

- `xgsScratchOut`: delete items
- `xgsPigtailVert`: delete items
- `xgs2Tap`: select item (if over an item)
- `xgs3Tap`: select line
- `xgsPlus`: toggle item (if over an item)
- `xgsTapHold`: begin area selection
- `xgsCircleCrossOut`: undo
- `xgsDblCircle`: create reference button
- `xgsUpCaretDot`: insert date/time
- `xgsDblUpCaret`: embed stationery
- `xgsHorzCounterFlick`: toggle mode
- `xgsVertCounterFlick`: toggle application borders

**Return Value**
`stsNotePaperTreatAsInk` The gesture should be treated as ink.

**See Also**
gwin.h

**msgAppSelectAll**
Selects all items in the view.
Takes P_NULL, returns STATUS.

**See Also**
app.h

**msgSelDelete**
Deletes selected items in the view.
Takes P_NULL, returns STATUS.

Comments
Close the space that the selection occupies if an entire line or lines is selected and this message does is not sent within a move/copy episode.

**See Also**
sel.h
msgOptionAddCards
Creates and adds the Pen and Paper option sheets.
Takes P_OPTION_TAG, returns STATUS.
This message is usually send to the NotePaper instance by the app framework if the instance holds the selection, is the client win of the app's main win, or is the client win of a scroll win that is the app's main win. However, to force NotePaper's option sheets to appear in the "Option" menu in other circumstances, this message should be forwarded to the NotePaper instance by the application if pArgs->tag is tagAppDocOptSheet.

See Also
app.h

msgImportQuery
Indicates whether or not passed in file can be imported.
Takes P_IMPORT_QUERY, returns STATUS. Category: class message.
NotePaper will respond positively to this message if the first 50f the file are printable ASCII characters.

See Also
import.h

msgImport
Imports the passed in file.
Takes P_IMPORT_DOC, returns STATUS.
After the file is imported, receiver's length is grown to accommodate imported text. If receiver's width is zero, it is grown to sixwide.

See Also
import.h

msgExportGetFormats
Passes back list of formats that can be exported.
Takes P_EXPORT_LIST, returns STATUS.

See Also
export.h

msgExport
Writes an ASCII version of receiver's data to the passed in file.
Takes P_EXPORT_DOC, returns STATUS.
A translated text version of each scribble item is written out.

See Also
export.h

Quick help and window tags
Tags used in the UI of NotePaper's option sheets, menus, and quick help.
Next up 37; Recycle: 2
Tag values 100-120 are reserved for pen and paper styles.
Tag values 200-255 are reserved for private window tags.
Mode icons

Mode icons (tags from itoggle.h) The bitmaps corresponding to the two tags below are found in the SystemResFile.

```c
#define tagNotePaperWriteIcon tagIconToggleOff
#define tagNotePaperEditIcon tagIconToggleOn
```

Quick help tag for mode icons

```c
#define tagNotePaperModeIcon MakeTag(clsNotePaper, 1)
```

Windows

Quick help tags for the main view and for the gesture margin.

```c
#define tagNotePaper MakeTag(clsNotePaper, 4)
#define tagNotePaperMargin MakeTag(clsNotePaper, 5)
```

Edit Menu

```c
#define tagNotePaperTranslate MakeTag(clsNotePaper, 6)
#define tagNotePaperEdit MakeTag(clsNotePaper, 7)
#define tagNotePaperClear MakeTag(clsNotePaper, 34)
#define tagNotePaperInsertLine MakeTag(clsNotePaper, 35)
```

Pen Menu

```c
#define tagPenMenu MakeTag(clsNotePaper, 3)
#define tagPenFineBlack MakeTag(clsNotePaper, 110)
#define tagPenBoldBlack MakeTag(clsNotePaper, 111)
#define tagPenFineGray MakeTag(clsNotePaper, 112)
#define tagPenBoldGray MakeTag(clsNotePaper, 113)
```

Arrange Menu

```c
#define tagArrangeMenu MakeTag(clsNotePaper, 8)
#define tagNotePaperTidy MakeTag(clsNotePaper, 9)
#define tagNotePaperCenter MakeTag(clsNotePaper, 10)
#define tagNotePaperAlignLeft MakeTag(clsNotePaper, 11)
#define tagNotePaperAlignRight MakeTag(clsNotePaper, 12)
#define tagNotePaperMerge MakeTag(clsNotePaper, 13)
#define tagNotePaperSplitAsWords MakeTag(clsNotePaper, 14)
#define tagNotePaperSplit MakeTag(clsNotePaper, 15)
```

Paper Option Card

NOTE: For TagPaperStyle(n), tag n is a value in the NP_PAPER_STYLE enumeration For NPPaperStyleFromTag converts a tag to a paper style.

```c
#define tagPaperCard MakeTag(clsNotePaper, 16)
#define tagPaperStyleLabel MakeTag(clsNotePaper, 17)
#define tagPaperStyle MakeTag(clsNotePaper, 18)
#define TagPaperStyle(n) MakeTag(clsNotePaper, 100 + n)
#define NPPaperStyleFromTag(t) (TagNum(t) - 100)
#define tagLineSpacingLabel MakeTag(clsNotePaper, 19)
#define tagLineSpacing MakeTag(clsNotePaper, 20)
#define tagLineOtherRuling MakeTag(clsNotePaper, 21)
#define tagLineOtherValue MakeTag(clsNotePaper, 22)
```
## Pen Option Card

- **tagPenCard**
- **tagPenStyleLabel**
- **tagPenStyle**

  - `tagPenFineBlack` (same value as in the pen menu)
  - `tagPenBoldBlack` (same value as in the pen menu)
  - `tagPenFineGray` (same value as in the pen menu)
  - `tagPenBoldGray` (same value as in the pen menu)

## Insertion Pad

- **tagNotePaperSkip**

## Standard Error Resource Tags

- **stsNotePaperPageWidth**

## Undo Resource Tags

- **tagNPUndoWriting**
- **tagNPUndoDeletion**
NPDATA.H

This file contains the API definition for clsNPData.

clsNPData inherits from clsObject.

NPData is the data class of PenPoint's ink-management or note-taking building block. (See notepapr.h for more information on the building block.) An NPData instance is a data base that manages items that follow the clsNPItem protocol. (See npitem.h). Its API defines messages for inserting, deleting, and enumerating the items it manages.

 ifndef NPDATA_INCLUDED
 #define NPDATA_INCLUDED
 ifndef CLSMGR_INCLUDED
 #include <clsmgr.h>
 #endif
 #include <geo.h>

 **Types and Constants**

 #define clsNPData MakeGlobalWKN(2568,1)

 **Messages**

 Next up: 39; Recycle: 4 5 6 7 15 20 33 34

 **msgNewDefaults**

 Initialize pArgs.

 Takes P_NP_DATA_NEW, returns STATUS.

 **Arguments**

typedef struct {
    XY32 lineSpacing;
    XY32 baseline;
    BOOLEAN isSubData; // private to clsNPData
    S32 spare1;
    S32 spare2;
} NP_DATA_NEW_ONLY, *P_NP_DATA_NEW_ONLY;
#define npDataNewFields
    objectNewFields
    NP_DATA_NEW_ONLY npData;

typedef struct {
    npDataNewFields
} NP_DATA_NEW, *P_NP_DATA_NEW;

 **Comments**

 Zeros out pArgs->npData and sets:

 pArgs->npData.lineSpacing.x = 0;
pArgs->npData.lineSpacing.y = 360;  // 360 twips = 18 points = 1/4"  
pArgs->npData.baseline.x = 0;
pArgs->npData.baseline.y = 360;
**Messages used to manipulate data**

**msgNPDataInsertItem**
Add item to the data base.
Takes OBJECT, returns STATUS.

```c
#define msgNPDataInsertItem MakeMsg(clsNPData, 8)
```

**msgNPDataInsertItemFromView**
Add item to the data base.
Takes P_NP_DATA_ADDED_NP_ITEM_VIEW, returns STATUS.

```c
#define msgNPDataInsertItemFromView MakeMsg(clsNPData, 38)
```

**Arguments**
- typedef struct {
  OBJECT item; // item that has been added
  OBJECT view; // view that added the item
} NP_DATA_ADDED_NP_ITEM_VIEW, *P_NP_DATA_ADDED_NP_ITEM_VIEW;

**Comments**
Observers will be notified of which view is responsible for the addition.

**msgNPDataDeleteItem**
Delete an item from the data base.
Takes OBJECT, returns STATUS.

```c
#define msgNPDataDeleteItem MakeMsg(clsNPData, 9)
```

**Comments**
Returns stsFailed if item is not found.

**msgNPDataMoveItem**
Move an item within the data base.
Takes P_NP_DATA_XY, returns STATUS.

```c
#define msgNPDataMoveItem MakeMsg(clsNPData, 10)
```

**Arguments**
- typedef struct {
  OBJECT item; // item to be moved
  XY32 xy; // new position for item
} NP_DATA_XY, *P_NP_DATA_XY;

**msgNPDataMoveItems**
Move all items below pArgs->y by pArgs->yDelta.
Takes P_MOVE_ITEMS, returns STATUS.

```c
#define msgNPDataMoveItems MakeMsg(clsNPData, 1)
```

**Arguments**
- typedef struct {
  COORD32 y;
  COORD32 yDelta;
} MOVE_ITEMS, *P_MOVE_ITEMS;
**Messages used to enumerate over data**

### ENUM_CALLBACK

This template describes the callback function used in item enumeration.

Returns STATUS.

**Arguments**

typedef struct {
    OBJECT data;     // in - the data being enumerated over
    OBJECT item;     // in - the item being enumerated
    P_UNKNOWN clientData;     // in - the client supplied data (or pointer)
} NP_DATA_ITEM, *P_NP_DATA_ITEM;

typedef STATUS FunctionPtr(P_ENUM_CALLBACK)(P_NP_DATA_ITEM pItem);

**Comments**

Your callback function takes a single parameter of type P_NP_DATA_ITEM. The clientData field is a copy of that you passed into the enumeration message using the ENUM_ITEM or ENUM_RECT_ITEM structures. During enumeration, you can add new items or delete the "current" item being enumerated. If you delete an item but want to keep using it, use must send it msgNPItemHold before deleting it and msgNPItemRelease when you are done using it.

Some of the enumeration messages refer to bPaintOrder or "Reverse" order. Paint order refers to the top-to-bottom, left-to-right ordering of items. Non-paint or reverse order is simply the opposite ordering. Items are sorted first by line and then by their left edge. An item is considered to be on the line closest to its baseline. The lines are "line spacing" apart starting from the top of the page. If no lines are displayed to the user, it is possible that non-intuitive item ordering will result.

Return an error status from the callback to terminate the enumeration.

### msgNPDataEnumOverlappedItems

Enumerates each item that overlaps the given rectangle.

Takes P_ENUM_RECT_ITEMS, returns STATUS.

#define msgNPDataEnumOverlappedItems MakeMsg(clsNPData, 2)

typedef struct {
    P_ENUM_CALLBACK function;     // in -- callback function described above
    RECT32 hitRect;     // in -- enum items overlapping hitRect
    BOOLEAN bPaintOrder;     // in -- enum in paint order?
    P_UNKNOWN clientData;     // in
} ENUM_RECT_ITEMS, *P_ENUM_RECT_ITEMS;

### msgNPDataEnumBaselineItems

Enumerates each item whose baseline overlaps the given rectangle.

Takes P_ENUM_RECT_ITEMS, returns STATUS.

#define msgNPDataEnumBaselineItems MakeMsg(clsNPData, 19)

typedef struct {
    P_ENUM_CALLBACK function;     // in -- callback function described above
    RECT32 hitRect;     // in -- enum items overlapping hitRect
    BOOLEAN bPaintOrder;     // in -- enum in paint order?
    P_UNKNOWN clientData;     // in
} ENUM_RECT_ITEMS, *P_ENUM_RECT_ITEMS;
msgNPDataEnumSelectedItems
Enumerates each item that is selected (in paint order).
Takes P_ENUM_ITEMS, returns STATUS.
#define msgNPDataEnumSelectedItems MakeMsg(clsNPData, 13)

typedef struct {
P_ENUM_CALLBACK function;  // in -- callback function described above
P_UNKNOWN clientData;     // in
} ENUM_ITEMS, *P_ENUM_ITEMS;

msgNPDataEnumSelectedItemsReverse
Enumerates each item that is selected (in reverse paint order).
Takes P_ENUM_ITEMS, returns STATUS.
#define msgNPDataEnumSelectedItemsReverse MakeMsg(clsNPData, 26)

typedef struct {
P_ENUM_CALLBACK function;  // in -- callback function described above
P_UNKNOWN clientData;     // in
} ENUM_ITEMS, *P_ENUM_ITEMS;

msgNPDataEnumAllItems
Enumerates each item (in paint order).
Takes P_ENUM_ITEMS, returns STATUS.
#define msgNPDataEnumAllItems MakeMsg(clsNPData, 14)

typedef struct {
P_ENUM_CALLBACK function;  // in -- callback function described above
P_UNKNOWN clientData;     // in
} ENUM_ITEMS, *P_ENUM_ITEMS;

msgNPDataEnumAllItemsReverse
Enumerates each item (in reverse paint order).
Takes P_ENUM_ITEMS, returns STATUS.
#define msgNPDataEnumAllItemsReverse MakeMsg(clsNPData, 27)

typedef struct {
P_ENUM_CALLBACK function;  // in -- callback function described above
P_UNKNOWN clientData;     // in
} ENUM_ITEMS, *P_ENUM_ITEMS;

msgNPDataSendEnumSelectedItems
Enumerates each selected item (in paint order).
Takes P_SEND_ENUM_ITEMS, returns STATUS.
#define msgNPDataSendEnumSelectedItems MakeMsg(clsNPData, 22)

typedef struct {
P_ENUM_CALLBACK function;  // in -- callback function described above
U8 clientData[32];        // in/out
} SEND_ENUM_ITEMS, *P_SEND_ENUM_ITEMS;
**msgNPDataGetCurrentItem**

Passes back the current item in the receiver.

Takes P_OBJECT, returns STATUS.

```c
#define msgNPDataGetCurrentItem MakeMsg(clsNPData, 30)
```

**msgNPDataGetNextItem**

Increments the current item to the next item and sets *pArgs to it.

Takes P_OBJECT, returns STATUS.

```c
#define msgNPDataGetNextItem MakeMsg(clsNPData, 31)
```

**msgNPDataItemCount**

Passes back the count of items in receiver.

Takes P_U32, returns STATUS.

```c
#define msgNPDataItemCount MakeMsg(clsNPData, 17)
```

**msgNPDataSelectedCount**

Passes back the count of selected items in receiver.

Takes P_U32, returns STATUS.

```c
#define msgNPDataSelectedCount MakeMsg(clsNPData, 18)
```

**msgNPDataSetBaseline**

Sets the receiver's baseline (used for alignment).

Takes P_XY32, returns STATUS.

```c
#define msgNPDataSetBaseline MakeMsg(clsNPData, 24)
```

**msgNPDataGetBaseline**

Gets the receiver's baseline (used for alignment).

Takes P_XY32, returns STATUS.

```c
#define msgNPDataGetBaseline MakeMsg(clsNPData, 25)
```
msgNPDataSetLineSpacing
Sets receiver's line spacing (used as the font size).
Takes P_XY32, returns STATUS.
#define msgNPDataSetLineSpacing MakeMsg(clsNPData, 35)

msgNPDataGetLineSpacing
Gets receiver's line spacing (used as the font size).
Takes P_XY32, returns STATUS.
#define msgNPDataGetLineSpacing MakeMsg(clsNPData, 36)

msgNPDataGetBounds
Passes back the bounding rectangle for all items in receiver.
Takes P_RECT32, returns STATUS.
#define msgNPDataGetBounds MakeMsg(clsNPData, 23)

msgNPDataGetSelBounds
Passes back the bounding rectangle for all selected items in receiver.
Takes P_RECT32, returns STATUS.
#define msgNPDataGetSelBounds MakeMsg(clsNPData, 32)

msgNPDataGetFontSpec
Passes back the receiver's font specification.
Takes P_SYSDC_FONT_SPEC, returns STATUS.
#define msgNPDataGetFontSpec MakeMsg(clsNPData, 28)

msgNPDataSetFontSpec
Sets the receiver's font specification.
Takes P_SYSDC_FONT_SPEC, returns STATUS.
#define msgNPDataSetFontSpec MakeMsg(clsNPData, 29)

msgNPDataGetCachedDCs
Passes back DC's with normal and bold fonts at the given line spacing.
Takes P_NP_DATA_DC, returns STATUS.
#define msgNPDataGetCachedDCs MakeMsg(clsNPData, 37)

Arguments
typedef struct {
    OBJECT dcNormal; // normal font dc
    OBJECT dcBold; // bold font dc
} NP_DATA_DCS, *P_NP_DATA_DCS;

Comments
Used by items that want to measure text without the overhead of creating a DC. These DC's cannot be used for drawing!!
Messages sent to observers

msgNPDataAddedItem
Observers notified when item has been added or moved.
Takes P_NP_DATA_ADDED_ITEM, returns STATUS. Category: observer notification.

#define msgNPDataAddedItem MakeMsg(clsNPData, 11)

typedef struct {
    OBJECT data; // the data that the item has been added to
    OBJECT item; // item that has been added
    OBJECT view; // view that added the item
} NP_DATA_ADDED_ITEM, *P_NP_DATA_ADDED_ITEM;

msgNPDataItemChanged
Observers notified when item has been changed.
Takes P_NP_DATA_ITEM_CHANGED, returns STATUS. Category: observer notification.

#define msgNPDataItemChanged MakeMsg(clsNPData, 12)

typedef struct {
    OBJECT data; // the data
    OBJECT item; // item that has been changed
    OBJECT view; // view that changed the item
    RECT32 bounds; // maximum bounds affected by the change
} NP_DATA_ITEM_CHANGED, *P_NP_DATA_ITEM_CHANGED;

msgNPDataHeightChanged
Observers notified when receiver's height has been changed.
Takes P_NP_DATA_ITEM_CHANGED, returns STATUS. Category: observer notification.

#define msgNPDataHeightChanged MakeMsg(clsNPData, 21)

typedef struct {
    OBJECT data; // the data
    OBJECT item; // item that has been changed
    OBJECT view; // view that changed the item
    RECT32 bounds; // maximum bounds affected by the change
} NP_DATA_ITEM_CHANGED, *P_NP_DATA_ITEM_CHANGED;

msgNPDataItemEnumDone
Observers notified when an enumeration that deleted or moved items is complete.
Takes NULL, returns STATUS. Category: observer notification.

#define msgNPDataItemEnumDone MakeMsg(clsNPData, 16)

When this message is received by an observer client, all deletions have been completed and all moved items have been temporarily removed from the data object. Thus the client has the option of repainting all remaining items at this time and then painting moved items as they are reinserted.

This message is handled by clsNotePaper and should not be handled by subclasses of clsNotePaper.
This file contains the API definition for clsNPItem.

clsNPItem inherits from clsObject.

NPItem is the item class for PenPoint's ink-management or note-taking building block. While instances of clsNPItem are never created, (subclasses like clsNPScribbleItem and clsNPTextItem are more interesting), NPItem defines a protocol as well as doing much of the work for basic operations.

To add new item types to the ink building block, create a subclass of clsNPItem that implements the messages defined below in the section: "Messages that are usually overridden by subclasses." Once this new item is inserted into a clsNPData object it will show up in the clsNotePaper view that observes that object. The new item will then behave like the other item in terms of basic operations like move, copy, deletion, style changes, etc.

```
#ifndef NPITEM_INCLUDED
#define NPITEM_INCLUDED
#endif

#ifndef GEO_INCLUDED
#include <geo.h>
#endif

#ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif

#ifndef BORDER_INCLUDED
#include <border.h>
#endif
```

### Types and Constants

```
#define clsNPItem MakeGlobalWKN(2569, 1)
#define stsNPItemNoSplit MakeWarning(clsNPItem, 0)
```

The NPData object handles versioning for NPItem's and their subclasses. If the version of the object being restored matches the runtime version, nothing special is done. However, if there is a difference, the version number of the filed object is stamped as a U16 property onto the file using tagItemVersion as the property's tag.

```
#define NP_ITEM_VERSION 1
#define tagItemVersion MakeTag(clsNPItem, 0)
```

### Messages

Next up: 44; Recycle: 3

**msgNewDefaults**

Initialize pArgs.

Takes P_NP_ITEM_NEW, returns STATUS.

```
typedef struct NP_ITEM_NEW_ONLY {
    RECT32 bounds;
    XY16 baseline;
    BOOLEAN selected;
    U32 penStyle; // (Pen styles are defined in notepapr.h.)
    S32 spare2;
} NP_ITEM_NEW_ONLY, *P_NP_ITEM_NEW_ONLY;
```
#define npItemNewFields
  objectNewFields
  NP_ITEM_NEW_ONLY item;

typedef struct NP_ITEM_NEW {
  npItemNewFields
} NP_ITEM_NEW, *P_NP_ITEM_NEW;

Zeroes out pArgs->npData and sets:
  pArgs->item.penStyle = penFineBlack;

msgNPItemGetPenStyle
Get the pen style of an item. (Pen styles are defined in noteAppr.h.)
Takes P_U32, returns STATUS.
#define msgNPItemGetPenStyle   MakeMsg(clsNPItem, 35)

msgNPItemDelete
Delete item from its data.
Takes pNull, returns STATUS.
#define msgNPItemDelete MakeMsg(clsNPItem, 11)

Deleting an item decrements its reference count and can cause the item to be destroyed. To prevent, call msgNPItemHold before calling msgNPItemDelete. Then call msgNPItemRelease after working with the item.

msgNPItemPaintBackground
Paints a gray background if the receiver is selected.
Takes P_NP_ITEM_DC, returns STATUS.
#define msgNPItemPaintBackground MakeMsg(clsNPItem, 41)

typedef struct {
  OBJECT dc;       // DC to paint into
  OBJECT dcPen;    // equivalent DC in pen units
} NP_ITEM_DC, *P_NP_ITEM_DC;

Subclasses should override this message if they want a different type of selection feedback.

msgNPItemSelect
Selects or deselects item.
Takes BOOLEAN, returns STATUS.
#define msgNPItemSelect MakeMsg(clsNPItem, 14)

msgNPItemSelected
Passes back item's selection status.
Takes P_BOOLEAN, returns STATUS.
#define msgNPItemSelected MakeMsg(clsNPItem, 15)
**msgNPItemMove**
Moves item to the indicated position.
Takes P_XY32, returns STATUS.
#define msgNPItemMove MakeMsg(clsNPItem, 5)

**msgNPItemDelta**
Moves item by the indicated amount.
Takes P_XY32, returns STATUS.
#define msgNPItemDelta MakeMsg(clsNPItem, 6)

**msgNPItemGetViewRect**
Passes back receiver's bounding rectangle.
Takes P_RECT32, returns STATUS.
#define msgNPItemGetViewRect MakeMsg(clsNPItem, 19)

**msgNPItemHitRect**
Returns stsOK if receiver's bounds overlaps pArgs.
Takes P_RECT32, returns STATUS.
#define msgNPItemHitRect MakeMsg(clsNPItem, 9)

**msgNPItemGetMetrics**
Gets the item's metrics.
Takes P_NP_ITEM_METRICS, returns STATUS.
#define msgNPItemGetMetrics MakeMsg(clsNPItem, 20)

```c
typedef struct NP_ITEM_METRICS {
  U8 selected: 1, // is item selected?
  marked: 1, // is item marked (in the clsMark sense)?
  reserved: 6;
  U8 refCount; // number external references to item
  // (not generally interesting to subclasses)
  XY16 baseline; // item's horizontal and vertical baseline
  // (currently only the y value is used)
  RECT32 bounds; // window relative bounds
  // (with respect to its bounds' origin)
  OBJECT data; // data object that item is in
  OBJECT adjunct; // see msgNPItemSetAdjunct for more information
  U32 penStyle; // item's pen style
} NP_ITEM_METRICS, *P_NP_ITEM_METRICS;
```

**msgNPItemSetBaseline**
Sets receiver's baseline.
Takes P_XY32, returns STATUS.
#define msgNPItemSetBaseline MakeMsg(clsNPItem, 21)
**msgNPltemSetBounds**

Sets receiver's bounds.

Takes P_RECT32, returns STATUS.

```c
#define msgNPltemSetBounds MakeMsg(clsNPltem, 30)
```

**msgNPltemHold**

Increments the reference count for the item.

Takes NULL, returns STATUS.

```c
#define msgNPltemHold MakeMsg(clsNPltem, 22)
```

**msgNPltemRelease**

Decrements the reference count for the item.

Takes NULL, returns STATUS.

```c
#define msgNPltemRelease MakeMsg(clsNPltem, 23)
```

**msgNPltemAlignToBaseline**

Moves item so that it align to passed in line spacing.

Takes P_XY32, returns STATUS.

```c
#define msgNPltemAlignToBaseline MakeMsg(clsNPltem, 33)
```

**msgNPltemPaint**

Paints item using the passed in drawing contexts.

Takes P_NP_ITEM_DC, returns STATUS.

```c
#define msgNPltemPaint MakeMsg(clsNPltem, 12)
```

```c
typedef struct {
    OBJECT dc; // DC to paint into
    OBJECT dcPen; // equivalent DC in pen units
} NP_ITEM_DC, *P_NP_ITEM_DC;
```

**msgNPltemSetPenStyle**

Sets the item's pen style. (Pen styles are defined in notepapr.h.)

Takes U32, returns STATUS.

```c
#define msgNPltemSetPenStyle MakeMsg(clsNPltem, 34)
```
msgNPltemSetOrigin
Set receiver's origin.
Takes P_XY32, returns STATUS.
#define msgNPltemSetOrigin MakeMsg(clsNPltem, 18)

msgNPltemScratchOut
Handles the scratch-out gesture on an item.
Takes P_RECT32, returns STATUS.
#define msgNPltemScratchOut MakeMsg(clsNPltem, 24)

Comments
Scribble items handle this message by deleting strokes that overlap pArgs. Other items simply delete themselves.

msgNPltemSplitGesture
Handles the split gesture on an item.
Takes P_XY32, returns STATUS.
#define msgNPltemSplitGesture MakeMsg(clsNPltem, 25)

Comments
The pArgs refers to the "hot point" for the gesture.

msgNPltemSplit
Split an item into its constituent items.
Takes NULL, returns STATUS.
#define msgNPltemSplit MakeMsg(clsNPltem, 26)

msgNPltemSplitAsWords
Splits receiver into words. Deletes receiver, inserts new items.
Takes NULL, returns STATUS.
#define msgNPltemSplitAsWords MakeMsg(clsNPltem, 16)

Return Value
stsItemNoSplit Returned if nothing was split.

msgNPltemJoin
Joins receiver and OBJECT and deletes OBJECT.
Takes OBJECT, returns STATUS.
#define msgNPltemJoin MakeMsg(clsNPltem, 27)

msgNPltemTie
Joins OBJECT and receiver and deletes them. Inserts new object.
Takes OBJECT, returns STATUS.
#define msgNPltemTie MakeMsg(clsNPltem, 17)
msgNPItemGetScribble
Pass back the item's scribble.
Takes P_OBJECT, returns STATUS.
#define msgNPItemGetScribble MakeMsg(clsNPItem, 4)

Comments
Subclasses that do not contain a scribble should not respond to this message.

msgNPItemGetString
Passes back the text string for the item.
Takes PP_STRING, returns STATUS.
#define msgNPItemGetString MakeMsg(clsNPItem, 38)

Comments
Subclasses that do not have a text representation should not respond to this message.
clsNPScribbleItem responds to this message by translating its scribble and returning the resulting string. The sender of this message should either use the passed back string immediately or make a copy of it.

msgNPItemSetString
Sets the text string for the item.
Takes P_STRING, returns STATUS.
#define msgNPItemSetString MakeMsg(clsNPItem, 42)

Comments
Not all items can handle this message.

msgNPItemToText
Item converts itself to a text item, passes back text item.
Takes P_OBJECT, returns STATUS.
#define msgNPItemToText MakeMsg(clsNPItem, 7)

Comments
Receiver deletes itself from its data and inserts the text item. If pArgs is pNull, the text item is not passed back.

msgNPItemToScribble
Item converts itself to a scribble item.
Takes P_ARGS, returns STATUS.
#define msgNPItemToScribble MakeMsg(clsNPItem, 36)

Comments
Receiver deletes itself from its data and inserts the scribble item.

msgNPItemHitRegion
Returns stsOK if receiver's path overlaps pArgs.
Takes P_RECT32, returns STATUS.
#define msgNPItemHitRegion MakeMsg(clsNPItem, 10)
**msgNPItemCalcBaseline**
Calculate and set receiver's baseline.
Takes P_XY32, returns STATUS.

#define msgNPItemCalcBaseline MakeMsg(clsNPItem, 28)

Comments
The calculation is based on the line spacing specified by pArgs.

**msgNPItemCalcBounds**
Receiver calculates and sets its new bounds.
Takes OBJECT, returns STATUS.

#define msgNPItemCalcBounds MakeMsg(clsNPItem, 37)

Comments
Usually send in response to the item's style changing. OBJECT is the data object in which the item will be inserted. If the item is in a data object, pArgs can be pNull.

**msgNPItemGetWordSpacing**
Receiver passes back the size of its "space" character.
Takes P_U16, returns STATUS.

#define msgNPItemGetWordSpacing MakeMsg(clsNPItem, 43)

Comments
This message is used by msgNotePaperTidy to determine the spacing of items.

**msgNPItemCanBeTranslated**
Receiver returns stsOK if it can be translated.
Takes pNull, returns STATUS.

#define msgNPItemCanBeTranslated MakeMsg(clsNPItem, 13)

Comments
Translation occurs in response to msgNPItemToText.

**msgNPItemCanBeUntranslated**
Receiver returns stsOK if it can be untranslated.
Takes pNull, returns STATUS.

#define msgNPItemCanBeUntranslated MakeMsg(clsNPItem, 31)

Comments
Untranslation occurs in response to msgNPItemToScribble.
This file contains the API definition for clsNPScribbleItem.

clsNPScribbleItem inherits from clsNPItem.

NPScribbleItem is the ink class of PenPoint's ink-management or note-taking building block. (See notepap.h for more information on the building block.) NPScribbleItem overrides NPItem messages as is appropriate. See npitem.h for details.

```
#ifndef NPSCR_INCLUDED
#define NPSCR_INCLUDED
#ifndef NPITEM_INCLUDED
#include "npitem.h"
#endif
```

### Types and Constants

```
#define clsNPScribbleItem MakeGlobalWKN(2570,1)
```

### Messages

**msgNewDefaults**


Takes P_NP_SCRIBBLE_ITEM_NEW, returns STATUS.

```
typedef struct NP_SCRIBBLE_ITEM_NEW_ONLY {
    OBJECT scribble;
    OBJECT data;    // data that item will be associated with
    S32 spare1;
} NP_SCRIBBLE_ITEM_NEW_ONLY, *P_NP_SCRIBBLE_ITEM_NEW_ONLY;
#define npScribbleItemNewFields
    npItemNewFields
    NP_SCRIBBLE_ITEM_NEW_ONLY scribbleItem;
typedef struct NP_SCRIBBLE_ITEM_NEW {
    npScribbleItemNewFields
    NP_SCRIBBLE_ITEM_NEW, *P_NP_SCRIBBLE_ITEM_NEW;
```
NPTEXT.H

This file contains the API definition for clsNPTextItem.

clsNPTextItem inherits from clsNPItem.

NPTextItem is the text class of PenPoint's ink-management or note-taking building block. (See notepapapr.h for more information on the building block.) NPTextItem overrides NPItem messages as is appropriate. See npitem.h for details.

```c
#ifndef NPTEXTINCLUDED
#define NPTEXTINCLUDED
#ifndef NPITEMINCLUDED
#include "npitem.h"
#include "notepapr.h"
#endif
#endif
```

**Types and Constants**

```c
#define clsNPTextItem MakeGlobalWKN(2571,1)
```

**Messages**

msgNewDefaults

Initialize pArgs. Zeros out pArgs->textItem.

Takes P_NP_TEXT_ITEM_NEW, returns STATUS.

```c
typedef struct NP_TEXT_ITEM_NEW_ONLY {
    OBJECT text;          // string object
    P_STRING pString;     // string if string object not given
    OBJECT data;          // data that item will be associated with
                           // (item's size measured using data's DC)
    S32 spare1;
} NP_TEXT_ITEM_NEW_ONLY, *P_NP_TEXT_ITEM_NEW_ONLY;
#define npTextItemNewFields
#define npTextItemNewFields
    nflItemNewFields
    NP_TEXT_ITEM_NEW_ONLY textItem;

typedef struct NP_TEXT_ITEM_NEW {
    npTextItemNewFields
    } NP_TEXT_ITEM_NEW, *P_NP_TEXT_ITEM_NEW;
```

```c
```
This file contains the API definition for the OrderedSet interface. The functions described in this file are contained in MISCLIB.

Overview

An OrderedSet implements a growable, ordered set of items. Each item has a key and associated data. The ordered set knows about the structure of the key, but treats the data as uninterpreted bytes. The items in an ordered set are homogeneous: there is only one size for the key, and another size for the data, for all the items in the set.

Keys are unsigned quantities, treated as either non-negative integers or indirect access identifiers. The client specifies:
• how keys are treated - direct or indirect;
• for indirect keys - access and comparison functions;
• whether duplicate keys are allowed;
• the key size - it must be 1, 2, or 4 bytes.

The data size (in bytes) is also specified by the client; it must be less than or equal to 1023.

The client provides an initial estimate of the number of items in the ordered set when the set is created; the set will allocate more memory if the estimate proves to be too small.

Performance considerations

The implementation of OrderedSet builds on the ByteArray storage abstraction. This implies that either the number of elements in the set is small enough that it is not a problem to use a linear array representation for the set, or that the number of lookups dominates the number of insertions and deletions.

Indirect Keys and Two Comparison Routines

Ordered sets with indirect keys have a funny property. If you want to search for a key that already exists in the set, everything's just fine. But if you want to do something with a key that ISN'T in the set (e.g. find out if the key is in the set), there is no indirect key to use. (This problem also arises when clients ask ordered sets questions such as "What's the next entry with a key greater than this key k?")

To solve this problem, indirect-keyed ordered sets must be provided two comparison routines by the creator. The first routine (passed as the compareKey1Indirect in a called to OrderedSetExtend()) is used when the implementation needs to compare two keys that are both in the set. The second routine (passed as compareKey1Direct in a call to OrderedSetExtend()) is used when the implementation needs to compare two keys, only one of which is in the set.

Caution:
If keys are indirect, OrderedSetFindMinMax(), OrderedSetFindMaxMin(), and OrderedSetNext() return the indirect key, not the value the key references.

**Known Limitations**

This package does not work correctly if the set has indirect keys and 0 (zero) is a legitimate key value.

```c
#ifndef ORDSET_INCLUDED
#define ORDSET_INCLUDED $Revision: 1.17 $
#include <bytarray.h>
#include <gosearch.h> // For ACCESS/COMPARE_FUNC

/* Private */
typedef U32 (CDECL *READ_KEY_FUNC) (
P_ORDERED_SET p,
P_UNKNOWN pKey);
typedef struct ORDERED_SET {
  U16 indirectKeys : 1;
  U16 uniqueKeys : 1;  // TRUE => no duplicate keys
  U16 spare : 2;  // Always set to 0
  U16 sizeofKeyMinus1 : 2;  // Number of bytes -1 a key needs
  U16 sizeofData :10;  // Number of bytes data occupies
  P_BYTE_ARRAY items;  // Storage of actual items
  ACCESS_FUNC access;
  COMPARE_FUNC compareKeyIdDirect;
  COMPARE_FUNC compareKeyIdIndirect;
  P_UNKNOWN context;  // 1st arg to access() & compare()
  READ_KEY_FUNC readKey;  // For internal use only!
} ORDERED_SET;

OrderedSetCountInternal

Returns the number of items currently stored in the ORDERED_SET.

Returns BYTE_INDEX.

```c
#define OrderedSetCountInternal(p) \n  (ByteArrayLength(p->items) 10rderedSetSizeofItem(p))
```n
**Comments**

High-performance version of OrderedSetCount, but subject to change if the implementation of ordered sets changes.

**Types and Constants**

```c
#define stsOrdSetDuplicateKey MakeStatus(clsMisc, 1)
#define findNextKeyInOS ((P_UNKNOWN)1)
#define findPreviousKeyInOS ((P_UNKNOWN)2)
typedef struct OS_ITEM_INFO {
  U32 key;
  P_UNKNOWN data;
  BOOLEAN isDuplicate;
} OS_ITEM_INFO, *P_OS_ITEM_INFO;
```n
# Exported Functions and Macros

## OrderedSetPrint
In debugging version, prints the contents of the ordered set.

Returns void.

```c
#ifdef DEBUG
void EXPORTED Function Pvt::totype OrderedSetPrint(
    P_ORDERED_SET p);
#endif  // DEBUG
```

Comments:
This function is undefined in the non-debugging version.

## OrderedSetCreate
Creates an ordered set.

Returns STATUS.

```c
STATUS EXPORTED OrderedSetCreate(
    P_ORDERED_SET * pp,
    OS_HEAP_MODE mode,
    U8 sizeofKey,
    U8 sizeofData,
    U32 initialCount,
    BOOLEAN uniqueKeys,
    BOOLEAN indirectKeys);
```

Comments:
`sizeofKey` and `sizeofData` specify the size in bytes of each item's key and data, respectively. The `initialCount` is a hint: the ordered set will grow or shrink as needed. However, if `initialCount` is approximately correct, performance will be better. If `initialCount=0`, 1 will be assumed. `uniqueKeys` should be `TRUE` if client wants all keys in the set to be unique, `FALSE` otherwise. Only the `osHeapLocal / osHeapShared` flags in `mode` are used.

Returns `stsOK` if able to create the set, in which case `*pp` will be the created set, otherwise `*pp` will be `Nil(p_ORDERED_SET)`.  

## OrderedSetSizeofKey
Returns the size of a key in bytes.

Returns `U16`.

```c
#define OrderedSetSizeofKey(p) 
    ((U16)(((p)->sizeofKeyMinus1 + 1))
```

## OrderedSetSizeofItem
Returns the size of an item (key plus data) in bytes.

Returns `U16`.

```c
#define OrderedSetSizeofItem(p) \ 
    ((U16)(OrderedSetSizeofKey(p) + (p)->sizeofData))
```
**OrderedSetHeapMode**

Returns the heap mode with which the Ordered Set was created.

Returns OS_HEAP_MODE.

```c
#define OrderedSetHeapMode(p) ByteArrayHeapMode((p)->items)
```

**OrderedSetExtend**

Modifies the functions and context of an ordered set with indirect keys.

Returns STATUS.

```c
void EXPORTED
OrderedSetExtend(
    P_ORDERED_SET p,
    ACCESS_FUNC access,
    COMPARE_FUNC compareKey1Direct,
    COMPARE_FUNC compareKey1Indirect,
    P_UNKNOWN context);
```

**OrderedSetContext**

Get the context passed to access and compare functions.

Returns P_UNKNOWN.

```c
#define OrderedSetContext(p) ((p)->context)
```

**OrderedSetModifyContext**

Modify the context passed to access and compare functions.

Returns void.

```c
#define OrderedSetModifyContext(p, _c) ((p)->context (_c))
```

**OrderedSetDefaultAccess**

Can be used as the client-specified access routine in `OrderedSetExtend()`.

Returns P_UNKNOWN.

```c
P_UNKNOWN CDECL
OrderedSetDefaultAccess(
    const P_ORDERED_SET p,
    const BYTE_INDEX index);
```

**Comments**

In ordered sets with indirect keys the client must supply a routine that returns the address of the keys that are passed into the client-supplied comparison routine. `OrderedSetDefaultAccess` computes the address of the key in the ordered set representation, and so may be used by clients as the access routine passed into `OrderedSetExtend()`.
**OrderedSetDestroy**

Destroys an ORDERED_SET.

Returns void.

void EXPORTED

**Function Prototype**

    OrderedSetDestroy( 
        P_ORDERED_SET p);

**OrderedSetInsert**

Inserts data with key into ordered set.

Returns STATUS.

STATUS EXPORTED

**Function Prototype**

    OrderedSetInsert( 
        P_ORDERED_SET p, 
        U32 key, 
        P_UNKNOWN data);

**Comments**

Copies sizeofData bytes from the buffer pointed to by data. Returns:

stsOSOutOfMem if no memory available, or

stsOrdSetDuplicateKey if key is duplicate and unique keys required, or

stsOK otherwise.

If sizeofKey is less than 4 bytes, the least significant byte(s) of key are copied.

**OrderedSetNthItem**

Locates the n-th item in the ordered set (item indices begin with 0).

Returns P_UNKNOWN.

P_UNKNOWN EXPORTED

**Function Prototype**

    OrderedSetNthItem( 
        P_ORDERED_SET p, 
        U32 n, 
        P_OS_ITEM_INFO info);

**Comments**

Returns a pointer to ordered set's copy of the data associated with the Nth item. This pointer is only valid until the next call on the same set.

Upon return, the following modifications have been made to the fields of info:

key key of nth item

isDuplicate is not set; use OrderedSetFind() if needed;

data duplicate of return value

**OrderedSetItemIndex**

Returns the index of an item

Returns BYTE_INDEX.

#define OrderedSetItemIndex(p, pData)  
    (ByteArrayFindIndex((p)->items, ((P_U8)(pData))) /  
     - OrderedSetSizeofKey(p) / OrderedSetSizeofItem(p))
**OrderedSetFind**

Locates the data for a specified key.

Returns P_UNKNOWN.

P_UNKNOWN EXPORTED

Function Prototype:

```c
OrderedSetFind(
    P_ORDERED_SET p,
    P_OS_ITEM_INFO info);
```

Comments:

Returns a pointer to ordered set's copy of the data associated with info->key. This pointer is only valid until the next call on the same set. If the info->key is not in the set, the returned value is Nil(P_UNKNOWN). If duplicate copies of the key exist in the set, an arbitrary item is found and its data returned. All of the other items with the same key may be examined via use of OrderedSetNext(). Upon return, the following modifications have been made to the fields of info:

- isDuplicate = 0 if key is unique in set, 1 otherwise
- data = duplicate of return value

**OrderedSetFindMinMax**

Locates the data for a key >= to specified key.

Returns P_UNKNOWN.

P_UNKNOWN EXPORTED

Function Prototype:

```c
OrderedSetFindMinMax(
    P_ORDERED_SET p,
    P_OS_ITEM_INFO info);
```

Comments:

Returns a pointer to ordered set's copy of the data associated with the minimum key in the ordered set that is >= info->key. If info->key is in the ordered set, this routine is equivalent to OrderedSetFind(). This pointer is only valid until the next call on the same set. Returns Nil(PUNKNOWN) if info->key has no minmax in the set. If duplicate copies of the minmax key exist in the set, an arbitrary item is found and its data returned. All of the other items with the same key may be retrieved with OrderedSetNext(). Upon return, the following modifications have been made to the fields of info:

- key = minmax key
- isDuplicate = 0 if key is unique in set, 1 otherwise
- data = duplicate of return value

**OrderedSetFindMaxMin**

Locates the data for a key <= to specified key.

Returns P_UNKNOWN.

P_UNKNOWN EXPORTED

Function Prototype:

```c
OrderedSetFindMaxMin(
    P_ORDERED_SET p,
    P_OS_ITEM_INFO info);
```

Comments:

Returns a pointer to ordered set's copy of the data associated with the maximum key in the ordered set that is <= info->key. If info->key is in the ordered set, this routine is equivalent to OrderedSetFind(). This pointer is only valid until the next call on the same set. Returns Nil(P_UNKNOWN) if info->key has no maxmin in the set. If duplicate copies of the maxmin key exist in the set, an arbitrary item is found and its data returned. All of the other items with the same key may be retrieved with OrderedSetNext(). Upon return, the following modifications have been made to the fields of info:

- key = maxmin key
- isDuplicate = 0 if key is unique in set, 1 otherwise
- data = duplicate of return value
and its data returned. All of the other items with the same key may be retrieved with OrderedSetNext().
Upon return, the following modifications have been made to the fields of info:

key: maxmin key
isDuplicate: 0 if key is unique in set, 1 otherwise
data: duplicate of return value

**OrderedSetNext**
Enumerates the data for keys in the Ordered Set.

Returns P UNKNOWN.

P UNKNOWN EXPORTED

Function Prototype

```c
OrderedSetNext(
P_ORDERED_SET p,
P_OS_ITEM_INFO info);
```

Comments

OrderedSetNext()'s behavior depends on whether the set has unique keys or not. In both cases, the
enumeration is guaranteed to be complete provided no insertions or deletions are performed on the
set during the enumeration.

- **IF THE SET HAS UNIQUE KEYS**

OrderedSetNext() enumerates all of the keys in the set in order.
The first item in the enumeration can be found by either (1) by calling OrderedSetNthItem() with
an "N" of 0 or (2) calling OrderedSetNext() with info->data set to Nil and info->key set to the
lowest possible key value.

- **IF THE SET DOES NOT HAVE UNIQUE KEYS**

OrderedSetNext() enumerates all of the keys with the same value. The order of enumeration is
unspecified.
The first item with a known key can be found by either (1) by calling OrderedSetFind with
info->key set to the known key value and info->data set to Nil

- **IN BOTH CASES**

Further items are found by calling OrderedSetNext() with the same info struct until it returns Nil.
OrderedSetNext() returns a pointer to the set's copy of the data associated with key. This pointer is
only valid until the next call on the same set.

Returns

Nil(P UNKNOWN) if specified key not in set or the enumeration is complete, or
pointer to set's copy of data or if key is in set or enumeration is incomplete.

Upon return, the following modifications have been made to the fields of info:

key: next key value, iff info->data had been one of the
three special values: Nil, next, prev.
isDuplicate: 0 if key is unique in set,
1 otherwise
data: duplicate of returned value
FOR SETS WITH DIRECT, NON-DUPLICATE KEYS ONLY

If the set has direct keys, setting info->data to findNextKeyInOS (findPreviousKeyInOS), OrderedSetNext() can be used to enumerate all items in the set in order of increasing (decreasing) key value. Such an enumeration (assuming non-unique keys) will have the structure:

```c
info.key = 0;
info.data = Nil(P_UNKNOWN);
if ((firstData = OrderedSetNext(...)) == Nil(P_UNKNOWN)) {
  info.data = findNextKeyInOS;
  if ((firstData = OrderedSetNext(...)) == Nil(P_UNKNOWN)) {
    // handle empty set
    ...
  }
}
// firstData and info now contain first item's information
```

// enumerate all keys
do {
  // enumerate all data with the same key
  while (OrderedSetNext(...)) {
    ...
  }
  info.data = findNextKeyInOS;
} until (!OrderedSetNext(...));

```

**OrderedSetEachItem**

Helper macro to simplify the enumeration of an Ordered Set.

Returns P_UNKNOWN.

```c
#define OrderedSetEachItem(_p, _item) \
   for ((_item).key = (U32)0, (_item).data = Nil(P_UNKNOWN); \
      OrderedSetNext(_p, &(_item)) != Nil(P_UNKNOWN);) \
      // The condition IS the iteration step
   ...

Comments

This macro is only useful for sets with direct, non-duplicate keys!

The arguments to OrderedSetEachItem() are:

- _p the ordered set to enumerate
- _item an OS_ITEM_INFO containing the enumerated item's info

Code using these macros should look like: OS_ITEM_INFO scratch; OrderedSetEachItem(os, scratch) {
  myPtr = (MY_PTR)scratch.data; ...
}

**OrderedSetDelete**

Deletes specified item from the Ordered Set.

Returns STATUS.

```
STATUS EXPORTED

Function Prototype

OrderedSetDelete(
    P_ORDERED_SET p,
    P_OS_ITEM_INFO info);
```

Comments

If duplicates are allowed, both info->key and info->data must be filled in by client; if keys are unique, only info->key need be filled in.
Returns:
stsOK if item was found in set and deleted, or
stsNoMatch if item not found in set, or
STATUS < 0 if internal error during deletion.

**OrderedSetCount**

Returns the number of items currently stored in the ORDERED_SET.

Returns U32.

U32 EXPORTED

Function Prototype: OrderedSetCount(
  P_ORDERED_SET p);
This file contains the API definition for clsQuickHelp.

clsQuickHelp inherits from clsFrame.

clsQuickHelp provides an interface to the Quick Help Server.

theQuickHelp is a well-known instance of clsQuickHelp.

theQuickHelp provides system wide quick help, and is the only instance of clsQuickHelp in the system, built at boot time. Clients should not create instances of this object, nor should they subclass this object. This file defines an interface to display quick help text in the standard quick help window. Programmers should rarely have to call ANY of the functions in this file, as default calling of quick help is provided by default in clsGWin (see gwin.h). However, some applications may need to invoke quick help, or change the quick help text, hence the public message to open quick help, and to show a quick help screen.

A quick help resource consists of a string array resource with each array item mapping to a single quick help panel. This resource is identified by creating a List resource ID from the administered portion of the quick help ID (MakeListResId(helpID, resGrpQhelp, 0)) and the quick help group. The TAG portion of the quick help ID is used to index into the string array. Each quick help strings will have two "parts". The first part will be the title and the second part will be the text. The title will be separated from the text by including two vertical line characters (||) following the title which will NOT be printed.

These resources, which are defined below, are put into the application resource files and displayed using msgQuickHelpShow, which takes the resource ID. As mentioned, gWin defines a default behavior for calling the object with this message. All application typically need to do is provide their gWin objects (or subclasses) with helpId resources.

Quick help for an object is generally displayed in one of two ways. The first is when an object decides to display quick help for itself. An example is gWin's response to the '?' gesture. gWin posts theQuickHelp msgQuickHelpShow, which opens the quick help window and displays quick help for the object. The second is when theQuickHelp window is open, and the system is in quick help mode. When the user taps on objects on the screen, the object is sent msgQuickHelpHelpShow. The object will respond by posting msgQuickHelpHelpShow back to theQuickHelp. When the quick help window is dismissed, by hitting closed or envoke help notebook, the object that received msgQuickHelpHelpShow will receive msgQuickHelpHelpDone. This message will also be sent when tapping on successive objects while in quick help mode. It will not be sent when quick help was initially brought up directly from the object when it posted msgQuickHelpShow (such as the gWin response to the '?' gesture.

#ifndef QHELP_INCLUDED
#define QHELP_INCLUDED
#endif
#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif
#ifndef RESFILE_INCLUDED
#include <resfile.h>
#endif
Debugging Flags

Quick Help uses the debugging flag set ‘q’. Defined flags are:

0001 General quick help debugging information

Types and Constants

These tags are used for defining three quick help screens: 1) the quick help intro screen that gives directions on quick help, 2) the "No help available" screen, and 3) the help not found screen.

`#define hlpQuickHelpSignOn MakeTag(clsQuickHelp, 1)`
`#define hlpQuickHelpNoHelp MakeTag(clsQuickHelp, 2)`
`#define hlpQuickHelpNotFound MakeTag(clsQuickHelp, 3)`

Messages

**msgQuickHelpShow**
Displays the Quick Help associated with the resource ID.

Takes `P_QUICK_DATA`, returns `STATUS`.

```c
#define msgQuickHelpShow MakeMsg(clsQuickHelp, 1)
```

**Arguments**

typedef struct QUICK_DATA {
    U32 helpId; // Help ID of the screen to show
    OBJECT appUID; // UID of the application. Used to find resources
                    // of application specific help IDs.
    U32 reserved; // Reserved for future use
} QUICK_DATA, *P_QUICK_DATA;

**Comments**

Gets the quick help resource from either the system resource files or the application specific resource files. If the quick help resource can't be found, will display the "Quick help not found" message in the quick help screen. Typically called from `gWin` in order to display the help screen for a help gesture. Would take the `gWin helpId` and the application uid. Needs the application object in order to reference the resource files of the application to find application specific help IDs. Typically not called directly by applications, but called indirectly through `gWin` inheritance. Will call `msgQuickHelpOpen` to open the quick help window as necessary.

Typically called by objects in response to a ? gesture, or in response to `msgQuickHelpHelpShow`.

**See Also**

gwin.h

**msgQuickHelpHelpShow**

Sent to a window to display a quick help request.

Takes `P_XY32`, returns `STATUS`.

```c
#define msgQuickHelpHelpShow MakeMsg(clsQuickHelp, 7)
```

**Comments**

Sent from the `QuickHelp` to a window when it is required to display its quick help. Typically the window will respond by posting `msgQuickHelpShow`. Sent as the user taps on various windows while quick help is being displayed.

**See Also**

`msgQuickHelpHelpDone`
msgQuickHelpHelpDone
Sent to a window when quick help is no longer displayed.
Takes OBJECT, returns STATUS.
```
#define msgQuickHelpHelpDone MakeMsg(clsQuickHelp, 8)
```
Comments
Sent to the last object asked to display quick help via msgQuickHelpHelpShow when help is no longer needed on said object. Can be sent because the user tapped somewhere else and a new object is about to be sent msgQuickHelpHelpShow, quick help has been terminated by the user, or the help notebook has been entered. Takes the new object receiving a msgQuickHelpHelpShow if because the user tapped elsewhere, or null if quick help is being terminated or going to the help notebook. Note that this message is only sent to object which previously received msgQuickHelpHelpShow, and not those objects generating a help request by posting msgQuickHelpShow directly.

See Also
msgQuickHelpHelpShow

msgQuickHelpOpen
Forces the Quick Help window to appear.
Takes nothing, returns STATUS.
```
#define msgQuickHelpOpen MakeMsg(clsQuickHelp, 2)
```
Comments
Opens the quick help window on the screen. If the quick help window is already on the screen, will simply return stsOK. The quick help window is a modal filter that will grab all input till closed via msgQuickHelpClose. Self sent to when msgQuickHelpShow is posted. Also sent from the help notebook icon to invoke quick help.

Notification Messages

msgQuickHelpOpened
Indicates that the quick help window has been opened.
Takes nothing, returns STATUS. Category: observer notification.
```
#define msgQuickHelpOpened MakeMsg(clsQuickHelp, 128)
```
Comments
Sent to observers of the quick help that the quick help window has been opened.

msgQuickHelpClosed
Indicates that the quick help window has been closed.
Takes nothing, returns STATUS. Category: observer notification.
```
#define msgQuickHelpClosed MakeMsg(clsQuickHelp, 129)
```
Comments
Sent to observers of theQuickHelp to indicate that the quick help window has been closed.

msgQuickHelpInvokedNB
Indicates that the notebook associated with quick help should be open.
Takes nothing, returns STATUS. Category: observer notification.
```
#define msgQuickHelpInvokedNB MakeMsg(clsQuickHelp, 130)
```
Comments

Sent to observers when `msgQuickHelpInvokeNB` is received. The help note book is an observer, and will bring itself up when this message is received.
This file contains the API for clsSelection. 
clsSelection inherits from clsObject.
theSelectionManager provides management of the system-wide selection. theSelectionManager is the one and only instance of clsSelection.

Introduction

Much of PenPoint's user interface is based on the "selection." The selection is often the center of the user's attention. In general it is very easy for the user to set the selection -- it often just requires a tap.
The precise definition of the selection is application-specific. In text the selection is often a set of characters. In a spreadsheet it might be a range of rows, columns, or cells. In a Table of Contents it might be a set of documents. Typically, an application "highlights" the selection with a grey background, handles, or some other graphic technique.
Because the selection corresponds to the center of the user's attention, many user interface operations are based on the selection. Here are some examples:

• The selection is the source of PenPoint's move and copy operations.
• Typically, the selection is altered by Applying an Option Sheet.
• The selection often determines which menu items are enabled and which are disabled.
• The selection and keyboard input target are often linked together.

Programmatically, other objects can inquire about the selection, get information from the selection and transfer data from the selection.

Road Map

Use the following to take ownership of the selection:

• msgSelSetOwner
• msgSelSetOwnerPreserve
• msgSelSelect (if object has clsEmbeddedWin in the object's ancestry)

Selection owners must be prepared to handle the following:

• msgSelDelete
• msgSelYield
• msgSelBeginCopy
• msgSelBeginMove
• msgControlProvideEnable (see section "Control Enabling")
Use the following to inquire about the selection:
- `msgSelOwner`
- `msgSelPrimaryOwner`
- `msgSelOwners`
- `msgSelIsSelected` (if object has `clsEmbeddedWin` in the object's ancestry)

The `SelectionManager` sends the following notifications:
- `msgSelChangedOwners`
- `msgSelPromotedOwner`

Destinations of PenPoint's Move and Copy mechanism must handle the following:
- `msgSelCopySelection`
- `msgSelMoveSelection`

**Move and Copy**

`sel.h` defines several messages that are used to implement PenPoint's Move and Copy operations. These messages are used in combination with PenPoint's data transfer messages which are defined in `xfer.h`. (PenPoint data transfer does not always necessarily involve the selection, but when it does, the messages described here are employed.)

`clsEmbeddedWin` (see `embedwin.h`) provides the default response for several of the steps described below.

Here's the typical "flow of control" for moving selected data:
- The source object handles the "Press" gesture (`xgsPressHold` in `xgesture.h`). The object might receive this gesture if it is a `gWin` (see `gwin.h`).
- If the `Press` gesture is not over the selection, the object typically selects what is under the gesture. "Selecting" includes either (1) self sending `msgSelSelect` or (2) sending `msgSelSetOwner` to the `SelectionManager`, whichever is appropriate.
- Next the object self-sends `msgSelBeginMove`.
- `msgSelBeginMove` is received. Note that `msgSelBeginMove` is sent in other cases than the Press gesture response. For instance, the standard application menu item "Move" (in the "Edit" menu) results in the selection owner receiving `msgSelBeginMove`.
- In response to `msgSelBeginMove`, the receiver should self send `msgEmbeddedWinBeginMove`. `msgEmbeddedWinBeginMove` takes, in its `pArgs`, the hot point of the gesture that kicks off the move, and the bounds of the selection being moved.
- In response to `msgEmbeddedWinBeginMove`, `embeddedWin` creates the floating "move icon." `clsEmbeddedWin` manages the icon.
- The icon takes over at this point and manages the process of moving the selection.
- When the icon is dropped on a destination, the icon sends `msgMoveCopyIconDone` to the source.
- `clsEmbeddedWin` handles `msgMoveCopyIconDone` and sends `msgSelMoveSelection` to the destination.
• In response to msgSelMoveSelection, the destination object retrieves the selection owner from the selection manager (using msgSelOwner) and engage in an xfer protocol with the selection. (The xfer protocols are described in xfer.h) The data should be copied to the position contained in msgSelMoveSelection’s pArgs, which is a P_XY32.

• After the data has been copied from the selection owner, the destination should send msgSelDelete to the selection owner.

• The destination object should select the data that it just absorbed.

The "flow of control" for copying selected data is very similar, with the following changes:

• The gesture that kicks off the protocol is "Tap-Press" (xgsTapHold in xgesture.h) rather than Press-Hold.

• The source object self sends and handles msgSelBeginCopy rather than msgSelBeginMove. The source object self sends msgEmbeddedWinBeginCopy rather than msgEmbeddedWinBeginMove.

• The destination receives msgSelCopySelection rather than msgSelMoveSelection.

• The destination object should not send msgSelDelete.

See Also

xfer.h.h

Two Selection Owners

Some objects need to own the selection, but they need to take in a fashion that (1) allows PenPoint to restore the original selection and (2) allows client code to find the original selection. For example, Option Sheets apply to a selection. But the various controls that appear within the option sheet might need to own the selection as well. Both selections need to be maintained.

Therefore theSelectionManager actually manages two selection owners: a selection owner and a preserved selection owner.

NOTE: The same object cannot be both the selection owner and preserved selection owner. See the detailed comments with msgSelSetOwner and msgSelSetOwnerPreserve for details.

When an object needs to take the selection but allow the current selection to be restored, that object should take the selection via msgSelSetOwnerPreserve, which "preserves" or "remembers" the original selection. The preserved selection can be restored by sending msgSelOwnerPreserve with a pArgs of pNull to theSelectionManager. Hence objects in option sheets take the selection via msgSelSetOwnerPreserve.

Essentially all clients should operate on the selection owner. This includes move and copy operations. The only client that should operate on the preserved selection owner, if one exists, is option sheets.

Control Enabling

Some controls, particularly menu items, should be disabled if there is no selection owner. And some controls should be disabled based on application-specific details about the selection state.

For instance, the "Move," "Copy," and "Delete" menu items should not be enabled if there is no selection owner. The "Move" menu item should be enabled if there is a selection and the selection owner is not read-only. The "Delete" menu item should be enabled if there is a selection owner and the contents of the selection are not empty.

To support this, clsControl allows control creators to specify that the control should send msgControlProvideEnable to the selection owner to get the proper enable/disable state.
Some standard application menus (SAMs) are set up to send `msgControlProvideEnable` to the selection owner. See `app.h` for details.

Therefore all selection owners should handle `msgControlProvideEnable`.

### Relationship of Selection to the Input Target

The input system's "Target" is the object to which keyboard events are sent. See `input.h` for more information.

Because the selection is normally the center of the user's attention, it often makes sense for the same object to own the selection and to be the input target. For instance, PenPoint's text component always becomes the input target whenever it takes the selection and sets the input target to null when it yields the selection.

There are, however, cases where it makes more sense to NOT link the selection and input target together. For instance, some types of fields take the input target without taking the selection. The decision is quite application-specific.

Implementing a correspondence between the input target and selection ownership is the client's responsibility.

### What to Do When the Selection Changes Within an Owner

Some parts of PenPoint's UI depend on knowing when the user's center of attention changes. For instance, each time that an Option Sheet is notified that the selection has changed it checks to be sure that the top card is still applicable.

Therefore, selection owners should set the selection to self EVERY TIME THE SELECTION CHANGES within them, even if they are already the selection owner. This lets observers take any appropriate action.

### Only One Instance

There is one and only one instance of `clsSelection`, and that instance is the global well-known `theSelectionManager`.

```c
#ifndef SEL_INCLUDED
#define SEL_INCLUDED

#ifndef CLSMGR_INCLUDED
#define CLSMGR_INCLUDED
#include <clsmgr.h>
#endif
#endif
```

### Common #defines and typedefs

### Status Codes

The `theSelectionManager` returns `stsSelYieldInProgress` when the selection manager is in the process of sending `msgSelYield` and therefore can't respond to the message.

```c
#define stsSelYieldInProgress MakeWarning(clsSelection,l)
```
**Types**

preservedOwner is defined only if havePreservedOwner is true. It IS possible to have a null preservedOwner.

```c
typedef struct SEL_OWNERS {
    OBJECT owner;
    OBJECT preservedOwner;
    BOOLEAN havePreservedOwner;
} SEL_OWNERS, *P_SEL_OWNERS;
```

**Messages Sent to theSelectionManager**

// Next Up: 26, Recycled: 9, 14, 20

---

**msgSelSetOwner**

Sets the selection owner.

Takes OBJECT, returns STATUS.

```c
#define msgSelSetOwner MakeMsg(clsSelection,2)
```

Comments

Send msgSelSetOwner to theSelectionManager to set the selection owner. theSelectionManager responds in one of the following ways:

If pArgs is not a valid selection owner (because it can't be called from other objects or is not a global object):

- theSelectionManager returns stsScopeViolation.

If pArgs is null, theSelectionManager:

- sends msgSelYield to the current selection if it exists and sets the current selection to null.
- sends msgSelYield the current preserved selection if it exists and sets the current preserved selection to null.
- sends msgSelChangedOwners to theSelectionManager's observers.

Otherwise, theSelectionManager:

- sends msgSelYield to the current preserved selection if it exists and is not equal to pArgs. theSelectionManager then sets the preserved selection to null and stops observing the preserved selection.
- sends msgSelYield to the current selection if it exists and is not equal to pArgs.
- sets the current selection to pArgs.
- adds itself as an observer of the new selection.
- sends msgSelChangedOwners to theSelectionManager's observers.

**Return Value**

stsScopeViolation pArgs is not a valid selection owner.

**See Also**

msgSelSetOwnerPreserve

---

**msgSelSetOwnerPreserve**

Sets the selection owner with the preserve option.

Takes OBJECT, returns STATUS.

```c
#define msgSelSetOwnerPreserve MakeMsg(clsSelection,5)
```
Send `msgSelSetOwnerPreserve` to `theSelectionManager` to set the selection owner while preserving the current selection owner.

See the section "Two Selection Owners" for more information.

`theSelectionManager`'s response to this message is similar to its response to `msgSelSetOwner`, with only subtle differences.

If `pArgs` is null, and there is no preservedOwner:
- `theSelectionManager` simply returns `stsOK`.

If `pArgs` is null, and a preserved owner exists (even if it is null), `theSelectionManager`:
- sends `msgSelYield` to the current owner if it exists.
- sends `msgSelPromote` to the current preserved owner if non-null.
- sets the current owner to the current preserved owner if non-null.
- sets the current preserved owner to null.
- sets the value for `SEL_OWNERS.havePreservedOwner` to false.
- sends `msgSelPromotedOwner` to `theSelectionManager`'s observers.

If `pArgs` is non-null but is not a valid selection owner (because it can't be called from other objects or is not a global object):
- `theSelectionManager` returns `stsScopeViolation`.

If `pArgs` is a valid selection owner and there is a no preserved owner:
- sends `msgSelDemote` to the current owner.
- sets the current preserved owner to be the current owner.
- sets the current owner to be `pArgs`.
- adds itself as an observer of the new selection.
- sets the value for `SEL_OWNERS.havePreservedOwner` to true.
- sends `msgSelChangedOwners` to `theSelectionManager`'s observers.

If `pArgs` is a valid selection owner and there is a preserved owner:
- sends `msgSelYield` to the current owner if it exists and is not the same as `pArgs`.
- sets the current owner to `pArgs`.
- adds itself as an observer of the new selection.
- sends `msgSelChangedOwners` to `theSelectionManager`'s observers.

**msgSelOwner**

Passes back the selection owner.

Takes `P_OBJECT`, returns `STATUS`.

```c
#define rnmsgSelOwner MakeMsg(clsSelection,1)
```
theSelectionManager passes back the current selection owner. It does not pass back the preserved selection owner.

**Return Value**

stsSelYieldInProgress theSelectionManager is currently sending msgSelYield.

---

**msgSelPrimaryOwner**

Passes back the primary selection owner.

Takes P_OBJECT, returns STATUS.

#define msgSelPrimaryOwner MakeMsg(clsSelection, 7)

The "primary owner" is the selection owner which an option sheet applies to. If there is a preserved selection owner, the primary owner is the preserved owner. Otherwise, the primary selection owner is the current owner.

**Return Value**

stsSelYieldInProgress theSelectionManager is currently sending msgSelYield.

See Also

msgSelSetOwner

---

**msgSelOwners**

Passes back the selection and preserved owners.

Takes P_SEL_OWNERS, returns STATUS.

#define msgSelOwners MakeMsg(clsSelection, 4)

typedef struct SELOwners {
    OBJECT owner,
    OBJECT preservedOwner;
    BOOLEAN havePreservedOwner;
} SELOwners, *P_SEL_OWNERS;

theSelectionManager posts msgSelOwners to its observers to inform the observers that the selection owner and/or preserved owner has been set. (The notification is sent even if the new owner is null.)
theSelectionManager sends this notification even if the old owner and new owner are the same. Hence if object A is the selection owner, and msgSelSetOwner is sent with object A, msgSelChangedOwners is sent to theSelectionManager's observers.

When a preserved selection owner is promoted back to the selection owner, msgSelPromotedOwner is sent rather than msgSelChangedOwners.

Example of use: In response to this message, option sheets check the applicability of the top card.

See Also

msgSelSetOwner

---

msgSelPromotedOwner

Notifies observers when the preserved owner has been promoted back to the selection owner.

Takes P_SEL_OWNERS, returns STATUS.

#define msgSelPromotedOwner MakeMsg(clsSelection, 8)

typedef struct SEL_OWNERS {
  OBJECT owner;
  OBJECT preservedOwner;
  BOOLEAN havePreservedOwner;
} SEL_OWNERS, *P_SEL_OWNERS;

theSelectionManager posts msgSelPromotedOwner to its observers to inform the observers that preserved selection owner has been promoted to the normal selection owner.

This happens as a result of theSelectionManager handling msgSelSetOwnerPreserve with a pArgs of null.

See Also

msgSelSetOwnerPreserve

---

**Messages Sent by theSelectionManager to Owners**

---

msgSelYield

theSelectionManager requires the release of the selection.

Takes BOOLEAN, returns STATUS.

#define msgSelYield MakeMsg(clsSelection, 11)

theSelectionManager sends this message to a selection owner to inform the object that it is no longer the selection owner. pArgs is true if object is yielding the primary selection and false when the object is yielding the preserved selection.

This message is not sent when an object takes the selection via msgSelSetOwner or msgSelSetOwnerPreserve and it already is the selection, or already is the preserved selection. (However, msgSelChangedOwners IS sent to theSelectionManager's observers.)

When handling this message, be careful about sending selection manager messages (such as msgSelSetOwner) as deadlock can occur.

After sending msgSelYield, theSelectionManager removes itself as an observer of the object.

See Also

msgSelSetOwner
**msgSelDemote**

Informs the owner that it is becoming the preserved owner.

Takes nothing, returns STATUS.

```c
#define msgSelDemote MakeMsg(clsSelection, 24)
```

Comments

The `SelectionManager` sends this message to a selection owner to tell the owner that it is becoming the preserved owner. (This can happen when the `SelectionManager` receives `msgSelSetOwnerPreserve`.) Receivers should not do anything in response to this message. (If for some reason receivers chose to handle this message, be careful about sending selection manager messages (such as `msgSelSetOwner`) as deadlock can occur.)

See Also

`msgSelPromote`

**msgSelPromote**

Informs the preserved owner that it is becoming the owner.

Takes nothing, returns STATUS.

```c
#define msgSelPromote MakeMsg(clsSelection, 25)
```

Comments

The `SelectionManager` sends this message to a preserved selection owner to tell the owner that it is becoming the normal selection owner. (This can happen when the `SelectionManager` receives `msgSelSetOwnerPreserve`.) Receivers should not do anything in response to this message. (If for some reason receivers chose to handle this message, be careful about sending selection manager messages (such as `msgSelSetOwner`) as deadlock can occur.)

See Also

`msgSelOwner`

---

## Embedded Window Messages

Most subclasses of `clsEmbeddedWin` should use these messages. See `embedwin.h` for information about how and why to use them.

The messages are defined here rather than in `embedwin.h` because they are abstract. Theoretically other classes can respond to these messages to implement behavior analogous to that of `embeddedWin` (although no other PenPoint system class does so).

**msgSelSelect**

Sets self to be the selection owner.

Takes nothing, returns STATUS.

```c
#define msgSelSelect MakeMsg(clsSelection, 19)
```

Comments

See the section "Embedded Window Selection Messages" for more information.

Send this message to an object to have that object make itself be the selection owner or the preserved selection owner.

Do not send this message to the `SelectionManager`.

See Also

`msgSelOwner.h`
**msgSelIsSelected**

Returns TRUE if self is current selection owner.

Takes nothing, returns BOOLEAN.

```c
#define msgSelIsSelected MakeMsg(clsSelection, 21)
```

**Comments**

See the section "Embedded Window Selection Messages" for more information.

Send this message to an object to inquire if it is the selection owner.

Do not send this message to theSelectionManager.

**Return Value**

- **true**: The object is the selection owner.
- **false**: The object is not the selection owner. (The object may be the preserved selection owner.)

**See Also**

embedwin.h

---

### Abstract Messages for Selection Move & Copy

**msgSelBeginCopy**

Initiate a copy operation.

Takes P_XY32, returns STATUS.

```c
#define msgSelBeginCopy MakeMsg(clsSelection, 23)
```

**Comments**

See the section "Move and Copy" for information about when this message is sent and how it should be handled.

pArgs will be null if this message is sent from a menu.

---

**msgSelBeginMove**

Initiates a move operation.

Takes P_XY32, returns STATUS.

```c
#define msgSelBeginMove MakeMsg(clsSelection, 22)
```

**Comments**

See the section "Move and Copy" for information about when this message is sent and how it should be handled.

pArgs will be null if this message is sent from a menu.

---

**msgSelCopySelection**

The receiver should copy the selection to self at (x, y).

Takes P_XY32, returns STATUS.

```c
#define msgSelCopySelection MsgNoError(MakeMsg(clsSelection, 16))
```

**Comments**

See the section "Move and Copy" for information about when this message is sent and how it should be handled.
msgSelMoveSelection
The receiver should move the selection to self at (x, y).
Takes P_XY32, returns STATUS.
#define msgSelMoveSelection MsgNoError(MakeMsg(clsSelection, 15))

Comments
See the section "Move and Copy" for information about when this message is sent and how it should be handled.

msgSelDelete
The selection owner should delete the selection.
Takes U32, returns STATUS.
#define msgSelDelete MakeMsg(clsSelection, 3)
#define SelDeleteReselect 0 // Display a selection after delete
#define SelDeleteNoSelect 1 // Don't display a selection after delete

Clients wishing to delete the selection send msgSelDelete to the selection owner. Selection owners should respond to this message by deleting the contents of the selection.
msgSelDelete is sent in two situations: (1) the user has hit the "Delete" menu item, or (2) an object has received msgSelMoveSelection, has copied the data (see xfer.h), and now wants to delete the original data.
See the section "Move and Copy" for information about how msgSelDelete is related to moving data.
pArgs must be one of SelDeleteReselect or SelDeleteNoSelect. This parameter is just a performance enhancement. The sender of msgSelDelete should pass SelDeleteNoSelect if it plans on taking the selection after the msgSelDelete, and SelDeleteReselect otherwise. The receiver of msgSelDelete can use pArgs as an optimization, but it is not strictly necessary since theSelectionManager will send a msgSelYield when the sender takes the selection. (The pArgs of msgSelDelete exist primarily for historical reasons. The simplest thing to do is for the sender to pass SelDeleteReselect and for the receiver to ignore pArgs.)

Abstract Messages For Linking Protocol

msgSelRememberSelection
The receiver should "remember" the selection and place the "remembrance" at (x, y).
Takes P_XY32, returns STATUS.
#define msgSelRememberSelection MsgNoError(MakeMsg(clsSelection, 17))

Comments
Most objects should not send or handle this message. It might be better defined as a clsEmbeddedWin message.
msgSelRememberSelection is sent to an object to ask it to "remember" the selection. The response to this message is highly object specific.
This message is not sent to the selection owner; it is sent to any object to ask it remember the selection.
An embeddedWin self sends this message in response to the "Create Reference Button" gesture (xgsDblCircle in xgesture.h). In response, an embeddedWin creates a goto button at the specified (x,y).

See Also
embedwin.h
SPELL.H

Spelling Checking

See Also
proof.h, pdict.h

#ifndef SPELL_INCLUDED
#define SPELL_INCLUDED

/DS0001 Low-level debug messages; LOTS of output
/DS0002 mid-level debug messages
/DS0004 high-level debugs - general information
/DS8000 disable dictionary

#ifndef GO_INCLUDED
#include <go.h>
#endif

Common Definitions

maxSpellList is the most bytes a list of spelling corrections can use. is the dictionary alphabet size

#define maxSpellList 128
#define maxSpellXlateChoices 30

Common typedefs

typedef struct SPELL_LIST {
    U16 count; // Number of strings in the list
    CHAR words[maxSpellList]; // List of concatenated strings
} SPELL_LIST, *P_SPELL_LIST;

typedef struct SPELL_XLATE {
    U16 index; // Offset within bank
    U8 bits; // Nibble and bank indicator
    U8 character; // Out: Character at that location
} SPELL_XLATE, *P_SPELL_XLATE;

typedef struct SPELL_DICT_LIST {
    P_CHAR pName; // name of dictionary (e.g. English)
    P_CHAR pPath; // path to dictionary (e.g. \boot\dicts\webf77k)
    U16 bankCount; // Number of 16K banks the lex is divided into
    P_UNKNOWN pLangHeader; // Pointer to language specific info
} SPELL_DICT_LIST, *P_SPELL_DICT_LIST;

Definitions of different types of word capitalization

Enum16(SPELL_CASE) {
    spellCommonCase, // all letters are in lower case
    spellProperCase, // The First Letter Of Each Word Is Capitalized
    spellUpperCase, // ALL LETTERS ARE CAPITALIZED
    spellSpecialCase, // tHere IS a StrANge Mix of cAPitALizATion
};

typedef struct SPELL_CASE_CONTEXT {
    SPELL_CASE minCase; // lowest case allowed for output dictionary words
    SPELL_CASE unkCase; // case for non-dictionary words
    BOOLEAN sentence; // do end-of-sentence processing
    BOOLEAN dictionary; // use the dictionary for capitalization info
    BOOLEAN allCapsWriter; // user writes all caps only
    BOOLEAN firstWord; // In/Out: This word is first in a sentence
} SPELL_CASE_CONTEXT, *P_SPELL_CASE_CONTEXT;
## Functions

### SpellDictSelect
Sets the active dictionary to the language specified.

Returns STATUS.

**Function Prototype**

```c
STATUS EXPORTED SpellDictSelect(
    S16 dictCode
);
```

*dictCode* is an index into spellDictList; -1 means deselect. Currently, only English can be selected, and its code is 0.

### SpellSetOptionsX
Turns the dictionary on or off.

Returns void.

```c
void EXPORTED SpellSetOptionsX(BOOLEAN mode);
```

Pass it true to turn the dictionary on, false to turn it off.

### SpellGetOptionsX
Returns current dictionary status.

Returns BOOLEAN.

```c
BOOLEAN EXPORTED SpellGetOptionsX(void);
```

True means spelling is on; false means it’s off.

### SpellCheck
Checks if a word is in the dictionary or not.

Returns BOOLEAN.

```c
BOOLEAN EXPORTED SpellCheck(P_CHAR pWord);
```

Argument may contain punctuation but should not contain spaces. This designed so higher-level software can parse a line of text into tokens and pass those tokens (with no further) to this routine.

### SpellCorrect
Finds all the corrections for a word and adds them to a SPELL_LIST structure.

Returns STATUS.

**Function Prototype**

```c
STATUS EXPORTED SpellCorrect(
    P_CHAR pWord,
    P_SPELL_LIST pSpellList, // Out: List to add the word to
    BOOLEAN phonetic        // Perform phonetic correction?
);
```

This also takes a space-delimited token, as described above, strips the punctuation, and puts it back on the correction candidates that the count field in the SPELL_LIST structure must be zero, unless you are deliberately adding to a list. This routine avoids adding duplicates to the if it already had some words in it.
**SpellCorrectWord**

Finds the first correction for a word. Returns 0 if none found, else 1.

Returns U16.

Function Prototype

```c
U16 EXPORTED SpellCorrectWord(
    P_CHAR pWord,  // Word to be corrected
    P_CHAR pCorrectWord  // Out: place to put the correction
);
```

The word is a space-delimited token, as described above. In this, "first" means "first in alphabetical order," this routine is suitable for most applications.

**SpellAddToDict**

Add a word to the Personal Dictionary.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED SpellAddToDict(
    P_CHAR pWord
);
```

Comments

The preferred way to add words to the current personal dictionary. As usual, it takes space-delimited tokens and strips off extraneous punctuation.

**SpellAddToAnyDict**

Add a word to any one of the personal dictionaries.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED SpellAddToAnyDict(
    OBJECT pDict,
    P_CHAR pWord
);
```

Comments

The preferred way to add words to a personal dictionary other than the current one. It takes a pdict object (clsPDict) that specifies the personal dictionary to add to, and space-delimited tokens. It strips off extraneous punctuation.

**SpellWordSetCase**

Convert all-upper-case input into a reasonable mix of upper and lower case using dictionary information and other lexical clues.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED SpellWordSetCase(
    P_CHAR pWord,
    P_SPELL_CASE_CONTEXT pSpellCaseContext
);
```

Call SpellWordSetCase the first time with pWord == pNull to the context structure. Then pass it the words to be (in order) with the same context structure each time. It each word in place. To modify the default behavior, change appropriate context parameters (see the definition of the CASE_CONTEXT structure).

Defaults:

- minCase: Spell
- unkCase: Spell
- sentence: true
- dictionary: true
- allCapsWriter: false
- firstWord: true
**SpellLineSetCase**

Convert all-upper-case input into a reasonable mix of upper and lower case using dictionary information and other lexical clues.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED SpellLineSetCase(
    P_CHAR pLine,
    P_SPELL_CASE_CONTEXT pSpellCaseContext
);
```

Identical to SpellWordSetCase, except it expects the input to be line of text, which it splits into tokens as required.

**Miscellaneous**

Address of the list of legal dictionaries

```c
extern const SPELL_DICT_LIST spellDictList[];
```
This file contains the API for the Spell Manager Class and the SpellManager.

clsSpellManager inherits from clsObject.

the SpellManager is a well-known instance of clsSpellManager.

See Also

spell.h, pdict.h

```c
#ifndef SPMGR_INCLUDED
#define SPMGR_INCLUDED
#endif

#ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif

#ifndef WIN_INCLUDED
#include <win.h>
#endif

#ifndef XLATE_INCLUDED
#include <xlate.h>
#endif

#ifndef GWIN_INCLUDED
#include <gwin.h>
#endif
```

Common typedefs

This structure is passed to the SpellManager when the user makes the gesture on a window.

```c
typedef struct SP_MGR_GESTURE {
    GWIN GESTURE gesture;
} SP_MGR_GESTURE, *P_SP_MGR_GESTURE;
```

Messages

Sent to Traversal Clients

**msgSpMgrCreateContext**

Piggybacked with msgTraverseCreate. *Ctx messages.

Takes VOID, returns STATUS.

```c
#define msgSpMgrCreateContext MakeMsg(clsSpellManager, 1)
```

Initiates a spelling traversal.

**msgSpMgrFindMisspelling**

Asks the recipient to find the next misspelled word (using SpellCheckO on successive space-delimited tokens).

Takes SP_MGR_DIALOG, returns STATUS.

```c
#define msgSpMgrFindMisspelling MakeMsg(clsSpellManager, 2)
```

Piggybacked with msgTraverseFind.
**msgSpMngrCorrectMisspelling**

Asks the recipient to correct the misspelled word he previously found in response to a msgSpMngrFindMisspelling message.

Takes SP_MGR_DIALOG, returns STATUS.

```c
#define msgSpMngrCorrectMisspelling MakeMsg(clsSpellManager, 3)
```

Piggybacked with msgTraverseApply. Correction is in the word field.

**msgSpMngrAcceptMisspelling**

Asks the recipient to accept the misspelled word he previously found in response to a msgSpMngrFindMisspelling message.

Takes SP_MGR_DIALOG, returns STATUS.

```c
#define msgSpMngrAcceptMisspelling MakeMsg(clsSpellManager, 5)
```

Piggybacked with msgTraverseApply. Dialog Struct is copied.

**Received From GWin**

**msgSpMngrGesture**

This causes theSpellManager to initiate a spell traversal from a gesture, as opposed to from a menu.

Takes P_SP_MGR_GESTURE, returns STATUS.

```c
#define msgSpMngrGesture MakeMsg(clsSpellManager, 4)
```

When a user makes the spelling gesture on an embedded window, the sends msgSpMngrGesture to theSpellManager with the_MGR_GESTURE structure filled in.

**Miscellaneous**

**Quick Help Tags**

- `-define SpMgrReplaceButtonTag MakeTag(clsSpellManager,1)`
- `-define SpMgrIgnoreButtonTag MakeTag(clsSpellManager,2)`
- `-define SpMgrCancelButtonTag MakeTag(clsSpellManager,3)`
- `-define SpMgrInsertionPadTag MakeTag(clsSpellManager,4)`
- `-define SpMgrTKTableTag MakeTag(clsSpellManager,5)`
- `-define SpMgrBackgroundTag MakeTag(clsSpellManager,6)`
- `-define SpMgrClearButtonDownTag MakeTag(clsSpellManager,7)`
- `-define SpMgrRememberButtonTag MakeTag(clsSpellManager,8)`
- `-define SpMgrTitleBarTag MakeTag(clsSpellManager,9)`
- `-define hlpSpMgrReplaceButton SpMgrReplaceButtonTag`
- `-define hlpSpMgrIgnoreButton SpMgrIgnoreButtonTag`
- `-define hlpSpMgrCancelButton SpMgrCancelButtonTag`
- `-define hlpSpMgrInsertionPad SpMgrInsertionPadTag`
- `-define hlpSpMgrTKTable SpMgrTKTableTag`
- `-define hlpSpMgrBackground SpMgrBackgroundTag`
- `-define hlpSpMgrClearButton SpMgrClearButtonTag`
- `-define hlpSpMgrRememberButton SpMgrRememberButtonTag`
- `-define hlpSpMgrTitleBar SpMgrTitleBarTag`

// Different help tags for when this is proof instead of spell
- `-define hlpProofInsertionPad MakeTag(clsSpellManager,10)`
- `-define hlpProofTKTable MakeTag(clsSpellManager,11)```
clsSR inherits from clsObject.

clsSR is the class of the SearchManager. It defines a protocol which clients can respond to implement Find and Replace. Clients of this protocol must respond to the "mark" protocol defined in mark.h.

### Debugging Flags

The Find and Replace mechanism uses the debug flag R10000.

```c
#ifndef SR_INCLUDED
#define SR_INCLUDED 1
#endif
#ifndef MARK_INCLUDED
#include <mark.h>
#endif
```

### Common #defines and typedefs

```c
#define srBufSize 80
typedef struct SR_FLAGS {
  BOOLEAN matchCase : 1, // case must match
  BOOLEAN matchWord : 1, // full word search
  BOOLEAN keepOldCase : 1, // replace with found case
  BOOLEAN findFromEdge : 1, // search from edge of doc
  BOOLEAN onBigCard : 1; // display big card
} SR_FLAGS;
typedef struct SR_METRICS {
  CHAR findText[srBufSize];
  CHAR replaceText[srBufSize];
  MARK_MSG_FLAGS markFlags;
  SR_FLAGS searchFlags;
} SR_METRICS, *P_SR_METRICS;
```

### Statuses

The current match cannot/may not be replaced.

```c
#define stsSRCannotReplace MakeStatus(clsSR, 1)
```

### Messages Sent to Clients via clsMark

#### msgSRNextChars

Asks the client to move the token to the next group of characters.

Takes P_SR_NEXT_CHARS, returns STATUS.

```c
#define msgSRNextChars MakeMsg(clsSR, 1)
```

**Arguments**

```c
typedef struct SR_NEXT_CHARS {
  MARK_MSG_HEADER header;
  U32 maxLen; // In: maximum size the group can be
  U32 len; // Out: the size of the group
  BOOLEAN blockStart; // Out: the group starts a block
  BOOLEAN blockEnd; // Out: the group ends a block
} SR_NEXT_CHARS, *P_SR_NEXT_CHARS;
```
Important: your handler must have the following as its first statement. Replace "clsYourClassHere" with the uid of your class. See mark.h.

MarkHandlerForClass(clsYourClassHere);

This group may be up to maxLen characters in size. The client sets the len parameter to the actual size of the group, and if the group is the start and/or end of a block of character, sets the respective flags. A block is defined as a logically contiguous group of characters that can be searched.

Any non-text element usually delimits the end of a block. If the element is an embedded window that should be searched, the token should be set to point to the embedded window and staMarkEnterChild (see mark.h) should be returned. If the element is not a child, then it should be simply skipped and the token moved to the next group of characters following it.

Example: If the following text is in the client’s data, and msgSRNextChars is received with a maxLen of 5, the token would should refer to the blocks 1 through 4 in succession. blockStart should be true for blocks 1 and 3 and blockEnd should be true for blocks 2 and 4. In this way, "SEN" and "MANTLE" can be found, but "GERMAN" which spans some non-text object won’t be mistakenly found.

```
M E S S E N G E R (non-text-thing) M A N T L E

+----1----+---2---+----------------+----3----+4+
```

**msgSRGetChars**

The component passes back the characters from the location identified by the token.

Takes P_SR_GET_CHARS, returns STATUS.

```c
#define msgSRGetChars MakeMsg(clsSR, 2)
```

**Arguments**

typedef struct SR_GET_CHARS {
    MARK_MSG_HEADER header;
    U32 first;  // In: character to start with
    U32 len;    // In: the number of characters to return
    U32 buflen; // In: length of the buffer
    P_CHAR pBuf; // In: pointer to the buffer to fill
} SR_GET_CHARS, *P_SR_GET_CHARS;

Important: your handler must have the following as its first statement. Replace "clsYourClassHere" with the uid of your class. See mark.h.

MarkHandlerForClass(clsYourClassHere);

pArgs->first is token-relative and pArgs->len is the number of characters to return. Thus (0,2) requests the first two characters, (1,1) requests the second character, and (3,0) requests no characters.

The string returned must be null-terminated. Note that if len is less than buflen then this is always possible without truncation. Otherwise, the number of characters returned should be one less than buflen and they should still be null terminated.

**msgSRReplaceChars**

Ask the component to replace some of the characters at the location identified by the token.

Takes P_SR_REPLACE_CHARS, returns STATUS.

```c
#define msgSRReplaceChars MakeMsg(clsSR, 3)
```

**Arguments**

typedef struct SR_REPLACE_CHARS {
    MARK_MSG_HEADER header;
    S32 first;    // In: replacement starts here
    U32 len;      // In: ...and is this long
    U32 buflen;   // In: repl. size in characters
    PCHAR pBuf;   // In: the buffer of the characters
} SR_REPLACE_CHARS, *P_SR_REPLACE_CHARS;
Important: your handler must have the following as its first statement. Replace "clsYourClassHere" with the uid of your class. See mark.h.

MarkHandlerForClass(clsYourClassHere);

pArgs->first is token-relative, and pArgs->len is the number of characters to replace. Thus (0,2) replaces the first two characters, (1,1) replaces the second character, and (3,0) replaces no characters starting between the third and fourth (thus effecting an insertion).

pArgs->first may be negative, indicating replacement of text BEFORE the current token (or large indicating AFTER). However, in no case will pArgs->first go beyond the boundaries indicated by the blockStart and blockEnd flags from previous calls to msgSRNextChars.

This message should only affect the token insofar as the replacement makes changes to the data the token refers to. For example: if the token refers to the three characters "cat" and the replace messages changes the substring "c" (0,1) into "womb", then the token should now refer to the six characters "wombat".

msgSRPositionChars
Asks the component to reposition the token to some of the characters in the current group.

Takes P_S_R_POSITION_CHARS, returns STATUS.

#define msgSRPositionChars MakeMsg(clsSR, 4)

typedef struct SR_POSITION_CHARS {
    MARK_MSG_HEADER header;
    S32 first;   // In: new position starts here
    U32 len;    // In: ... and is this long
} SR_POSITION_CHARS, *P_SR_POSITION_CHARS;

Important: your handler must have the following as its first statement. Replace "clsYourClassHere" with the uid of your class. See mark.h.

MarkHandlerForClass(clsYourClassHere);

pArgs->first is token-relative, and pArgs->len is the number of characters to reposition to. Thus (0,2) positions to the first two characters, (1,1) positions to the second character, and (3,0) positions to between the third and fourth characters.

pArgs->first may be negative indicating positioning BEFORE the current token (or large indicating AFTER). However, in no case will pArgs->first go beyond the boundaries indicated by the blockStart and blockEnd flags from previous calls to msgSRNextChars.

Messages to theSearchManager
These messages are sent to theSearchManager by PenPoint's standard UI elements. Typical clients do not send them.

msgSRInvokeSearch
Starts a Find & Replace option sheet.

Takes P_S_R_INVOKE_SEARCH, returns STATUS.

#define msgSRInvokeSearch MakeMsg(clsSR, 10)
typedef struct SR_INVOKE_SEARCH {
    OBJECT target;  // nil if fromGesture or fromSelection
    BOOLEAN fromSelection : 1,  // start from the selection
    fromGesture : 1,  // start from the gesture given
    doFind : 1,  // do an initial find
    findBackward : 1,  // direction for initial find
    noUI : 1,  // don't open option sheet
    useWord : 1,  // use the word at the gesture or selection
    useFlags : 1;  // use the flags in metrics
    U16 reserved;
    GWIN_GESTURE gesture;  // the gesture if fromGesture
    SR_METRICS metrics;  // optional initial text and flags
    U32 reserved2;
} SR_INVOKE_SEARCH, *P_SR_INVOKE_SEARCH;

The target of the search is the target argument. However if fromSelection is true then it is the selection; or if fromGesture is true then it is from the gesture.

The user's last saved metrics are always used except that

- metrics.findText is used if it is not the empty string
- metrics.replaceText is used if it is not the empty string
- metrics.markFlags & metrics.searchFlags are used if pArgs->useFlags is true

If doFind is true, then an initial find is executed.

If noUI is true, then the option sheet isn't created. This is only useful in conjunction with doFind (otherwise, nothing has happened!), the result being a "find next" operation.

If useWord is true, then the find text will be fetched from the target with msgSRGetChars.

msgSRRememberMetrics

Asks theSearchManager to remember the current settings of a Find & Replace option sheet

Takes P_SR_METRICS, returns STATUS.

#define msgSRRememberMetrics MakeMsg(clsSR, 12)

typedef struct SR_METRICS {
    CHAR findText[srBufSize];
    CHAR replaceText[srBufSize];
    MARK_MSG_FLAGS markFlags;
    SR_FLAGS searchFlags;
} SR_METRICS, *P_SR_METRICS;

As a result, when theSearchManager option sheet next appears it will have these settings.
This file contains the API definition for clsString.

clsString inherits from clsByteBuf.

clsString provides a facility to store null-terminated ASCII byte strings. Each object of clsString stores a single string. This class provides convenient object filing of the string data. Storage for each object's string is allocated out of the creator's shared process heap using OSHepBlockAlloc.

Clients who want to store uninterpreted byte arrays should use clsByteBuf (see bytebuf.h).

clsString and clsByteBuf do not share messages. clsByteBuf messages cannot be sent to a clsString object.

```c
#ifndef STROBJ_INCLUDED
#define STROBJ_INCLUDED
#include <go.h>
#include <clsmgr.h>
typedef OBJECT STROBJECT, *P_STROBJECT;
```

### Class Messages

**msgNew**

Creates a new string object.

Takes P_STROBJ_NEW_ONLY, returns STATUS. Category: class message.

**Arguments**

typedef struct STROBJ_NEW_ONLY {
  P_CHAR pString;
} STROBJ_NEW_ONLY, *P_STROBJ_NEW_ONLY;

#define clsObjNewFields 
  objectNewFields 
  STROBJ_NEW_ONLY  strobj;

typedef struct STROBJ_NEW {
  clsObjNewFields
  STROBJ_NEW, *P_STROBJ_NEW;
}

**Comments**

This message allocates shared heap storage for the specified string and copies the client string data into it.

**msgNewDefaults**

Initializes the STROBJ_NEW structure to default values.

Takes P_STROBJ_NEW, returns STATUS. Category: class message.

**Message**

typedef struct STROBJ_NEW {
  clsObjNewFields
} STROBJ_NEW, *P_STROBJ_NEW;

**Arguments**

```c
Sets
pNew->strobj.pString = pNull;
```
Object Messages

msgStrObjGetStr
Passes back the object's string.
Takes PP_CHAR, returns STATUS.

#define msgStrObjGetStr MakeMsg(clsString, 1)

Comments
The pointer passed back references the object's global storage. Clients must not modify or free this storage.

msgStrObjSetStr
Copies the specified string data into the object's string buffer.
Takes P_CHAR, returns STATUS.

#define msgStrObjSetStr MakeMsg(clsString, 2)

Comments
Previously retrieved string pointers will be invalid after this operation. Clients must call msgStrObjGetStr to retrieve a pointer to the valid object buffer.

Observer Messages

msgStrObjChanged
Sent to observers when the string object data changes.
Takes OBJECT, returns nothing. Category: observer notification.

#define msgStrObjChanged MakeMsg(clsString, 3)

Comments
The message argument is the UID of the clsString object that changed.
This file contains the API definition for clsTable.

clsTable inherits from clsObject.

clsTable provides a general-purpose table mechanism with random and sequential access. The table allows clients to create, destroy, modify, and access the table and its data using a row and column metaphor. Data for the table is stored in a table file, whose lifetime can be independent to that of the table object.

Tables are two dimensional arrays consisting of a fixed number of columns and a variable number of rows. Each column can contain data of a single data type such as a U32, a variable length string, a fixed sized byte field, date and time, etc.

The number of and types of these columns are defined when the table is created. Once that table has been created, these parameters cannot be changed.

Clients access rows in the table using a TBL_ROW_POS data structure. The value for this row position is returned to the client when a row is added to the table. All messages for manipulating data in the table require this value to specify an individual row.

Clients address columns using their position in the TBL_COL_DESC array which the client provides in the TBL_CREATE data structure during msgNew.

The table is an observable object and clients choosing to be observers will receive notification when data in the table changes or a row has been added to or removed from the table.

```
 ifndef TS_INCLUDED
 define TS_INCLUDED
 include <clsmgr.h>
 include <fs.h>
 include <resfile.h>
```

## Status Codes

Status values return by messages to clsTable.

```
#define stTBLRefCountNotZero MakeStatus( clsTable, 1 )
#define stTBLColNameNotFound MakeStatus( clsTable, 2 )
#define stTBLStrBufTooSmall MakeStatus( clsTable, 3 )
#define stTBLBadNewFlags MakeStatus( clsTable, 4 )
#define stTBLEndOfFileTable MakeStatus( clsTable, 5 )
#define stTBLInvalidSortColValue MakeStatus( clsTable, 7 )
#define stTBLCorruptedIndex MakeStatus( clsTable, 8 )
#define stTBLColNotIndexed MakeStatus( clsTable, 9 )
#define stTBLContainsIndexedCols MakeStatus( clsTable, 10 )
```
Common macros and typedefs

- Class Declaration
  #define clsTable MakeWKN(2003,1,wknGlobal)

- Object Declarations
  typedef OBJECT TABLE;
  typedef OBJECT TBLOBJ;
  typedef TBLOBJ *P_TBLOBJ;

- Table Parameter Definitions
  #define TBL_MAXCOLNAMELEN nameBufLength
  #define TBL_MAXTBLNAMELEN nameBufLength
  #define TBL_MAXROWCOUNT 0x2000 // 8192 entries

- Table Row Definitions
  typedef RES_ID TBL_ROW_POS, *P_TBL_ROW_POS;
  typedef U16 TBL_ROW_NUM, *P_TBL_ROW_NUM;
  typedef U16 TBL_ROW_COUNT, *P_TBL_ROW_COUNT;
  typedef U16 TBL_ROW_LENGTH, *P_TBL_ROW_LENGTH;
  typedef S32 TBL_ROW_OFFSET, *P_TBL_ROW_OFFSET;
  typedef S16 TBL_REF_COUNT, *P_TBL_REF_COUNT;

- Table Data Type Definitions
  typedef P_U8 P_ROW_BUFFER, *PP_ROW_BUFFER;
  typedef P_UNKNOWN P_TBL_COL_DATA_HOLDER;

- Column Index Declarations
  typedef U16 TBL_COL_INX_TYPE, *P_TBL_COL_INX_TYPE;
  typedef U16 TBL_COL_COUNT, *P_TBL_COL_COUNT;
  typedef U16 TBL_COL_LENGTH, *P_TBL_COL_LENGTH;
  typedef U32 TBL_COL_OFFSET, *P_TBL_COL_OFFSET;

- Column Descriptor Definitions
  typedef enum TBL_TYPES {
    tsChar = 0, // fixed length byte array of case sensitive chars
    tsCaseChar = 1, // fixed length byte array of case insensitive chars
    tsU16 = 2, // unsigned 16 bit integer
    tsU32 = 3, // unsigned 32 bit integer
    tsFP = 4, // double precision floating point
    tsDate = 5, // date field (system compressed time format)
    tsString = 6, // null-terminated variable length ascii string (case sensitive)
    tsCaseString = 7, // same as tsString but is case insensitive
    tsByteArray = 8, // variable length byte array, contained in unsigned chars
    tsUUID = 9, // UUID struct.
    tsLastType = tsUUID
  } TBL_TYPES;
  typedef struct TBL_COL_DESC {
    CHAR name[TBL_MAXCOLNAMELEN]; // Column name
    TBL_TYPES type; // Column type
    TBL_COL_LENGTH length; // Column data length
    TBL_COL_INX_TYPE repeatFactor; // # of times to repeat the column
    TBL_COL_OFFSET offset; // Column offset in the row
    BOOLEAN sorted; // Is the column sorted?
  } TBL_COL_DESC, *P_TBL_COL_DESC;

- Variable Length Data Buffer Definition
  typedef struct TBL_STRING {
    U16 strLen; // In/Out: length of string or byte array column data
    U16 strMax; // In: length of string or byte array buffer
    P_CHAR pStr; // In: pointer to client buffer.
  } TBL_STRING, *P_TBL_STRING;
### Class Messages

#### msgNew

Creates a new table object.

Takes P_TBL_NEW, returns STATUS. Category: class message.

**Arguments**

```c
typedef enum TBL_FREE_BEHAVE {
  tsFreeNoDeleteFile = 0,  // Free only the object, not the file
  tsFreeDeleteFile   = flag0,  // Destroy the file when freed
  tsFreeWhenNoClients = flag1,  // Free when # clients accessing is 0
  tsFreeNoObservers  = flag2,  // Free when # of observers is 0
  tsFreeNoCompact    = flag3,  // Don't compact the table when freed
  tsFreeDefault      = tsFreeNoDeleteFile
} TBL_FREE_BEHAVE, *P_TBL_FREE_BEHAVE;
```

```c
typedef enum TBL_EXIST {
  tsExistOpen      = 0,  // Open an existing table
  tsExistGenError  = 1,  // Return error if table exists
  tsExistGenUnique = 2,  // Create table with a unique name
  tsNoExistCreate  = MakeU16(0, 0),  // Create a new table
  tsNoExistGenError = MakeU16(0, 1),  // Return error if no table exists
  tsExistDefault   = tsExistOpen | tsNoExistCreate
} TBL_EXIST, *P_TBL_EXIST;
```

```c
typedef struct TBL_CREATE {
  TBL_COL_COUNT colCount;
  P_TBL_COL_DESC colDescAry;
} TBL_CREATE, *P_TBL_CREATE;
```

```c
typedef struct TBL_NEW_ONLY {
  CHAR name[TBL_MAXTBLNAMELEN];  // Table name
  FS_LOCATOR locator;  // Table file
  TBL_EXIST exist;  // Table exist behavior
  TBL_CREATE create;  // Column specifications
  TBL_FREE_BEHAVE freeBehavior;  // Table free behavior
  BOOLEAN createSemaphore;  // Provide semaphore?
} TBL_NEW_ONLY, *P_TBL_NEW_ONLY;
```

#define tableNewFields table;
#define objectNewFields table;

```c
typedef struct TBL_NEW {
  tableNewFields
} TBL_NEW, *P_TBL_NEW;
```

#### Comments

This message creates a new table file or opens an existing file.

The table name is an optional field. The locator and colDescAry fields must be valid and colCount must be non zero or this message returns stsBadParam.

**Return Value**

- **stsTBLBadNewFlags**: TBL_EXIST flags were invalid.
- **stsBadParam**: locator or colDescAry fields are invalid. colCount is 0.

#### msgNewDefaults

Initializes the TBL_NEW structure to default values.

Takes P_TBL_NEW, returns STATUS. Category: class message.

**Message**

```c
typedef struct TBL_NEW {
  tableNewFields
} TBL_NEW, *P_TBL_NEW;
```
Zeroes out pNew->table and sets:

\[ pNew->table.name[0] = '\0'; \]
\[ pNew->table.locator.uid = objNull; \]
\[ pNew->table.locator.pPath = pNull; \]
\[ pNew->table.exist = tsExistDefault; \]
\[ pNew->table.create.colCount = 0; \]
\[ pNew->table.create.colDescAry = pNull; \]
\[ pNew->table.freeBehavior = tsFreeDefault; \]
\[ pNew->table.createSemaphore = false; \]

**msgDestroy**

Destroys an existing table object.

Takes OBJ_KEY, returns STATUS. Category: class message.

Comments

This message destroys the table object and frees the table files if the object was created with the tsFreeDeleteFile flag specified.

The table file will not be destroyed regardless of whether tsFreeDeleteFile was specified if there are still accessors to the table. Only the object will be freed.

Return Value

sts TBLRefCountNotZero The number of accessors of the table is not zero. The table file will not be destroyed.

**Object Messages**

**Table Row Addition and Deletion Messages**

**msgTBLAddRow**

Adds a row/record with no data to the table server object.

Takes P_TBL_ROW_POS, returns STATUS.

#define msgTBLAddRow MakeMsg(clsTable, 1)

Comments

The row position (TBL_ROW_POS) for the new row is passed back. The row position is the key to access data in the row or to delete the row.

**msgTBLDeleteRow**

Deletes the specified row.

Takes P_TBL_ROW_POS, returns STATUS.

#define msgTBLDeleteRow MakeMsg(clsTable, 5)

Comments

Rows are deleted from the table at the completion of this call. The row’s TBL_ROW_POS is no longer valid after the row has been deleted.

Return Value

sts TBLRowNotFound TBL_ROW_POS value was not found in the table.
Table Data Messages

msgTBLColGetData
Attempts to get the data for the specified row and column.
Takes P_TBL_COL_GET_SET_DATA, returns STATUS.
#define msgTBLColGetData MakeMsg(clsTable, 13)

Arguments
typedef struct TBL_COL_GET_SET_DATA {
    TBL_ROW_POS tblRowPos; // In: Table row position
    TBL_COL_INX_TYPE colNumber; // In: Column number
    P_TBL_COL_DATA_HOLDER tblColData; // Out: Column data
} TBL_COL_GET_SET_DATA, *P_TBL_COL_GET_SET_DATA;

Comments
tblColData is of type P_TBL_STRING if the column type is tsString, tsCaseString, or tsByteArray.
The client is responsible for allocating storage for the tblStr.pStr buffer. If the buffer is too small to accommodate the requested data, the table will return stsTBLStrBuffTooSmall and pass back the truncated data and the actual length of the data in tblStr.strLen.

Return Value
stsTBLStrBuffTooSmall Returned if column type is tsString, tsCaseString or tsByteArray and tblStr.strMax is less than the actual data length. The data is truncated and the length is returned in tblStr.strLen.

msgTBLColSetData
Attempts to set the data for the specified row and column.
Takes P_TBL_COL_GET_SET_DATA, returns STATUS.
#define msgTBLColSetData MakeMsg(clsTable, 14)

Arguments
typedef struct TBL_COL_GET_SET_DATA {
    TBL_ROW_POS tblRowPos; // In: Table row position
    TBL_COL_INX_TYPE colNumber; // In: Column number
    P_TBL_COL_DATA_HOLDER tblColData; // Out: Column data
} TBL_COL_GET_SET_DATA, *P_TBL_COL_GET_SET_DATA;

Comments
tblColData is of type P_TBL_STRING if the column type is tsString, tsCaseString, or tsByteArray.
Clients are responsible for setting the strLen field of the TBL_STRING argument for all column types.

Return Value
stsTBLEndOfFile TBL_ROW_POS value was not found in the table.

msgTBLRowGetData
Attempts to get the contents of an entire row.
Takes P_TBL_GET_SET_ROW, returns STATUS.
#define msgTBLRowGetData MakeMsg(clsTable, 15)

Arguments
typedef struct TBL_GET_SET_ROW {
    TBL_ROW_POS tblRowPos; // In: Which row
    P_UNKNOWN pRowData; // Out: Row data
} TBL_GET_SET_ROW, *P_TBL_GET_SET_ROW;

Comments
Not valid for tables containing variable length columns.
The client is responsible for providing storage for the pRowData buffer. The length of a table row can be obtained using msgTBLGetRowLength.
Return Value
stsTBLEndOfTable TBL_ROW_POS value was not found in the table.
stsTBLContainsIndexedCols Table contains variable length columns.

See Also
msgTBLGetRowLength

msgTBLRowSetData
Sets the contents of an entire row.
Takes P_TBL_GET_SET_ROW, returns STATUS.

#define msgTBLRowSetData MakeMsg(clsTable, 16)

typedef struct TBL_GET_SET_ROW {
    TBL_ROW_POS tblRowPos; // In: Which row
    P_UNKNOWN pRowData; // Out: Row data
} TBL_GET_SET_ROW, *P_TBL_GET_SET_ROW;

Comments
Not valid for tables containing variable length columns.

Return Value
stsTBLEndOfTable TBL_ROW_POS value was not found in the table.
stsTBLContainsIndexedCols Table contains variable length columns.

See Also
msgTBLGetRowLength

Table Information Messages

msgTBLGetInfo
Gets the table header information.
Takes P_TBL_HEADER, returns STATUS.

#define msgTBLGetInfo MakeMsg(clsTable, 10)

typedef struct TBL_HEADER {
    TBL_COL_COUNT colCount; // number of columns in table
    CHAR name[TBL_MAXTABLENAMELEN]; // non-file table reference
    TBL_ROW_COUNT nRows; // how many rows in table
    TBL_ROW_LENGTH rowLength; // row buffer length
    TBL_ROW_POS firstRow; // position of first row in table
    TBL_ROW_POS currentRow; // position of current row in table
    TBL_ROW_POS lastRow; // position of last row in table
    TBL_REF_COUNT refCount; // number of active clients.
} TBL_HEADER, *P_TBL_HEADER, **PP_TBL_HEADER;

See Also
msgTBLGetColCount,

msgTBLGetColCount
Gets the number of columns in the table.
Takes P_TBL_COL_COUNT, returns STATUS.

#define msgTBLGetColCount MakeMsg(clsTable, 7)

msgTBLGetColDesc
Passes back the column description for the specified column.
Takes P_TBL_GET_COL_DESC, returns STATUS.

#define msgTBLGetColDesc MakeMsg(clsTable, 2)
typedef struct TBL_GET_COL_DESC {
    TBL_COL_INX_TYPE colInx; // In: column number
    TBL_COL_DESC colDesc; // Out: column description
} TBL_GET_COL_DESC, *P_TBL_GET_COL_DESC;

msgTBLGetRowCount

Gets the current number of rows in the table.
Takes P_TBL_ROW_COUNT, returns STATUS.
#define msgTBLGetRowCount MakeMsg(clsTable, 6)

msgTBLGetRowLength

Gets the length (in bytes) of the specified row.
Takes P_TBL_ROW_LENGTH, returns STATUS.
#define msgTBLGetRowLength MakeMsg(clsTable, 8)

Comments
The row length indicates the total width of all columns for each row in the table. This information is useful when getting and setting row data.

See Also
msgTBLRowGetData

msgTBLGetState

Gets the current state of a specified row.
Takes P_TBL_GET_STATE, returns STATUS.
#define msgTBLGetState MakeMsg(clsTable, 11)

Arguments
typedef enum TBL_STATE {
    tsBegin = 0, // rowPos is the first row
    tsEnd = 1, // rowPos is the last row
    tsPosition = 2 // rowPos is not first or last
} TBL_STATE, *P_TBL_STATE;

typedef struct TBL_GET_STATE {
    TBL_STATE tblState; // Out: State of the specified row
    TBL_ROW_POS tblRowPos; // In: Row position of the specified row.
} TBL_GET_STATE, *P_TBL_GET_STATE;

Comments
The state of a row in the table indicates its general positioning within the table.

Return Value
stsTBLEndOfTable TBL_ROW_POS value was not found in the table.

Table Access Messages

msgTBLBeginAccess

Initiates table access by a client on this table.
Takes P_TBL_BEGIN_ACCESS, returns STATUS.
#define msgTBLBeginAccess MakeMsg(clsTable, 17)

Arguments
typedef struct TBL_BEGIN_ACCESS {
    OBJECT sender; // In: sender's id IFF wants to be observer
    TBL_ROW_LENGTH rowLength; // Out: Length of the first row
} TBL_BEGIN_ACCESS, *P_TBL_BEGIN_ACCESS;

Comments
Passes back the row length of the first row. Adds the sender to the table's observer list.
**msgTBLEndAccess**

Ends client access to the table.

Takes P_TBL_END_ACCESS, returns STATUS.

```c
#define msgTBLEndAccess MakeMsg(clsTable, 18)
```

**Arguments**

typedef struct TBL_END_ACCESS {
  OBJECT sender; // In: Sender's uid
} TBL_END_ACCESS, *P_TBL_END_ACCESS;

**Comments**

Removes sender from the observer list.

**msgTBLSemaClear**

Releases the table's semaphore.

Takes nothing, returns STATUS.

```c
#define msgTBLSemaClear MakeMsg(clsTable, 23)
```

**Comments**

The next client currently waiting on the table semaphore will unblock when this message completes.

**msgTBLSemaRequest**

Requests access to the table's semaphore.

Takes nothing, returns STATUS.

```c
#define msgTBLSemaRequest MakeMsg(clsTable, 22)
```

**Comments**

Waits on the table semaphore if another client already has access. Provides exclusive access of the table semaphore to the sender when it returns.

Semaphore access has no timeout.

---

### Table Search Messages

**msgTBLFindFirst**

Finds the first record that meets the search specification.

Takes P_TBL_FIND_ROW, returns STATUS.

```c
#define msgTBLFindFirst MakeMsg(clsTable, 3)
```

**Arguments**

typedef enum TBL_BOOL_OP {
  tsEql = 0, // Match if operands are equal
  tsEqual = 1, // Match if operands are equal
  tsLess = 2, // Match if opnd1 < opnd2
  tsGreater = 3, // Match if opnd1 > opnd2
  tsGreaterEqual = 4, // Match if opnd1 >= opnd2
  tsLessEqual = 5, // Match if opnd1 <= opnd2
  tsNotEqual = 6, // Match if the operands do not match
  tsSubstring = 7, // Match if opnd1 is an exact substring of opnd2
  tsStartsWith = 8, // Match if opnd1 starts with opnd2
  tsAlwaysTrue = 9 // Match the first (or next) row
} TBL_BOOL_OP, *P_TBL_BOOL_OP;

typedef struct TBL_SEARCH_SPEC {
  TBL_COL_INX_TYPE colOperand; // In: Which column
  TBL_BOOL_OP relOp; // In: Operation
  P_TBL_COL_DATA_HOLDER pConstOperand; // In: Value to search against
} TBL_SEARCH_SPEC, *P_TBL_SEARCH_SPEC;

**Table Search Messages**

- **msgTBLFindFirst**
- **msgTBLSemaClear**
- **msgTBLSemaRequest**
- **msgTBLFindFirst**
- **msgTBLSemaClear**
- **msgTBLSemaRequest**
typedef struct TBL_FIND_ROW {
    TBL_ROW_POS rowPos;    // In:Out - current table position
    TBL_ROW_NUM rowNum;    // Out: indexed column row number
    TBL_SEARCH_SPEC srchSpec;  // In: search query
    TBL_COL_INX_TYPE sortCol;    // In: which column sort to use (if any)
    P_ROW_BUFFER pRowBuffer;    // In: pointer to client's buffer space
} TBL_FIND_ROW, *P_TBL_FIND_ROW;

Comments
Passes back the TBL_ROW_POS and TBL_ROW_NUM of the row.

srchSpec.pConstOperand is of type P_TBL_STRING if the column type is tsString, tsCaseString, or tsByteArray. The length of the string/array used in the search is decal red in the strLen field of the TBL_STRING struct. Clients are responsible for setting this field to the appropriate length for columns of type tsString, tsCaseString, and tsByteArray.

srchSpec.pConstOperand is ignored if srchSpec.relOp is tsAlwaysTrue.

Currently, tsSubstring searches are always case sensitive regardless of the column type.

Return Value
stsTBLEndOfTable  No data was found matching the search spec.
stsTBLInvalidSortColValue  sortCol is not a valid column value.

msgTBLFindNext
Find the next record following the specified TBL_ROW_POS that meets the search specification.

Takes P_TBL_FIND_ROW, returns STATUS.

#define msgTBLFindNext       MakeMsg(clsTable, 4)

typedef struct TBL_FIND_ROW {
    TBL_ROW_POS rowPos;    // In:Out - current table position
    TBL_ROW_NUM rowNum;    // Out: indexed column row number
    TBL_SEARCH_SPEC srchSpec;  // In: search query
    TBL_COL_INX_TYPE sortCol;    // In: which column sort to use (if any)
    P_ROW_BUFFER pRowBuffer;    // In: pointer to client's buffer space
} TBL_FIND_ROW, *P_TBL_FIND_ROW;

Comments
Passes back the TBL_ROW_POS and TBL_ROW_NUM of the row.

srchSpec.pConstOperand is of type P_TBL_STRING if the column type is tsString, tsCaseString, or tsByteArray. The length of the string/array used in the search is decal red in the strLen field of the TBL_STRING struct. Clients are responsible for setting this field to the appropriate length for columns of type tsString, tsCaseString, and tsByteArray.

srchSpec.pConstOperand is ignored if srchSpec.relOp is tsAlwaysTrue.

If srchSpec.colOperand is an unsorted column, then the order of the rows searched is random.

Return Value
stsTBLEndOfTable  No data was found matching the search spec, or rowPos is was not found in the table.
stsTBLInvalidSortColValue  sortCol is not a valid column value.

Table Utility Messages

msgTBLFindColNum
Passes back the column number for the specified column name.

Takes P_TBL_COL_NUM_FIND, returns STATUS.

#define msgTBLFindColNum       MakeMsg(clsTable, 12)
typedef struct TBL_COL_NUM_FIND {
    char name; // In: Column name
    TBL_COL_INX_TYPE number; // Out: Column number
} TBL_COL_NUM_FIND, *P_TBL_COL_NUM_FIND;

stsTBLColNameNotFound A column with the specified name does not exist.

msgTBLCompact
Compacts the table without closing it.
Takes nothing, returns STATUS.
#define msgTBLCompact MakeMsg(clsTable, 24)

This message allows clients to compact a table on demand. Compaction frees up any storage associated with previously deleted rows and compacts the table to its minimum file size. Ordinarily, a table is compacted automatically when the last client accessing the table closes it unless specifically prevented by specifying tsFreeNoCompact during msgNew.

msgTBLRowNumToRowPos
Converts a TBL_ROW_NUM to its corresponding TBL_ROW_POS for the specified column.
Takes P_TBL_CONVERT_ROW_NUM, returns STATUS.
#define msgTBLRowNumToRowPos MakeMsg(clsTable, 28)

typedef struct TBL_CONVERT_ROW_NUM {
    TBL_ROW_POS rowPos; // Out: - Table row pos.
    TBL_ROW_NUM rowNum; // In: - Index row number.
    TBL_COL_INX_TYPE colNum; // In: - Indexed (sorted) column number.
} TBL_CONVERT_ROW_NUM, *P_TBL_CONVERT_ROW_NUM;

This message is defined only for sorted columns. Unsorted columns do not have a defined order.

stsTBLEndOfTable rowNum is larger than the number of rows in the table.
stsTBLColNotIndexed The specified column is not sorted.

Observer Messages

msgTBLRowAdded
Sent to observers indicating that a row has been added.
Takes P_TBL_ROW_POS, returns STATUS. Category: observer notification.
#define msgTBLRowAdded MakeMsg(clsTable, 19)

A pointer to the newly added TBL_ROW_POS is sent as an argument.

msgTBLRowDeleted
Sent to observers indicating that a row has been deleted.
Takes nothing, returns STATUS. Category: observer notification.
#define msgTBLRowDeleted MakeMsg(clsTable, 20)
**msgTBLRowChanged**

Sent to observers indicating that row data has been changed.

Takes P_TBL_ROW_POS, returns STATUS. Category: observer notification.

```c
#define msgTBLRowChanged MakeMsg(clsTable, 21)
```

**Comments**

A pointer to the changed TBL_ROW_POS is sent as an argument.
This file contains the API definition for the UndoManager. The UndoManager is the wknProcessGlobal instance of clsUndo.

clsUndo inherits from clsList.

The functions described in this file are contained in MISC.LIB.

Introduction

The UndoManager provides a centralized facility for managing undo information. The UndoManager supports undo of user interface actions.

An undoable operation, or "undo transaction," is a collection of "undo items." Typically an undoable operation is a small UI action (e.g. deleting some text).

When the user issues an "Undo" command the most recent undo transaction will be undone. A typical scenario goes something like this:

- In response to some user interface action, a message handler begins an undo transaction with msgUndoBegin and then sends messages which manipulate the application's data.
- As the data manipulation routines do their work, they add undo items to the undo transaction via msgUndoAddItem.
- When the user interface handler regains control, the transaction is closed with msgUndoEnd.
- At some later date, the transaction might be undone. The UndoManager undoes a transaction by sending msgUndoltem to each item in the transaction (in the reverse order in which they were added).
- If the transaction is not undone, but instead falls off the end of the undo transaction queue, then the transaction is freed. (A transaction is also freed if the application is terminated.) The UndoManager frees a transaction by sending msgUndoFreeItemData to each item in the transaction. (But see the comments near the typedef UNDO_ITEM for some circumstances under which the UndoManager doesn't send msgUndoFreeItemData but instead frees the item itself.)

Common Messages

Typical application code will send the following messages to the UndoManager:

- msgUndoBegin
- msgUndoEnd
- msgUndoAddItem

Typical application code will receive the following messages from the UndoManager:

- msgUndoltem
- msgUndoFreeItemData

See the individual descriptions of each of these messages for more information.
Debugging Flags

Undo's debugging flag set is 'U.' Defined flags are:

0001 Show messages sent to the UndoManager.
0002 Show clsUndo initialization.
0004 Show msgUndoAddItem.
0008 Show undoing a undo transaction.
0010 Show creating a undo transaction.
0020 Show destroying an undo transaction.

The Current Transaction

At any time, there is at most one current undo transaction open. The current undo transaction includes:

- a unique id of type UNDO_ID
- the OS_TASK_ID of the task that issued the msgUndoBegin that began the transaction
- a nesting count which is the number of msgUndoBegin's minus the number of msgUndoEnd's. (See the section "Nesting of msgUndoBegin and msgUndoEnd.")
- a heap with local scope from which clients can allocate space for undo information
- a list of undo items added to the transaction so far.

The Undo Queue

theUndoManager maintains a queue of undo transactions. By default the UndoManager has a queue length of 2, but an application can set the limit by sending msgUndoLimit to the UndoManager.

Your code should not depend on any particular queue size.

Nesting of msgUndoBegin and msgUndoEnd

In response to msgUndoBegin, the UndoManager opens a new transaction if there is no open transaction; otherwise it simply increments a "nesting count." The nesting count is decremented when the UndoManager receives msgUndoEnd. When the count becomes zero, the transaction is closed.

This allows you to write code that doesn't know whether it there is an open transaction or not. If the code wants to record undo information, it can simply send a msgUndoBegin / msgUndoEnd pair. If there was no open transaction, the result is that one will be created. And if there is one open, then the code's items will be added to that one.

It is vital that every msgUndoBegin have a matching msgUndoEnd!

To guard against erroneous code never terminating the current transaction, and thus having that transaction slowly consume all of system memory, there is a bounds on the depth of nesting permitted. (This bounds is approximately 1000.) If the bounds is exceeded, the open transaction is automatically closed.
Memory Management

Each undo item records the information necessary to undo and/or free itself.

Often this information has to be remembered in allocated memory or objects that must be freed once the item can no longer be undone. For instance, an undoable operation might involve deleting an object. However, you probably don’t want to destroy the object until you’re sure that the operation can’t be undone. But eventually that object has to be destroyed.

Normally theUndoManager will send msgUndoFreeItemData to the object stored in each UNDO_ITEM. The handler should respond by freeing any resources associated with the item. Typically those resources are pointed to by item.pData.

But there are five ways in which you and theUndoManager can cooperate so that theUndoManager can free the resources for you.

- If uIDataIsHeapNode is set in item.flags, then item.pData must point to a heap block. theUndoManager will free item.pData by calling OSHeapBlockFree(item.pData).
- If uIDataInUndoHeap is set in item.flags, then item.pData must point to heap block allocated from the current transaction’s heap. theUndoManager will free item.pData when it destroys the transactions’s heap.
- If uIDataIsObject is set in item.flags, then item.pData must be an object UID. theUndoManager will free item.pData by calling ObjectSend(msgDestroy, item.pData, ...). (See the section "Freeing Undone Items" for one reason NOT to use this variation.)
- If uIDataIsSimple is set in item.flags, then item.pData is treated as a 32 bit value. There is no need for theUndoManager to do anything to free item.pData.
- If none of the above flags is set in item.flags, and item.dataSize is non-zero, then when the item is added to the transaction (with msgUndoAddItem) theUndoManager copies item.dataSize bytes from item.pData into a block allocated from the current transaction’s heap. theUndoManager then frees item.pData when it destroys the transaction’s heap.

Freeing Undone Items

Even an item that has been undone will be freed. It might be automatically freed by theUndoManager, as described in the section on Memory Management, or it might be freed by sending msgUndoFreeItemData to item.object.

Often freeing an item's data is done the same way regardless of whether the item has been undone or not. But there are cases where the difference is very important. Here's an example. Assume that the undoable operation includes deleting an object. If the operation is undone, then the object is "put back" into the application.

If the item IS undone, then the object should NOT be destroyed when the item is freed. But if the operation IS NOT undone, then the object should be destroyed when the object is destroyed.

For items that need to free the item's data differently in these two cases, the fact that the item has been undone should be recorded in the item when msgUndoItem is received. Then the code responding to msgUndoFreeItemData can check this recorded value. (One convenient place to record this value is in the item's ufClient flags.)
Adding Items When No Transaction is Open

When the UndoManager is undoing a transaction, there is no current open transaction. But, as described in the typical scenario above, data manipulation routines will attempt to add items anyhow. Therefore it is CRITICAL that your code check the value returned from msgUndoAddItem and handle it properly.

There are several ways to do this, but here's one convenient approach. (This approach works ONLY if you DON'T use any of the UndoManager's memory management functionality.)

If you're not using the memory management facilities of the UndoManager, then you're most likely allocating memory to hold the client data part of an undo item. That memory has been allocated before calling msgUndoAddItem and must be freed if the msgUndoAddItem fails. Conveniently, an item's client data can be freed by sending msgUndoFreeItemData to the object stored in item.object.

Simply define a utility routine that attempts to add an item, and which frees the item if adding fails. Then always use that routine to add items. The routine will look something like:

```
if (ObjectCall(msgUndoAddItem, theUndoManager, pItem) < stsOK) {
    return ObjCallWarn(msgUndoFreeItemData, pItem->object, pItem);
} else {
    return stsOK;
}
```

Subclass Issues

A class and any number of its ancestors may contribute items to an undo transaction.

Thus, every msgUndoFreeItemData handler should first check that item.subclass is the expected value. If it isn't, the message should be passed onto the ancestor. So a msgUndoFreeItemData handler should look something like:

```
MsgHandlerWithTypes(RTItemUndoFreeItemData, P_UNDO_ITEM, PP_DATA)
{
    if (pArgs->subclass != clsRTItem) {
        return ObjectCallAncestorCtx(ctx);
    } else {
        ...
    }
}
```

Flushing the Undo Queue

There may be "points of no return" in an application's execution beyond which undoing previous operations is impossible or non-sensical. (For instance, it may not be possible to undo operations if the application's data files are saved via msgAppSave.)

You should flush the queue when one of these "points of no return" is encountered. The queue can be flushed by performing the following three steps: (1) get the current undo limit via msgUndoGetMetrics, (2) send msgUndoLimit with a pArgs of 0 (which actually flushes the queue), and (3) send msgUndoLimit, but this time with the limited returned by the previous call to msgUndoGetMetrics.
Aborting a Transaction

Sometimes it is necessary to abort an operation part way through. (For instance, the user might not
confirm the operation.) If this happens, you should abort the then the undo transaction with
msgUndoAbort. See the comments on msgUndoAbort for more information.

ifndef UNDO_INCLUDED
#define UNDO_INCLUDED
ifndef LIST_INCLUDED
#include <list.h>
endif

Types and Constants

typedef STATUS UNDO_ID; // A transaction's id.
#define stsUndoAbortingTransaction MakeStatus(clsUndo, 1)
#define stsUndoDataFreed MakeWarning(clsUndo, 1)
#define undoStateNil 0
#define undoStateBegun flag0
#define undoStateUndoing flag1
#define undoStateRedoing flag2 // Not implemented
#define undoStateAborting flag3

Exported Functions

STATUS PASCAL
InitClsUndo(void);

Message Arguments

UNDO_ITEM

typedef struct UNDO_ITEM {
    OBJECT          object;  // In: object that undoes/frees item
    OBJECT          subclass; // In: See "Subclass Issues" section
    U16             flags;   // In: See "Memory Management" section
    PUnknown        pData;   // In: See "Memory Management" section
    SIZEOF           dataSize; // In: See "Memory Management" section
} UNDO_ITEM, *P_UNDO_ITEM;

The following flags are used in the flags field of an UNDO_ITEM.

#define ufReserved   (0xff00)
#define ufClient     (flag0|flag1|flag2|flag3)
#define ufDataType   (flag4|flag5|flag6|flag7|ufReserved)
#define ufDataInUndoHeap flag4
#define ufDataIsHeapNode flag5
#define ufDataIsObject (flag5|flag4)
#define ufDataIsSimple (flag6|flag4)

Other Message Arguments

typedef struct UNDO_METRICS {
    UNDO_ID         id;      // In:Out Nil => get current
    OS_HEAP_ID      heapId;  // Out
    U16             state;   // Out
    U16             transactionCount; // Out
    U16             itemCount; // Out
    U32             limit;   // Out
    U32             resid;  // Out
    U32             info;    // Reserved
} UNDO_METRICS, *P_UNDO_METRICS;
typedef struct UNDO_NEW_ONLY {
    U32 reserved;
    P_UNKNOWN pReserved;
    U32 maxTransactions;
} UNDO_NEW_ONLY, *P_UNDO_NEW_ONLY;

typedef struct UNDO_NEW {
    UNDO_NEW_ONLY undoNewFields
} UNDO_NEW, *P_UNDO_NEW;

# define undoNewFields
    \
    listNewFields
    UNDO_NEW_ONLY undo;

Messages

Next: 11; recycled: none

msgUndoAbort
Aborts the current undo transaction.

Takes pNull, returns STATUS.

#define msgUndoAbort MakeMsg(clsUndo, 10)

Comments
The current transaction is flagged as being aborted. Until the transaction is closed, any attempted msgUndoAddItem, msgUndoBegin, and msgUndoEnd (including the one that finally closes the transaction) will fail and return stsUndoAbortingTransaction. Once the msgUndoEnd that closes the transaction is received, any remaining undo items in the aborted transaction are freed.

msgUndoAddItem
Adds a new item to the current undo transaction if and only if it is still open.

Takes P_UNDO_ITEM, returns STATUS.

#define msgUndoAddItem MakeMsg(clsUndo, 0)

typedef struct UNDO_ITEM {
    OBJECT object;     // In: object that undoes/frees item
    OBJECT subclass;   // In: See "Subclass Issues" section
    U16 flags;         // In: See "Memory Management" section
    P_UNKNOWN pData;   // In: See "Memory Management" section
    SIZEOF dataSize;   // In: See "Memory Management" section
} UNDO_ITEM, *P_UNDO_ITEM;

Comments
theUndoManager returns stsFailed if an open transaction does not exist. Any other error status indicates that there are not enough resources available to add the item.

msgUndoBegin
Creates a new undo transaction if there is no current transaction, or increments the nesting count if there is a current transaction.

Takes RES_ID, returns STATUS or UNDO_ID.

#define msgUndoBegin MakeMsg(clsUndo, 1)
Return Value

stsFailed Nesting limit exceeded.

stsOK Returned status is actually the id of the new (or currently open) transaction. Cast it to type
UNDO_ID.

The RES_ID for a transaction is determined by the first msgUndoBegin with a non-null argument. The
string identified by the RES_ID of the current undo transaction is used as the string for the "Undo"
menu item. The RES_ID should specify a resGrpTK string resource list. (This is analogous to the quick
help strings that are found in the resGrpQHelp string resource list.)

**msgUndoCurrent**

Undoes the most recent undo transaction.

Takes pNull, returns STATUS.

```c
#define msgUndoCurrent MakeMsg(clsUndo, 2)
```

Comments

msgUndoCurrent undoes the most recent transaction. If a transaction is currently open the transaction
is closed first, and then undone.

It is unusual for a client to send this message. The only real reason for sending this message is if some
piece of client code is implementing an alternative UI mechanism to invoke the undo mechanism.

**msgUndoEnd**

Decrement the nesting count of (and thus may end) the current transaction.

Takes pNull, returns STATUS.

```c
#define msgUndoEnd MakeMsg(clsUndo, 3)
```

Comments

See the "Nesting of msgUndoBegin and msgUndoEnd" section for information about how to send this
message.

Return Value

stsFailed No open transaction.

**msgUndoGetMetrics**

Passes back the metrics associated with an undo transaction.

Takes P_UNDO_METRICS, returns STATUS.

```c
#define msgUndoGetMetrics MakeMsg(clsUndo, 4)
```

typedef struct UNDO_METRICS {
    UNDO_ID id;                 // In:Out Nil => get current
    OS_HEAP_ID heapId;          // Out
    U16 state;                  // Out
    U16 transactionCount;       // Out
    U16 itemCount;              // Out
    U32 limit;                  // Out
    U32 resId;                  // Out
    U32 info;                   // Reserved
} UNDO_METRICS, *P_UNDO_METRICS;

Comments

Only an pArgs->id of Nil(UNDO_ID), representing the current undo transaction, is supported.

Return Value

stsFailed The specified transaction does not exist or there is in sufficient memory available to
manipulate it.
**msgUndoLimit**

Sets the maximum number of remembered undo transactions.

Takes U32, returns STATUS.

```c
#define msgUndoLimit MakeMsg(clsUndo, 8)
```

**Comments**

The default undo limit is 2. If your application wants to support a longer undo history, send `msgUndoLimit` to `theUndoManager` with the desired limit.

If there are more transactions in the queue than the new limit, the extra transactions will be freed.

Setting the limit to 0 flushes all transactions and effectively disables undo until the limit is set to some non-zero value.

`msgUndoLimit` always returns `stsOK`.

**msgUndoRedo**

Not implemented.

Takes pNull, returns STATUS.

```c
#define msgUndoRedo MakeMsg(clsUndo, 5)
```

**Comments**

Not implemented. Do not send this message.

---

### Client Messages

#### msgUndoItem

Sent to `pArgs->object` to have the item undone.

Takes P_UNDO_ITEM, returns STATUS.

```c
#define msgUndoItem MakeMsg(clsUndo, 6)
```

**Message Arguments**

- `OBJECT object;` // In: object that undoes/frees item
- `OBJECT subclass;` // In: See "Subclass Issues" section
- `U16 flags;` // In: See "Memory Management" section
- `P_UNKNOWN pData;` // In: See "Memory Management" section
- `SIZEOF dataSize;` // In: See "Memory Management" section

**Comments**

Note that the item will be freed in a separate step later.

#### msgUndoFreeItemData

Sent to `pArgs->object` to have `pArgs->pData` freed.

Takes P_UNDO_ITEM, returns STATUS.

```c
#define msgUndoFreeItemData MakeMsg(clsUndo, 7)
```

**Message Arguments**

- `OBJECT object;` // In: object that undoes/frees item
- `OBJECT subclass;` // In: See "Subclass Issues" section
- `U16 flags;` // In: See "Memory Management" section
- `P_UNKNOWN pData;` // In: See "Memory Management" section
- `SIZEOF dataSize;` // In: See "Memory Management" section

**Comments**

See the "Memory Management," "Subclass Issues" and "Freeing Undone Items" sections for information about how to respond to this message.
This file contains the API definition for clsXfer and clsXferList.

clsXfer inherits from clsStream.

clsXfer defines the mechanisms used for transferring data between objects.

clsXferList inherits from clsList.

clsXferList is used by the transfer mechanism.

Most clients of PenPoint's data transfer mechanism should use the procedural interfaces defined in this file.

The functions described in this file are contained in XFER.LIB.

Introduction

Key Concepts

This file describes some of PenPoint's support for transferring data.

There are a few central concepts that underlie PenPoint's data transfer mechanism:

- Sender and Receiver. There are two sides to any data transfer. "Sender" refers to the object providing the data and "Receiver" refers to the object receiving the data. These two objects can be in different processes, or in the same process. They can even be the same object!

- Two Stages. Each PenPoint data transfer has two major stages. In the first stage the Sender and Receiver engage in a simple protocol to determine if the data can be transferred, and if so what "type" the data has. In the second stage, the data is actually transferred using a protocol that is specific to the type agreed to during Stage 1.

- Data Transfer Types. A Sender and Receiver need to agree on a data transfer type that they both understand. PenPoint defines several data transfer types and clients can define additional types. See the section "Determining a Data Transfer Type" for more information.

- Data Transfer Protocol. Each data transfer type has an associated data transfer protocol. Once a transfer type has been agreed upon, the Sender and Receiver engage in the type-specific protocol to actually move the data. Note the same Data Transfer Protocol can be employed for multiple Data Transfer Types, but that each Data Transfer Type uses one and only one protocol.

Roadmap

Typical Receivers use the following to determine the desired data transfer type.

- XferMatch()

Typical Senders respond to or use the following to provide a list of data transfer types.

- msgXferList
- XferAddIds()
Typical Senders and Receivers who use data transfer types that use one-shot protocols use the following:
  • msgXferGet

Senders and Receivers who use data transfer types that use stream-based protocols use the following:
  • msgXferStreamConnect
  • msgXferStreamWrite
  • msgXferStreamFreed
  • XferStreamConnect()
  • XferStreamAccept()

Relationship between Data Transfer and PenPoint’s UI

PenPoint’s data transfer mechanism is intentionally independent of the user interface that might trigger a data transfer. None of the interfaces defined in this file depend or define any part of a PenPoint application’s user interface.

However, the examples given in the commentary often use PenPoint’s UI as an example of how a data transfer might be started. The file sel.h describes PenPoint’s Move and Copy operations in detail.

During a Move or Copy operation, the Sender object is the owner of the selection. The Receiver is the object upon which the move/copy icon was dropped and which receives msgSelMoveSelection or msgSelCopySelection as a result. The Receiver sends msgSelOwner to theSelectionManager to get the Sender object and then engages in a data transfer with that object.

A Typical Scenario

A typical data transfer session goes something like this:
  • The Receiver decides that it is the receiving end of a data transfer operation. (For instance, the receiver might receive msgSelMoveSelection or msgSelCopySelection; see sel.h.)
  • The Receiver figures out the UID of the Sender object. (For instance, in the case of msgSelCopySelection or msgSelMoveSelection, the Sender object is the current selection owner, which can retrieved by sending msgSelOwner to theSelectionManager.)
  • The Receiver determines a mutually agreeable data transfer type using the utility routine XferMatch. (See section "Determining a Common Data Transfer Type" for more detailed information about XferMatch and alternatives.)
  • The Sender and Receiver use the Data Transfer Protocol associated with the agreed-upon type to actually transfer the data.

Data Transfer Types

A data transfer type is represented by a TAG.

Below is a list of PenPoint’s predefined data transfer types and the data transfer protocol associated with each. (The protocols are described in the next section.)

- : xferString: one-shot using XFER_FIXED_BUF
- : xferLongString: one-shot using XFER_BUF
- : xferName: one-shot using XFER_FIXED_BUF
- : xferFullPathName: one-shot using XFER_FIXED_BUF
- : xferRTF: stream
In addition export.h and embedwin.h each define an additional data transfer type; see these files for more information.

Determining a Common Data Transfer Type

The Sender and Receiver must agree on a data transfer type.

For instance, a note taking application might be willing to provide either xferScribbleObject or xferLongString data. A text editor might be willing to consume xferString, xferLongString or xferRTF data. Somehow the common data type (xferLongString) must be found and used.

In PenPoint's data transfer mechanism, the Receiver is ultimately responsible for determining the mutually agreeable data transfer type.

Typical Receivers can use a simply utility function, XferMatch, to compute the data transfer type. Typical Senders must respond to msgXferList and add data transfer types to the provided list with the utility function XferAddIds.

(Most clients don’t need to know about the inner workings of XferMatch, but they are documented in the section "Details of XferMatch" for sophisticated clients or the merely curious.)

Data Transfer Protocols

Each data transfer type uses a specific data transfer protocol.

There are three types of protocols:
- one-shot protocols
- stream-based protocols
- client-defined protocols

One Shot Protocols

Several data transfer types use a "One-Shot" protocol to transfer data. The protocols are called "one-shot" because all of the data can be transferred via a single message send.

In all one-shot transfers, the Receiver uses ObjectSendUpdate to send msgXferGet to the Sender. (ObjectSendUpdate must be used because the Sender and Receiver might be in different processes.)

The type of the pArgs to msgXferGet depends on the data transfer type -- the specific types are described in the section "Data Transfer Types." However, all legal pArgs to msgXferGet have one thing in common -- their first field is a data transfer type. The Receiver must fill in at least this field before sending msgXferGet so that the Sender can tell which data transfer type is being used.

The Sender responds to msgXferGet by filling in pArgs as necessary. Some one-shot protocols require the Sender to allocate memory. (For instance, the xferLongString data transfer type requires that the sender allocate memory for pArgs->pBuf field of an XFER_BUF.)

Some one-shot protocols require that Sender allocate memory. Any Sender-allocated memory must be allocated using OSHeapBlockAlloc and osProcessSharedHeapId. The Receiver must free this memory with OSHheapBlockFree.
Stream-Based Protocols

Stream-based protocols make use of a specialized stream that is implemented by clsXferStream. clsXferStream adds the ability for two streams to be linked through an internal "pipe."

Once a Receiver has decided to engage in a stream-based transfer (as described in the Section "A Typical Scenario" earlier), the steps in stream-based protocol are as follows:

• The Receiver calls XferStreamConnect.
• XferStreamConnect creates the Receiver’s stream and then sends msgXferStreamConnect to the Sender.
• In response to msgXferStreamConnect, the Sender calls XferStreamAccept. Essentially all Senders of stream-based protocols should pass self as the "Producer" parameter when they call XferStreamAccept -- motivation and exceptions are described below.
• XferStreamAccept properly creates the Sender’s stream.
• When control returns to it, the Receiver sends msgStreamReadData to its stream.
• As a result of the Receiver’s msgStreamReadData, the Sender receives msgXferStreamWrite.
• In response to msgXferStreamWrite, the Sender writes data using msgStreamWriteData.
• IMPORTANT NOTE: In order to avoid overflowing internal buffers, Senders should not write huge chunks of data in a single call. Chunks that exceed 64K won’t work at all. Memory is used more efficiently if chunk sizes don’t exceed 10K, although things will work at any size up to 64K.
• The last two steps can be repeated any number of times. Eventually the Receiver gets stsEndOfData returned when sending msgStreamReadData.
• The Receiver sends msgDestroy to its stream.
• As a result of the Receiver’s msgDestroy, the Sender receives msgXferStreamFree.
• In response to msgXferStreamFree, the Sender sends msgDestroy to its stream.

The Sender must be prepared to handle msgXferStreamFreed at any time. (In addition to normal termination, msgXferStreamFreed can indicate that the Receiver has died or otherwise has prematurely destroyed its side of the pipe.)

An Available Simplification

Some Senders may know that they can contain only a limited amount of data. Or they may find the obligation to respond to msgXferStreamWrite multiple times and record how much data was actually written each time to be unduly burdensome.

These Senders can pass objNull as the "Producer" parameter in their call of XferStreamConnect. As a result of doing this, msgXferStreamWrite will only be sent once, and in response these Senders should write all of their data in a single chunk.

Client-Defined Protocols

Clients can define their own data transfer types. There is a wide range of possibilities. Clients can use msgXferGet that use a new pArgs type. They can use streams but define structure on the data being streamed. Or they define an entirely new transfer protocol.
Other Information

Details of XferMatch

Most clients can simply use XferMatch without understanding how it works, but it's described here for specialized clients or the curious.

- XferMatch creates an instance of clsXferList
- It then sends msgXferList to the passed-in Sender.
- The Sender responds to msgXferList by adding items to the xfer list by calling XferAddIds.
- XferMatch then scans the two lists (one passed in by the Receiver and one filled in by the Sender) using the utility function XferListSearch.
- If no mutually acceptable data transfer type is found, XferMatch returns stsNoMatch. Otherwise XferMatch returns stsOK and passes back the data transfer type in *pId.
- Just before returning, XferMatch destroys the xferList.

As an alternative to calling XferMatch, the Receiver could create the list, send msgXferList to the Sender, and then search the list for the best match (perhaps by using XferListSearch).

Also, a sophisticated Sender can use msgListAddItem (rather than XferAddIds) to add the types to the list.

Creating Instances of clsXfer and clsXferList

Normal clients of PenPoint's data transfer mechanism have no need to create instances of clsXfer and clsXferList. Instances are created internally when using the data transfer functions.

```c
#ifndef XFER_INCLUDED
#define XFER_INCLUDED
#endif
#ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif
#ifndef STREAM_INCLUDED
#include <stream.h>
#endif
#ifndef STREAM_INCLUDED
#include <list.h>
#endif
```

Common #defines and typedefs

Predefined Data Transfer Types

```c
#define xferString MakeTag(clsXfer, 1)  // XferGet (FixedBuf)
#define xferLongString MakeTag(clsXfer, 2)  // XferGet (Buf)
#define xferName MakeTag(clsXfer, 3)  // XferGet (FixedBuf)
#define xferFullPathName MakeTag(clsXfer, 4)  // XferGet (FixedBuf)
#define xferRTF MakeTag(clsXfer, 5)  // Stream
#define xferGoRTF MakeTag(clsXfer, 6)  // Obsolete
#define xferFlatLocator MakeTag(clsXfer, 7)  // XferGet (FixedBuf)
#define xferASCIIMetrics MakeTag(clsXfer, 10)  // XferGet (AsciiMetrics)
#define xferScribbleObject MakeTag(clsXfer, 11)  // XferGet (Object)
#define xferPicSegObject MakeTag(clsXfer, 12)  // XferGet (Object)
```
**XferList**

Normal clients need not create xferLists since the functions create and destroy xferLists as needed.

An xferList is a subclass of clsList that always allocates globally accessible memory for the list.

```c
#define XFER_LIST_NEW LIST_NEW
#define P_XFER_LIST_NEW P_LIST_NEW
```

**Messages**

**msgXferList**

Ask Sender for its list of data transfer types.

Takes OBJECT, returns STATUS.

```c
#define msgXferList MakeMsg(clsXfer, 1)
```

**Comments**

This message is sent to the Sender to have the Sender provide the list of data transfer types it can provide.

The Sender can add types to the passed-in list using either msgListAddItem or XferListAddIds.

If the Sender has a preferred data transfer type, it should put this type at the beginning of the list. The Sender can use clsList messages to change the ordering of the list (see list.h).

**See Also**

msgListAddItems

**msgXferGet**

Sent by a Receiver to get "one-shot" data transfer information.

Takes lots-of-things, returns STATUS.

```c
#define msgXferGet MakeMsg(clsXfer, 8)
```

**Comments**

msgXferGet is sent by the Receiver to the stream to retrieve the data being transferred.

The type of this message's pArgs depends on the data transfer type being used. In all cases, the first field of pArgs must be a data transfer type so that the Sender (when it receives this message) knows what type of data to supply and what the true type of pArgs really is.

**Return Value**

staNoMatch specified data transfer type is inappropriate

**Variable Size Buffer**

This type is used as the pArgs of msgXferGet when the data transfer type is xferLongString. This type might also be used for client-defined data transfers.

[The rest of this description is complicated by the reversal of names. The Receiver side of the data transfer operation sends msgXferGet and the the Sender side of the data transfer operation receives msgXferGet.]

The Receiver (which sends msgXferGet) must set the "id" field to xferLongString. The Sender receives msgXferGet and fills in the rest of the structure.

The Sender allocates the memory for pArgs->pBuf using OSHeapBlockAlloc from osProcessSharedHeapId. The Receiver must free this data using OSHeapBlockFree.
When used for `xferLongString`, the "pBuf" field is a null-terminated string and the "len" field includes the terminating null character. (In other words, upon return, `pArgs->len` must equal (`strlen(pArgs->pBuf) + 1`).)

typedef struct XFER_BUF {
    TAG id; // In: Data transfer type
    U32 data; // Unused: future use
    U32 len; // Out: Length of data in pBuf
    P UNKNOWN pBuf; // Out: Buffer containing data
} XFER_BUF, *P_XFER_BUF;

**Fixed Size Buffer**

This type is used as the `pArgs` of `msgXferGet` when the data transfer type is:
- `xferString`
- `xferName`
- `xferFullPathName`
- `xferFlatLocator`

[The rest of this description is complicated by the reversal of names. The Receiver side of the data transfer operation sends `msgXferGet` and the the Sender side of the data transfer operation receives `msgXferGet`.]

The Receiver (which sends `msgXferGet`) must set the "id" field to one of the data transfer types listed above. The Sender receives `msgXferGet` and fills in the rest of the structure.

typedef struct XFER_FIXED_BUF {
    TAG id; // In: Data transfer type
    U32 data; // Unused. Reserved for future use
    U32 len; // Out: Length of data in buf
    U8 buf[300]; // Out: Buffer containing data
} XFER_FIXED_BUF, *P_XFER_FIXED_BUF;

**Object Transfer**

This type is used as the `pArgs` of `msgXferGet` when the data transfer type is:
- `xferScribbleObject`
- `xferPicSegObject`

[The rest of this description is complicated by the reversal of names. The Receiver side of the data transfer operation sends `msgXferGet` and the the Sender side of the data transfer operation receives `msgXferGet`.]

The Receiver (which sends `msgXferGet`) must set the "id" field to one of the data transfer types listed above, and must set the "receiver" field to self (or some other object in the Receiver's task). The Sender receives `msgXferGet` and fills in the rest of the structure.

The Sender makes a copy of the object using `msgCopy` and returns the uid of the object in `pArgs->uid`. When the Sender sends `msgCopy`, it should use `pArgs->receiver` as the value of `msgCopy`'s `pArgs->requestor`.

typedef struct XFER_OBJECT {
    TAG id; // In: Data transfer type
    OBJECT receiver; // In: Receiver
    OBJECT uid; // Out: Uid of object
    CLASS objClass; // Out: Class of object
    U32 reserved[4]; // Reserved for future use
} XFER_OBJECT, *P_XFER_OBJECT;
**ASCII Metrics**

This type is used as the pArgs of msgXferGet when the data transfer type is xferASCII

[The rest of this description is complicated by the reversal of names. The Receiver side of the data transfer operation sends msgXferGet and the the Sender side of the data transfer operation receives msgXferGet.]

The Receiver (which sends msgXferGet) must set the "id" field to xferASCII. The Sender receives msgXferGet and fills in the rest of the structure.

"ASCII Metrics" include information about the character data that can be transferred from the Sender. In some cases (e.g. PenPoint’s text component) it describes the selected text.

(Essentially any Sender that can provide xferASCII can also provide some type of character data -- typically xferString, xferLongString or xferRTF)

The "spare" field is always set to 0. The "first" field is offset of the first selected character. The "length" field is the number of characters in the selection. The "level" field describes which lexical unit the selection "contains."

```c
typedef struct XFER_ASCII_METRICS {
    TAG id;  // In: data transfer type.
    U32 spare;  // Out: always 0
    U32 first;  // Out: character offset w.r.t. entire text
    // maxU32 implies a bad request
    U32 length;  // Out: number of chars available to transfer
    U16 level;  // Out: 0: undefined or unknown, 1: chars,
    // 2: words, 3: sentences, 4: paragraphs
} XFER_ASCII_METRICS, *P_XFER_ASCII_METRICS;
```

**Stream Specific Messages**

**msgXferStreamConnect**

Sent to the Sender to ask it to link the Sender’s and Receiver’s pipe.

Takes XFER_CONNECT, returns STATUS.

```c
#define msgXferStreamConnect MakeMsg(clsXfer, 2)
```

**Arguments**

```c
typedef struct XFER_CONNECT {
    TAG id;   // In: Id Receiver sent to XferStreamConnect
    OBJECT stream;   // In: Stream created by Receiver
    P UNKNOWN clientData;   // In: clientData Receiver sent to
    // XferStreamConnect
} XFER_CONNECT, *P_XFER_CONNECT;
```

**Comments**

The Sender responds by calling XferStreamAccept to complete the connection.

In its call to XferStreamAccept, the Sender identifies the object that will generate the actual data, known as the Producer. Essentially all Senders should pass self as the value of Producer.

See the section "Stream-Based Protocols" for more information.

**msgXferStreamAuxData**

Passes back auxiliary information associated with the pipe.

Takes PP_UNDEFINED, returns STATUS.

```c
#define msgXferStreamAuxData MakeMsg(clsXfer, 4)
```
Comments

The Sender or Receiver can store auxiliary information with the pipe, using `msgXferStreamSetAuxData` and retrieve that information with `msgXferStreamAuxData`.

This information can be used by either the Sender or Receiver to store private information or to pass information across the pipe.

Warning: There is only one auxiliary data slot in the pipe. Only one of the Sender or Receiver should write the data, although both can read it. Subclasses must be aware of their ancestor's behavior in this regard.

See Also

`msgXferStreamSetAuxData`

---

`msgXferStreamSetAuxData`

Stores arbitrary client data with the pipe.

Takes P_UNKNOWN, returns STATUS.

```c
#define msgXferStreamSetAuxData MakeMsg(clsXfer, 5)
```

See Also

`msgXferStreamAuxData`

---

`msgXferStreamWrite`

Asks the Sender to write more data to the stream.

Takes STREAM, returns STATUS.

```c
#define msgXferStreamWrite MakeMsg(clsXfer, 3)
```

Comments

The Sender responds by writing to its stream using `msgStreamWrite`. The Sender may need access to its instance data to handle this message. The Sender can either implement its own facility for mapping from the stream to the necessary instance data (perhaps using properties; see clsmgr.h) or it can use `msgXferStreamSetAuxData` and `msgXferStreamAuxData`.

See the section "Stream-Based Protocols" for more information.

---

`msgXferStreamFreed`

Sent to the Sender when the Receiver's side of the stream has been freed.

Takes STREAM, returns STATUS.

```c
#define msgXferStreamFreed MakeMsg(clsXfer, 6)
```

Comments

The Sender handles this message by sending `msgDestroy` to the stream passed in as a parameter. This means that both streams (and hence both ends of the "pipe") have been freed.

See the section "Stream-Based Protocols" for more information.

---

**Public Functions**

---

`XferMatch`

The Receiver calls `XferMatch` to find a mutually acceptable data transfer type.

Returns STATUS.

```c
STATUS EXPORTED XferMatch(
    OBJECT Sender,          // In: Sender to find match with
    TAG ids[],              // In: Array of types the Receiver understands
    sizeof idsLen,         // In: Length of the ids[] array
    P_TAG pId);            // Out: matching data type
```
See the section "Determining a Common Data Transfer Type" for detailed information.

**stsNoMatch** No common data transfer type could be found.

**non-error** The common data transfer type is passed back in *pld.

### XferListSearch

Searches two sets of data transfer types for a match.

Returns STATUS.

**Function Prototype**

```
STATUS EXPORTED XferListSearch(
    OBJECT listObject, // In: List object containing Sender types
    TAG ids[],       // In: Array of types the Receiver understands
    SIZEOF idsLen,  // In: Length of the ids[] array
    P_TAG pId);     // Out: Matching data type
```

**Comments**

Most clients of the data transfer mechanism use XferMatch rather than calling this function.

XferListSearch scans the two sets of transfer types (one in listObject and one in the passed-in array) to find the best match.

XferListSearch checks each item in listObject against each item in the array in order from 0 to n-1. Hence if the array contains [tagA, tagB] and the list contains [tabB, tagA], tagA is returned. Objects should put data types into the listObject or the array in order of most desired to least desired.

**Return Value**

**stsNoMatch** No common data transfer type could be found.

**non-error** The common data transfer type is passed back in *pld.

### XferAddIds

Adds data transfer types to listObject.

Returns STATUS.

**Function Prototype**

```
STATUS EXPORTED XferAddIds(
    OBJECT listObject,
    TAG ids[],
    SIZEOF idsLen);
```

**Comments**

Typical Senders call this function while handling msgXferList.

XferAddIds adds each item in the array of data transfer types to the list by sending msgListAddItem to listObject.
Stream Specific Functions

XferStreamConnect

A Receiver calls this function to create a stream connection to a Sender.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED XferStreamConnect(
    OBJECT owner, // In: Sender to connect stream to
    TAG id, // In: Desired data transfer type. (This is
             // passed to Sender via msgXferStreamConnect.)
    P_UNKNOWN clientData, // In: clientData. (This is passed to Sender
                             // via msgXferStreamConnect.)
    P_OBJECT pStream); // Out: Stream to perform msgStreamRead on
```

Comments

See the section "Stream-Based Protocols" for more information.

XferStreamAccept

Called by Sender in response to msgXferStreamConnect.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED XferStreamAccept(
    OBJECT connect, // In: pArgs->stream from msgXferStreamConnect
    UI6 bufSize, // In: Size of transfer buffer (up to 64k)
    OBJECT Producer, // In: Object to receive msgXferStreamWrite
    P_OBJECT pStream); // Out: Stream for Sender side of the "pipe"
```

Comments

As part of the Sender's response to msgXferStreamConnect, the Sender calls XferStreamAccept to properly create the Sender's side of the stream.

See the section "Stream-Based Protocols" for more information.
Part 10 / Connectivity
This file contains the API definition for theAddressBookMgr.

theAddressBookMgr is an instance of a private class. It is the only instance of that class in the system.

theAddressBookManager is a well known object that handles registration of and access to "system" address books. Registered address books are primarily responsible for managing the storage and retrieval of service specific addressing information.

Registered address books adhere to the protocol defined in addrbook.h. Information about its functionality and use can be found there.

theAddressBookMgr provides the facility to help other applications to provide a UI for picking the system address book. When an application wants to provide this pick list as an option card, it just needs to pass on msgOptionAddCards before it calls its ancestor to theAddressBookMgr. TheAddressBookMgr will do the rest.

ifndef ABMGR_INCLUDED
#define ABMGR_INCLUDED
#include <uuid.h>
#include <go.h>
define tagABMgrABList

Status Codes

#define stsABMgrAddrBookNotActive
#define stsABMgrAddrBookOpen
#define stsABMgrNoneActive
#define stsABMgrAddrBookNotRegistered
#define stsABMgrNoOpenAddrBook

Common #defines and typedefs

Enum16(AB_MGR_ID_TYPE) { 
    abMgrApplication = 0, // Client is an application
    abMgrObject = 1, // Client is a service/data object
    abMgrNone = 2, // abmgr internal use only
};

typedef struct AB_MGR_ID { 
    CHAR name[nameBufLength]; // Name of the address book
    AB_MGR_ID_TYPE type; // Address book object type
    union {
        OBJECT uid;
        UUID uid;
    } value;
} AB_MGR_ID, *P_AB_MGR_ID;
Messages

msgABMgrRegister

Registers an application or a service as an address book instance.

Takes P_AB_MGR_ID, returns STATUS.

#define msgABMgrRegister MakeMsg(theAddressBookMgr, 1)

typedef struct AB_MGR_ID {
    CHAR name[nameBufLength];  // Name of the address book
    AB_MGR_ID_TYPE type;  // Address book object type
    union {
        OBJECT uid;  // UID of the service/object
        UUID uuid;  // UUID of the application working dir
    } value;
} AB_MGR_ID, *P_AB_MGR_ID;

When an instance of an address book is registered with theAddressBookMgr, it can later be selected as "the system address book".

Address books send this message to register themselves with theAddressBookMgr. Each instance of each address book should be registered with theAddressBookMgr. If an address book application is a subclass of clsAddrBookApplication (see addrbook.h), then theAddressBookMgr automatically registers a newly created instance of this class.

If an address book is an application, theAddressBookMgr will automatically re-registers the app on warm boot. If an address book is a service, however, it would have to re-register itself after a warm boot.

msgABMgrUnregister

Unregisters an application or a service as an address book instance.

Takes P_AB_MGR_ID, returns STATUS.

#define msgABMgrUnregister MakeMsg(theAddressBookMgr, 2)

When an application instance is deleted, or when a service is de-installed. If an address book application is a subclass of clsAddrBookApplication (see addrbook.h), then theAddressBookMgr automatically unregisters a deleted instance of this class.

msgABMgrOpen

Used by address book clients to begin access to address books.

Takes nothing, returns STATUS.

#define msgABMgrOpen MakeMsg(theAddressBookMgr, 3)
Address book clients send `msgABMgrOpen` to `theAddressBookMgr`. If the system address book is an application, then `theAddressBookMgr` activates the application. If the system address book is a service, then `theAddressBookMgr` binds to the service(`msgSMBind`).

Clients must call `msgABMgrClose` when they're finished with the address book.

On warm boots, `theAddressBookMgr` requires that clients reopen the system address book.

### `msgABMgrClose`

Used by address book clients to end access to address books.

Takes nothing, returns `STATUS`.

```c
#define msgABMgrClose MakeMsg(theAddressBookMgr, 4)
```

### Arguments

```c
typedef struct {
    BOOLEAN activated;
    AB_MGR_ID addressBook;
} AB_MGR_LIST, *P_AB_MGR_LIST;
```

### Comments

If the system address book is an application, then `theAddressBookMgr` deactivates the application. If the system address book is a service, then `theAddressBookMgr` binds to the service(`msgSMUnbind`).

The address book is reference counted, so all `msgABMgrOpen` calls must be followed by an `msgABMgrClose`.

### `msgABMgrList`

Creates a list of currently registered address book in `pArgs`.

Takes `P_LIST`, returns `STATUS`.

```c
#define msgABMgrList MakeMsg(theAddressBookMgr, 5)
```

### Comments

Every time `msgABMgrList` is called, a new list object is created. It is up to the client to call `msgListFree`(`not msgDestroy`) to destroy the list and the items in the list. Set the free mode to `listFreeItemsAsData`.

Each element of the list is a `P_AB_MGR_LIST`.

### `msgABMgrActivate`

Make a registered address book the system address book.

Takes `P_AB_MGR_ID`, returns `STATUS`.

```c
#define msgABMgrActivate MakeMsg(theAddressBookMgr, 6)
```

### Arguments

```c
typedef struct AB_MGR_ID {
    CHAR name[nameBufLength]; // Name of the address book
    AB_MGR_ID_TYPE type; // Address book object type
    union {
        OBJECT uid; // UID of the service/object
        UUID uid; // UUID of the application working dir
    } value;
} AB_MGR_ID, *P_AB_MGR_ID;
```

### Comments

In the current implementation only one address book can be the system address book at a time. If there is currently a system address book, that address book is deactivated first.

Clients that are applications set the type field to 'application' and set the value field to the UUID of their application working directory. Clients that are services or data objects set the type field to 'object' and set the value field to their object UID.
msgABMgrDeactivate

Deactivates the current system address book.

Takes P_AB_MGR_ID, returns STATUS.

```c
#define msgABMgrDeactivate
    MakeMsg(theAddressBookMgr, 7)
```

```c
typedef struct AB_MGR_ID {
    CHAR name[nameBufLength]; // Name of the address book
    AB_MGR_ID_TYPE type; // Address book object type
    union {
        OBJECT uid;
        UUID uuid; // UID of the service/object
    } value;
} AB_MGR_ID, *P_AB_MGR_ID;
```

---

msgABMgrIsActive

Indicates if the specified AB_MGR_ID is currently set.

Takes P_AB_MGR_ID, returns STATUS.

```c
#define msgABMgrIsActive
    MakeMsg(theAddressBookMgr, 8)
```

```c
typedef struct AB_MGR_ID {
    CHAR name[nameBufLength]; // Name of the address book
    AB_MGR_ID_TYPE type; // Address book object type
    union {
        OBJECT uid;
        UUID uuid; // UID of the service/object
    } value;
} AB_MGR_ID, *P_AB_MGR_ID;
```

---

msgABMgrChanged

Sent to observers of theAddressBookMgr when the system address book changes.

Takes P_AB_MGR_NOTIFY, returns STATUS.

```c
#define msgABMgrChanged
    MakeMsg(clsAddressBook, 9)
```
Enum16 (AB_MGR_CHANGE_TYPE) {
    abMgrRegister = 0,       // an ab has been registered
    abMgrUnregister = 1,
    abMgrActivated = 2,
    abMgrDeactivated = 3,
    abMgrOpened = 4,
    abMgrClosed = 5,
};
typedef struct {
    AB_MGR_CHANGE_TYPE type;
    AB_MGR_ID addressBook;
} AB_MGR_NOTIFY, *P_AB_MGR_NOTIFY;

pArgs->activated is set to TRUE if pArgs->addressBook is made the system address book, and to FALSE
if pArgs->addressBook has been deactivated as the system address book.
clsAddressBook inherits from clsObject.

This header file defines the address book protocol.

The address book protocol defines what minimal set of information is to be kept by an address book app or service, how information is to be stored, retrieved, queried by an address book client. Please refer to abmgr.h for information on address book manager.

All requests to access address book information is channeled through the address book manager. There can be multiple address book clients at one time. Whether or not address book clients can access information from more than address book application/service simultaneously is completely up to the implementation of the address book manager. The current implementation of the AddressBookMgr provided by GO only allows access to one address book at a time.

Because the AddressBookMgr uses ObjectSend to relay messages to address books, pointers in pArgs in any address book protocol messages should point to some shared memory space.

There are 3 major types of address information defined by the protocol:

- individual personal information(e.g. name, phone number, street address)
- service information(individual's fax phone number, email address, etc)
- distribution list information

All information is kept/retrieved in attribute-value form. The basic entity in an address book is an "entry", all information is presented relative to an entry. E.g. to access any information in an address book, a "key" to an entry must be presented. Within an entry, a client can set/get entry related information(name, street address, etc.). Service address information is also kept as part of an entry. Because there can be multiple service addresses for each entry(e.g. an individual has 2 fax numbers and 1 email address), a service address is accessed through a "service id" or the name of the service.(e.g. service name = "fax")

The Address Book Protocol specifies a minimum set of attributes and attribute types to be supported by third party address book applications or services. If a developer thinks that some addition attributes or attribute types are common enough that they should be defined in the protocol, please contact GO Corporation Developer Support.

```
#ifndef ADDRBOOKINCLUDED
#define ADDRBOOKINCLUDED
#endif

#ifndef GOINCLUDED
#include <go.h>
#endif

#ifndef UIDINCLUDED
#include <uid.h>
#endif

#ifndef CLSMGRINCLUDED
#include <clsmgr.h>
#endif

#ifndef DIALENVINCLUDED
#include <dialenv.h>
#endif
```
Common `#defines` and `typedefs`:

All address book apps should be a sub-class of this app. Being a sub-class of `clsAddrBookApplication` frees an address book application from having to register, and unregister itself with `TheAddressBookMgr`. The `AddressBookMgr` will notice when an instance of `clsAddrBookApplication` has been created/destroyed, and will automatically register/unregister the instance. Aside from providing this auto registration/unregistration, `clsAddrBookApplication` provides no other special behavior to its sub-class.

```c
#define clsAddrBookApplication MakeWKN(3284, 1, wknGlobal)
```

Pre-defined Attribute Types:

- `#define abNumber` MakeTag(clsAddressBook, 0) // 32-bit number
- `#define abString` MakeTag(clsAddressBook, 1) // null-terminated string
- `#define abPhoneNumber` MakeTag(clsAddressBook, 2) // `DIALENV_TELEPHONE_NUMBER`
- `#define abOther` MakeTag(clsAddressBook, 3) // some encoded byte array
  // interpreted by address
  // books simply as a byte
  // stream

Pre-defined attribute ids:

- `#define AddrBookGroupNameId` MakeTag(clsAddressBook, 0) // `abString`
- `#define AddrBookGivenNameId` MakeTag(clsAddressBook, 1) // `abString`
- `#define AddrBookSurnameId` MakeTag(clsAddressBook, 2) // `abString`
- `#define AddrBookHomePhoneNumberId` MakeTag(clsAddressBook, 3) // `abPhoneNumber`
- `#define AddrBookBusinessPhoneNumberId` MakeTag(clsAddressBook, 4) // `abPhoneNumber`
- `#define AddrBookCountryId` MakeTag(clsAddressBook, 5) // country in post
  // addr, abString
- `#define AddrBookStateId` MakeTag(clsAddressBook, 6) // state or pref-
  // cture, abString
- `#define AddrBookZiplId` MakeTag(clsAddressBook, 7) // zip, abString
- `#define AddrBookCityId` MakeTag(clsAddressBook, 8) // city, abString
- `#define AddrBookDistrictId` MakeTag(clsAddressBook, 9) // ku in Japanese
  // addr, abString
- `#define AddrBookStreetId` MakeTag(clsAddressBook, 10) // `abString`
- `#define AddrBookCompanyId` MakeTag(clsAddressBook, 11) // `company name,
  // abString`
- `#define AddrBookTitleId` MakeTag(clsAddressBook, 12) // `title of an
  // individual entry
  // abString`
- `#define AddrBookPositionId` MakeTag(clsAddressBook, 13) // `position of an
  // individual entry
  // abString`
- `#define AddrBookNicknameId` MakeTag(clsAddressBook, 14) // `nickname of an
  // individual entry
  // abString`
- `#define AddrBookBusinessPhoneNumber2Id` MakeTag(clsAddressBook, 15) // `2nd business
  // phone #
  // abPhoneNumber`
- `#define AddrBookFaxId` MakeTag(clsAddressBook, 16) // `fax # of an
  // individual entry
  // abPhoneNumber`
- `#define AddrBookSvcNameId` MakeTag(clsAddressBook, 17) // `name of svc,
  // abString`
Common #defines and typedefs

#define AddrBookSvcNoteId MakeTag(clsAddressBook, 13)  // user defined
                      // svc nickname
                      // abString
#define AddrBookSvcShortId MakeTag(clsAddressBook, 14)  // service short
                      // address

The following two special id's are used in specifying a query

#define AddrBookEntryKeyId MakeTag(clsAddressBook, 15)
#define AddrBookSvcIdId MakeTag(clsAddressBook, 16)

This is the type for address book transfer protocol. If an address book supports move/copy protocol,
then it should transfer an entry in a XFER_BUF structure, where XFER_BUF.pBuf is a pointer to
ADDR_BOOK_ENTRY structure.

#define AddrBookXferType MakeTag(clsAddressBook, 17)
#define AddrBookAll       (maxU16)
#define AddrBookAllSvcSelectAttrs (maxU16-1)
#define AddrBookSelectSvcSelectAttrs (maxU16-2)
#define AddrBookSelectSvcAllAttrs   (maxU16-3)

If the client wants all attributes (either all entry attributes or all service attributes), the address book
should return the attributes in some well-known order. The next batch of #define's specifies the order for
the common fields

#define AddrBookSurNameIndex 0
#define AddrBookGivenNameIndex 1
#define AddrBookHomePhoneIndex 2
#define AddrBookBussPhoneIndex 3
#define AddrBookCountryIndex 4
#define AddrBookStateIndex 5
#define AddrBookZipIndex 6
#define AddrBookCityIndex 7
#define AddrBookDistrictIndex 8
#define AddrBookStreetIndex 9
#define AddrBookCompanyIndex 10
#define AddrBookTitleIndex 11
#define AddrBookPositionIndex 12
#define AddrBookNicknameIndex 13
#define AddrBookBussPhone2Index 14
#define AddrBookFaxIndex 15
#define AddrBookSvcNameIndex 0
#define AddrBookSvcNoteIndex 1
#define AddrBookSvcShortIndex 2

typedef P_UNKNOWN ADDR_BOOK_SERVICE_ID,  *P_ADDR_BOOK_SERVICE_ID;
typedef TAG ADDR_BOOK_ATTR_ID,       *P_ADDR_BOOK_ATTR_ID;
typedef TAG ADDR_BOOK_ATTR_TYPE,      *P_ADDR_BOOK_ATTR_TYPE;
typedef U16 ADDR_BOOK_ATTR_LENGTH,    *P_ADDR_BOOK_ATTR_LENGTH;
typedef P_UNKNOWN ADDR_BOOK_ATTR_VALUE, *P_ADDR_BOOK_ATTR_VALUE;
typedef P_UNKNOWN ADDR_BOOK_KEY,       *P_ADDR_BOOK_KEY;
typedef CHAR    ADDR_BOOK_ATTR_LABEL[nameBufLength];

ADDR_BOOK_ATTR.length is the length of ADDR_BOOK_ATTR.value. The following table lists what the
length field mean, given a certain attribute type:

<table>
<thead>
<tr>
<th>Attr Type</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>abString</td>
<td>length of the string</td>
</tr>
<tr>
<td>abNumber</td>
<td>SizeOf(U32)</td>
</tr>
<tr>
<td>abPhoneNumber</td>
<td>SizeOf(DIALENV_TELEPHONE_NUMBER)</td>
</tr>
<tr>
<td>abOther</td>
<td>length of attribute in bytes</td>
</tr>
</tbody>
</table>
The following table lists what the value field should be, given a certain attribute type:

<table>
<thead>
<tr>
<th>Attr Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>abString</td>
<td>a ptr to actual storage of the str</td>
</tr>
<tr>
<td>abNumber</td>
<td>the number itself</td>
</tr>
<tr>
<td>abPhoneNumber</td>
<td>P_DIALENV_TELEPHONE_NUMBER</td>
</tr>
<tr>
<td>abOther</td>
<td>a ptr to a byte array that contains the attribute.</td>
</tr>
</tbody>
</table>

abString  a ptr to actual storage of the str
abNumber   the number itself
abPhoneNumber  P_DIALENV_TELEPHONE_NUMBER
abOther a ptr to a byte array that contains the attribute.

typedef struct ADDR_BOOK_ATTR {
    ADDR_BOOK_ATTR_ID id;
    ADDR_BOOK_ATTR_TYPE type;
    ADDR_BOOK_ATTR_LENGTH length; // length of value, in bytes
    ADDR_BOOK_ATTR_VALUE value;
    ADDR_BOOK_ATTR_LABEL label; // for display purpose
} ADDR_BOOK_ATTR, *P_ADDR_BOOK_ATTR;

typedef struct ADDR_BOOK_ATTR_DESC {
    ADDR_BOOK_ATTR_ID id;
    ADDR_BOOK_ATTR_TYPE type;
    ADDR_BOOK_ATTR_LABEL label; // for display purpose
} ADDR_BOOK_ATTR_DESC, *P_ADDR_BOOK_ATTR_DESC;

typedef struct ADDR_BOOK_SERVICE {
    ADDR_BOOK_SERVICE_ID svcId; // uniquely identify a svc inst
    U16 numAttrs;
    P_ADDR_BOOK_ATTR attrs;
} ADDR_BOOK_SERVICE, *P_ADDR_BOOK_SERVICE;

Enum16 (ADDR_BOOK_ENTRY_TYPE) {
    abIndividual = 0,
    abGroup = 1,
};

#define abMaxSvcNameMatch 5

typedef struct ADDR_BOOK_SERVICE_QUAL {
    U16 numAttrIds;
    P_ADDR_BOOK_ATTR_ID svcAttrIds;
    U16 numSvcNames;
    CHAR svcNames[abMaxSvcNameMatch][nameBufLength];
} ADDR_BOOK_SERVICE_QUAL, *P_ADDR_BOOK_SERVICE_QUAL;

Heap field is an in-parameter in msgAddrBookGet and msgAddrBookSearch, it is not applicable for other msgs. A client should specify the heap id of the heap that it would like space allocated. Typically a client would use OSTaskSharedHeapId(clientsTaskId). A client should not use osProcessSharedHeapId or osProcessHeapId because they refer to different heaps in different processes. It is very important that clients free allocated space.

typedef struct ADDR_BOOK_ENTRY {
    OS_HEAP_ID heap; // where should the address
    ADDR_BOOK_ENTRY_TYPE type; // book alloc necessary space
    ADDR_BOOK_KEY key;
    U16 numAttrs;
    P_ADDR_BOOK_ATTR attrs;
    U16 numServices; // applicable only for
    P_ADDR_BOOK_SERVICE services; // msgAddrBookGet and
    ADDR_BOOK_SERVICE_QUAL svcQual; // msgAddrBookSearch
    // abIndividual only
    // abIndividual only
    // service qualifier, for Get
} ADDR_BOOK_ENTRY, *P_ADDR_BOOK_ENTRY;
Status Codes

Error Status Values

```c
#define stsAddrBookBufTooSmall MakeStatus(clsAddressBook, 1)
#define stsAddrBookEntryExists MakeStatus(clsAddressBook, 2)
#define stsAddrBookSvcDataExists MakeStatus(clsAddressBook, 3)
#define stsAddrBookEntryNotFound MakeStatus(clsAddressBook, 4)
#define stsAddrBookSvcNotFound MakeStatus(clsAddressBook, 5)
#define stsAddrBookBadKey MakeStatus(clsAddressBook, 6)
#define stsAddrBookUnknownType MakeStatus(clsAddressBook, 7)
#define stsAddrBookInvalidAttr MakeStatus(clsAddressBook, 8)
#define stsAddrBookReadOnlyAttr MakeStatus(clsAddressBook, 9)
#define stsAddrBookDuplicateAttr MakeStatus(clsAddressBook, 10)
```

Non Error Status Values

```c
#define stsAddrBookGroupEntry MakeWarning(clsAddressBook, 7)
#define stsAddrBookNotSupported MakeWarning(clsAddressBook, 8)
```

Messages

msgAddrBookGet

fills in the specified entry field data, given an address book key for the entry.

Takes `P_ADDR_BOOK_ENTRY`, returns `STATUS`.

```c
#define msgAddrBookGet MakeMsg(clsAddressBook, 1)
```

```c
typedef struct ADDR_BOOK_ENTRY {
    OS_HEAP_ID heap;         // where should the address
    ADDR_BOOK_TYPE type;     // book alloc necessary space
    ADDR_BOOK_KEY key;       // applicable only for
    U16 numAttrs;            // msgAddrBookGet and
    P_ADDR_BOOK_ATTR attrs;  // msgAddrBookSearch
    U16 numServices;         // Read only, abIndividual only
    P_ADDR_BOOK_SERVICE services;    // abIndividual only
    ADDR_BOOK_SERVICE_QUAL svcQual;        // service qualifier, for Get
} ADDR_BOOK_ENTRY, *P_ADDR_BOOK_ENTRY;
```

If attribute type is `abString` and the client-provided space is not big enough, `stsAddrBookBufTooSmall` is returned, and as much information as there is room for is filled in (null-terminated). Similarly, if attribute type is `abOther`, `stsAddrBookBufTooSmall` is returned, and the client-provided buffer is filled in (w/o null-termination).

Parameters:

- `pArgs->key` In: specify from which entry to get info
- `pArgs->type` Out: type of the entry
- `pArgs->numAttrs` In: number of elements in `pArgs->attrs` array. Each of `pArgs->attrs.id` specifies the id of the attribute the client wants the address book to return. If the client sets this field to AddrBookAll, then the address book will return all entry attributes (excluding services), and it will allocate the necessary space. The client needs to deallocate the space. If the field is set to 0, then no attributes are returned. Out: number of attributes returned
pArgs->attrs[x].id In: which attributes to get
pArgs->attrs[x].type Out: attribute type
pArgs->attrs[x].length Out: attribute length of each attr specified in entryAttrIds. See previous table on attribute type-attribute length.
pArgs->attrs[x].value In: if this field is pNull, the address book will allocate space for the value. Out: attribute value. see previous table on attribute value-attribute length.
pArgs->attrs[x].label Out: attribute label, for display.
pArgs->numServices In: number of elements in pArgs->services array The client should specify AddrBookAll here if it wants all services and all service attributes for each service. If it wants only selective attributes from all services, then set numServices to AddrBookAllSvcSelectAttr. If it wants all attributes from selective services, then set numServices to AddrBookSelectSvcAllAttr. Lastly, if the client wants selective attrs from selective svc's, then set numServices to AddrBookSelectSvcSelectAttr. In all cases, the address book will allocate the necessary storage for all info, which needs to be freed by the client. If the field is set to 0, then no service information is returned Out: number of services returned.
pArgs->svcQual In: If numServices is AddrBookAllSvcSelectAttr, or AddrBookSelectSvcSelectAttr, then numAttrIds is the number of elements in the svcAttrIds array, and svcAttrIds contains the ids of the attributes whose values should be retrieved. If numServices is AddrBookSelectSvcAllAttr or AddrBookSelectSvcSelectAttr, then numSvcNames is the number of elements in the svcNames array, and svcNames contains the names of services whose attribute values should be retrieved. For any other values of numServices, this field is irrelevant.
pArgs->services Out: Allocated space if so requested.
pArgs->services[y].svcld In: For each services specifically requested (as opposed to using AddrBookAll or AddrBookAllSvcSelectAttr, and other such constants in pArgs->numServices), there needs to be a svcld, telling the address book which service to return
pArgs->services[y].attrs: In/Out: analogous to pArgs->attrs

msgAddrBookSet
Sets the specified entry and service data.
Takes P_ADDR_BOOK_ENTRY, returns STATUS.

#define msgAddrBookSet MakeMsg(clsAddressBook, 2)

typedef struct ADDR_BOOK_ENTRY {
  OS_HEAP_ID heap;
  ADDR_BOOK_ENTRY_TYPE type;
  ADDR_BOOK_KEY key;
  U16 numAttrs;
  P_ADDR_BOOK_ATTR attrs;
  U16 numServices;
  P_ADDR_BOOK_SERVICE services;
  ADDR_BOOK_SERVICE_QUAL svcQual;
} ADDR_BOOK_ENTRY, *P_ADDR_BOOK_ENTRY;
Parameters:

pArgs->key  In: specify from which entry to get info
pArgs->numAttrs  In: how many attributes in the entry to set
pArgs->attr[x].id  In: which attributes to set
pArgs->attr[x].type  NA: don't need to specify
pArgs->attr[x].length  In: client-specified size of the corresponding entryAttrValue field. mandatory for abOther, unnecessary for other types.
pArgs->attr[x].value  In: attribute value. see previous table on attribute value-attribute length.
pArgs->numServices  In: number of services to set. Set it to 0 if not setting any service info
pArgs->svcAttrIds  NA: not applicable
pArgs->services[y].svcId  In: service id of the service that set applies to
pArgs->services[y].attrs  In: analogous to pArgs->attrs.

msgAddrBookAdd

Adds the specified entry and service data.

Takes P_ADDR_BOOK_ENTRY, returns STATUS.

```c
typedef struct ADDR_BOOK_ENTRY {
    OS_HEAP_ID heap;  // where should the address
    ADDR_BOOK_ENTRY_TYPE type;  // book alloc necessary space
    ADDR_BOOK_KEY key;  // applicable only for
    UI16 numAttrs;  // msgAddrBookGet and
    P_ADDR_BOOK_ATTR attrs;  // msgAddrBookSearch
    UI16 numServices;  // Read only, abIndividual only
    P_ADDR_BOOKSERVICE services;  // abIndividual only
    ADDR_BOOKSERVICE_QUAL svcQual;  // service qualifier, for Get
} ADDR_BOOK_ENTRY, *P_ADDR_BOOK_ENTRY;
```

Parameters:

pArgs->key  In: If the msg is used to add a service addr then the client specifies the entry key of the entry to which we add the service address. Out: if the msg is used to add an entry, then address book fill this field w/ the key of the entry just added

pArgs->numAttrs  In: how many attributes in the entry to have specified initial values.

pArgs->attr[x].id  In: which attributes to add. To add a brand new individual entry, then at least AddrBookGivenNameId or AddrBookSurNameId need to be specified. To add a group entry, AddrBookGroupNameId needs to be specified.

pArgs->attr[x].type  NA: don't need to specify

pArgs->attr[x].length  In: mandatory if attribute type is abOther

pArgs->attr[x].value  In: attribute value. see previous table on attribute value-attribute length.

pArgs->numServices  In: number of services to set. Set it to 0 if not adding any service info

pArgs->svcAttrIds  NA: not applicable
**msgAddrBookDelete**

Deletes the specified entry and service data.

Takes P_ADDR_BOOK_ENTRY, returns STATUS.

```c
#define msgAddrBookDelete MakeMsg(clsAddressBook, 4)
```

**Message Arguments**

```c
typedef struct ADDR_BOOK_ENTRY {
    OS_HEAP_ID          heap;        // where should the address book alloc necessary space
    ADDR_BOOK_ENTRY_TYPE type;       // applicable only for msgAddrBookGet and msgAddrBookSearch
    ADDR_BOOK_KEY       key;         // msgAddrBookGet and msgAddrBookSearch
    U16                 numAttrs;
    P_ADDR_BOOK_ATTR    attrs;       // msgAddrBookGet and msgAddrBookSearch
    U16                 numServices;  // Read only, abIndividual only
    P_ADDR_BOOK_SERVICE services;    // abIndividual only
    ADDR_BOOK_SERVICE_QUAL svcQual;   // service qualifier, for Get
} ADDR_BOOK_ENTRY, *P_ADDR_BOOK_ENTRY;
```

**Parameters:**

- **pArgs->key**  In: entry id of the entry to be deleted. If deleting a service, then this field still needs to be specified. Only the specified service is deleted.
- **pArgs->numServices**  In: number of services to delete. Set it to 0 if deleting the entire entry.
- **pArgs->services[x].svclid**  In: Id's of the services to be deleted.

All other fields in ADDR_BOOK_ENTRY structure are not applicable.

---

**msgAddrBookSearch**

Searches for the entry that matches the search spec.

Takes P_ADDR_BOOK_SEARCH, returns STATUS.

```c
#define msgAddrBookSearch MakeMsg(clsAddressBook, 5)
```

**Arguments**

```c
Enum16(ADDR_BOOK_SEARCH_TYPE) {
    abSearchIndividuals = 0,    // Enumerate address book entries
    abSearchGroups = 1,         // Enumerate groups
    abSearchAll = 2,            // Enumerate all entries
};
Enum16(ADDR_BOOK_SEARCH_DIR) {
    abEnumNext = 0,             // Search forward
    abEnumPrevious = 1         // Search backwards
};
Enum16(ADDR_BOOK_ATTR_OPS) {
    abAnd = 0,
    abOr = 1
};
```
Enum16(ADDR_BOOK_VALUE_OPS) {
    abEqual = 0,
    abNotEqual = 1,
    abGreater = 2,
    abGreaterEqual = 3,
    abLessEqual = 4,
    abMatchBeginning = 6,    // string matching
    abMatchEnd = 7,         // string matching
    abMatchPartial = 8,    // string matching
    abMaxValue = abMatchPartial
};

If a client wants to specify a query that says "match an entry whose last name is "Smith" and whose zip code is "94024", then the .query field in pArgs for msgAddrBookSearch would have 2 elements:

<table>
<thead>
<tr>
<th>pArgs</th>
<th>query</th>
<th>id</th>
<th>length</th>
<th>value</th>
<th>valueOp</th>
<th>attrOp</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr[0]</td>
<td>AddrBookGivenNameId</td>
<td>N/A</td>
<td>Smith</td>
<td>abEqual</td>
<td>abAnd</td>
<td></td>
</tr>
<tr>
<td>attr[1]</td>
<td>AddrBookZipId</td>
<td>N/A</td>
<td>94024</td>
<td>abEqual</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

Essentially, the attrOp field specifies the operator between attr[x] and attr[x+1]. valueOp specifies the relationship between the attribute id and its specified value. e.g. (a == 1) AND (b == 2), the "=="'s are valueOp, "AND" is an attrOp. By definition, pArgs->attrs[pArgs->numAttrs-1].attrOp does not need to be specified.

typedef struct ADDR_BOOK_QUERY_ATTR {
    ADDR_BOOK_ATTR_ID id;
    ADDR_BOOK_ATTR_LENGTH length;
    ADDR_BOOK_VALUE_OPS valueOp;
    ADDR_BOOK_ATTR_VALUE value;
    ADDR_BOOK_ATTR_OPS attrOp;
} ADDR_BOOK_QUERY_ATTR, *P_ADDR_BOOK_QUERY_ATTR;

typedef struct ADDR_BOOK_QUERY {
    U16 numAttrs;
    P_ADDR_BOOK_QUERY_ATTR attrs;
} ADDR_BOOK_QUERY, *P_ADDR_BOOK_QUERY;

typedef struct ADDR_BOOK_SEARCH {
    ADDR_BOOK_KEY key;    // In: Starting Pt. Out: Result
    ADDR_BOOK_SEARCH_TYPE type;  // In:
    U32 nth;           // In: look for the nth entry meeting
                         // the search criteria. nth = 1
                         // if looking for the first entry
                         // meeting the search criteria.
    ADDR_BOOK_ATTR_ID sort;
    ADDR_BOOK_SEARCH_DIR dir;
    ADDR_BOOK_ENTRY_TYPE outType;
    ADDR_BOOK_QUERY query;  // In: what to look for, set query to
                             // pNull to enumerate
    ADDR_BOOK_ENTRY result; // Out: result entry
} ADDR_BOOK_SEARCH, *P_ADDR_BOOK_SEARCH;

pArgs->key is the pArgs->nth entry that matches the search spec, sorted by the attribute specified in pArgs->sort, the entry is just before/after(depending on the value of pArgs->dir) of pArgs->key. If key is nil, the enumeration starts with the first element if abEnumNext is specified, and the last element if abEnumPrevious is specified.

Parameters:

pArgs->key In Start point of the search Out:Resulting entry id of the match
pArgs->nth In Look for the nth entry meeting the search criteria
pArgs->sort In Attribute id of the attribute that the result should be sorted by
pArgs->dir  In  search backwards or forwards.
pArgs->outType  Out: type of the matched entry
pArgs->query  In  an elaborate explanation is available below
pArgs->result  In  How each field is specified is the same as that for msgAddrBookGet. Except for the
   key field, which will be filled in by msgAddrBookSearch Out: same as msgAddrBookGet

msgAddrBookGetServiceDesc

Gets the service address description from the address book.

Takes P_ADDR_BOOK_SERVICES, returns STATUS.

#define msgAddrBookGetServiceDesc MakeMsg(clsAddressBook, 9)
#define abServiceDescFields
   CHAR name[nameBufLength];  
   U16 maxPerEntry;  
   U16 numAttrs;  
   P_ADDR_BOOK_ATTR_DESC attrs;

Arguments
typedef struct ADDR_BOOK_SVC_DESC {
   abServiceDescFields
} ADDR_BOOK_SVC_DESC, *P_ADDR_BOOK_SVC_DESC;
typedef struct ADDR_BOOK_SERVICES {
   OS_HEAP_ID  heap;  
   U16  numServices;  
   P_ADDR_BOOK_SVC_DESC  services;
} ADDR_BOOK_SERVICES, *P_ADDR_BOOK_SERVICES;

Comments
Parameters:
pArgs->numServices  Out: number of installed services an array of ADDR_BOOK_SVC_DESC's is
   allocated and should be freed by the caller.

Return Value  stsOK

msgAddrBookEnumGroupMembers

Enumerates through the members in a group.

Takes P_ADDR_BOOK_ENUM_GROUP_MEMBER, returns STATUS.

#define msgAddrBookEnumGroupMembers MakeMsg(clsAddressBook, 6)

Arguments
typedef struct ADDR_BOOK_ENUM_GROUP_MEMBER {
   ADDR_BOOK_KEY  groupKey;
   ADDR_BOOK_KEY  startKey;
   BOOLEAN  recurse;
   ADDR_BOOK_ATTR_ID  sort;
   U32  count;
   P_ADDR_BOOK_KEY  pKeys;
} ADDR_BOOK_ENUM_GROUP_MEMBER, *P_ADDR_BOOK_ENUM_GROUP_MEMBER;

Comments
Parameters:
pArgs->groupKey  In: key of the group
pArgs->startKey  In: where to start the group enumeration. Use pNull to start from the beginning.
   Out: last entry key returned in pArgs->pKeys. Client usually uses the out value to be the next in
   value of the next msgAddrBookEnumGroupMembers call.
pArgs->recurse  In: whether to recursively enumerate groups
pArgs->sort  In: attr id of the field to sort the returned entry id by
pArgs->count  In: number of entries to return, which is also the number of slots in the pKeys array. Use AddrBookAll to get every member. In this case address book will allocate the necessary space, and the client should free the space. Out: number of entries actually returned
pArgs->pKeys  Out: keys of the members of pArgs->groupKey

msgAddrBookIsAMemberOf

Determines if an entry is a member of a group.
Takes P_ADDR_BOOK_IS_A_MEMBER_OF, returns STATUS.
#define msgAddrBookIsAMemberOf MakeMsg(clsAddressBook, 7)
Arguments
typedef struct ADDR_BOOK_IS_A_MEMBER_OF {
    ADDR_BOOK_KEY groupKey;
    ADDR_BOOK_KEY memberKey;
    BOOLEAN recurse;
} ADDR_BOOK_IS_A_MEMBER_OF, *P_ADDR_BOOK_IS_A_MEMBER_OF;
Comments
Parameters:
pArgs->groupKey  In: key of the group
pArgs->memberKey  In: potential member's key
pArgs->recurse  In: whether to recursively test for membership
Return Value
stsOK  if pArgs->memberKey is a member of pArgs->groupKey.
stsNoMatch  if pArgs->memberKey is not a member of pArgs->groupKey

msgAddrBookGetMetrics

Passes back the metrics for the address book.
Takes P_ADDR_BOOK_METRICS, returns STATUS.
#define msgAddrBookGetMetrics MakeMsg(clsAddressBook, 8)
Arguments
typedef struct ADDR_BOOK_METRICS {
    U32 numEntries;  // Total number of entries
    U32 numGroups;  // Number of groups in the address book
    U16 numServices;  // Number of known services
    U32 spare1;
    U32 spare2;
} ADDR_BOOK_METRICS, *P_ADDR_BOOK_METRICS;

msgAddrBookAddAttr

Adds a new attribute to active address books.
Takes P_ADDR_BOOK_ATTR, returns STATUS.
#define msgAddrBookAddAttr MakeMsg(clsAddressBook, 12)
Arguments
typedef struct ADDR_BOOK_ATTR {
    ADDR_BOOK_ATTR_ID id;
    ADDR_BOOK_ATTR_TYPE type;
    ADDR_BOOK_ATTR_LENGTH length;  // length of value, in bytes
    ADDR_BOOK_ATTR_VALUE value;
    ADDR_BOOK_ATTR_LABEL label;  // for display purpose
} ADDR_BOOK_ATTR, *P_ADDR_BOOK_ATTR;
This operation will change the address book database schema. If the attribute is of type \texttt{abNumber}, the value is initialized to be 0 for all existing address book entries. If the attribute is of type \texttt{abPhoneNumber}, then the value is initialized to be 0. If the attribute is of type \texttt{abString} or \texttt{abOther}, the value is initialized to be 0 length byte array.

After an attribute is added to an address book, clients can then set the attribute value in subsequent \texttt{msgAddrBookSet}'s and get the attribute value in the subsequent \texttt{msgAddrBookGet}'s. Failure to first make an attribute known to an address book and then try to set or get the attribute value will cause \texttt{stsAddrBookInvalidAttr} to be returned.

Parameters:

\begin{itemize}
\item \texttt{pArgs->id} In: the id (should be a tag) of the new attribute. It has to be different from all other attribute ids in the same address book.
\item \texttt{pArgs->type} In: one of \texttt{abNumber}, \texttt{abString}, \texttt{abOther}, \texttt{abPhoneNumber}
\item \texttt{pArgs->label} In: a string, for display purpose. The address book will copy the string to its own storage.
\end{itemize}

Return Value

\texttt{stsRequestNotSupported} if the address book does not allow dynamically changing its database schema.

\texttt{stsAddrBookDuplicateAttrId} There is another attribute in the address book with the same id.

### msgAddrBookCount

Finds the number of entries that match the search spec

Takes \texttt{P_ADDR_BOOK_COUNT}, returns \texttt{STATUS}.

```
#define msgAddrBookCount

typedef struct ADDR_BOOK_COUNT {
    ADDR_BOOK_KEY key;
    ADDR_BOOK_ATTR_ID sort;
    ADDR_BOOK_SEARCH_DIR dir;
    ADDR_BOOK_QUERY query;
    U16 count;
} ADDR_BOOK_COUNT, *P_ADDR_BOOK_COUNT;
```

Arguments

```
Arguments

typedef struct ADDR_BOOK_COUNT {
    ADDR_BOOK_KEY key;
    ADDR_BOOK_ATTR_ID sort;
    ADDR_BOOK_SEARCH_DIR dir;
    ADDR_BOOK_QUERY query;
    U16 count;
} ADDR_BOOK_COUNT, *P_ADDR_BOOK_COUNT;
```

Comments

Parameters:

\begin{itemize}
\item \texttt{pArgs->key} In where to stop counting, AddrBookAll to count the entire database
\item \texttt{pArgs->dir} In whether to start counting from the beginning or the end of the address book.
\item \texttt{pArgs->query} In qualifier. See \texttt{msgAddrBookSearch}
\end{itemize}

### Observer Messages

#### msgAddrBookEntryChanged

Sent to observers when an entry has been changed, added or deleted.

Takes \texttt{P_ADDR_BOOK_ENTRY_CHANGE}, returns \texttt{STATUS}.

```
#define msgAddrBookEntryChanged

MakeMsg(clsAddressBook, 11)
```
ADDRBOOK.H
Observer Messages

Enum16(ADDR_BOOK_CHANGE_TYPE) {
   abServiceChanged = 0,
   abServiceDeleted = 1,
   abServiceAdded = 2,
   abEntryAdded = 3,
   abEntryDeleted = 4,
   abEntryNameChanged = 5,
   abEntryChanged = 6,
   abServiceInstalled = 7, // svcs have been installed
   abServiceDeinstalled = 8, // svcs have been deinstalled
};

typedef struct ADDR_BOOK_ENTRY_CHANGE {
   OBJECT addrBook; // Address book UID
   ADDR_BOOK_CHANGE_TYPE type; // Type of change
   ADDR_BOOK_KEY entryKey; // Internal address book key of the
   // changed entry
   ADDR_BOOK_SERVICE_ID svcId; // service id, if applicable
} ADDR_BOOK_ENTRY_CHANGE, *P_ADDR_BOOK_ENTRY_CHANGE;

Comments
If pArgs->type is abServiceChanged, abServiceDeleted, abServiceAdded, then the address book fills in pArgs->svcId to be the id of the service address affected. pArgs->entryKey is filled in by the address book except when pArgs->type is abServiceInstalled or abServiceDeinstalled. In that case, the address book is notifying clients that some service has been installed or deinstalled, and the service information returned by the previous msgAddrBookGetServiceDesc is no longer up-to-date.
This file contains the API for clsATP.
clsATP inherits from clsObject.
Provides remote access to stations using the AppleTalk protocol suite.

```c
#ifndef ATALK_INCLUDED
#define ATALK_INCLUDED

/* Common #defines and typedefs */

typedef U8 DDP_TYPE, *P_DDP_TYPE;
typedef U8 ATP_FLAGS;
typedef struct ATP_ADDRESS {
  U16 network;
  U8 node;
  U8 socket;
} ATP_ADDRESS, *P_ATP_ADDRESS;
typedef struct USER_BYTES {
  U8 ub1;
  U8 ub2;
  U8 ub3;
  U8 ub4;
} USER_BYTES, *P_USER_BYTES;
typedef struct ATP_OPTIONS {
  DDP_TYPE ddpType;
  ATP_FLAGS flags;
  U16 transactionID; // In: transaction id when sending a response
  U32 interval; // Out: transaction id when receiving a request
  U16 retries; // timeout value in milliseconds
  U8 numUserByteSets; // In: number of valid user byte sets to send
  U8 reserved; // Out: number of valid user byte sets received
  USER_BYTES userBytes[8];
} ATP_OPTIONS, *P_ATP_OPTIONS;

#define ATP_XO_Flag 0x01
#define ATP_Checksum_Flag 0x02
#define ATP_ALONoResponse_Flag 0x04

typedef U8 NBP_NAME, *P_NBP_NAME;
#define NBP_NAME_Size 99

Format for an NBP name is:

U8 objectNameLength;
U8 objectName[objectNameLength];
U8 typeNameLength;
U8 typeName[typeNameLength];
U8 zoneNameLength;
U8 zoneName[zoneNameLength];
typedef U8 NBP_ENUMERATOR;
```
typedef struct NBP_TUPLE {
    ATP_ADDRESS address;
    NBP_ENUMERATOR enumerator;
    NBP_NAME name[ NBP_NAME_Size ];
} NBP_TUPLE, * P_NBP_TUPLE;

typedef U8 ZONES_BUFFER, * P_ZONES_BUFFER;

Messages

msgNBPRegister
Registers a name with the network.

Takes P_NBPREGISTER, returns STATUS.

#define msgNBPRegister MakeMsg( clsATP, 1 )

Arguments
typedef struct NBPREGISTER {
    P_NBP_NAME pName; // name to register
} NBPREGISTER, * P_NBPREGISTER;

msgNBPRemove
Removes a previously registered name from the network.

Takes P_NBPREMOVE, returns STATUS.

#define msgNBPRemove MakeMsg( clsATP, 2 )

Arguments
typedef struct NBPREMOVE {
    P_NBP_NAME pName; // name to remove
} NBPREMOVE, * P_NBPREMOVE;

msgNBPLookup
Looks up names registered with the network.

Takes P_NBPLOOKUP, returns STATUS.

#define msgNBPLookup MakeMsg( clsATP, 3 )

Arguments
typedef struct NBPLOOKUP {
    P_NBP_NAME pName; // name spec to lookup
    P_TP_BUFFER pBuffer; // ptr to buffer containing names found
    UI6 length; // size of buffer in bytes
    UI6 numMatches; // In-Out: number of names wanted/found
} NBPLOOKUP, * P_NBPLOOKUP;

msgNBPConfirm
Confirms the network address of a registered name.

Takes P_NBP_CONFIRM, returns STATUS.

#define msgNBPConfirm MakeMsg( clsATP, 4 )

Arguments
typedef struct NBP_CONFIRM {
    P_NBP_NAME pName; // name to confirm address of
    P_TP_ADDRESS pAddress; // ptr to address of name
} NBP_CONFIRM, * P_NBP_CONFIRM;
msgZIPGetZoneList
Obtains a list of zone names.
Takes P_ZIP_GETZONES, returns STATUS.

#define msgZIPGetZoneList MakeMsg(clsATP, 6)

typedef struct ZIP_GETZONES {
  P_ZONES_BUFFER pBuffer;    // ptr to buffer to contain zone names
  U16 length;                // size of buffer in bytes
  U16 numZones;              // Out: number of zones found
} ZIP_GETZONES, *P_ZIP_GETZONES;

msgZIPGetMyZone
Obtains my zone name.
Takes P_ZIP_GETZONES, returns STATUS.

#define msgZIPGetMyZone MakeMsg(clsATP, 7)

typedef struct ZIP_GETZONES {
  P_ZONES_BUFFER pBuffer;    // ptr to buffer to contain zone names
  U16 length;                // size of buffer in bytes
  U16 numZones;              // Out: number of zones found
} ZIP_GETZONES, *P_ZIP_GETZONES;

msgATPRespPktSize
Sets the maximum size of ATP response packets.
Takes P_ATP_RESPPKTSIZE, returns STATUS.

#define msgATPRespPktSize MakeMsg(clsATP, 8)

typedef struct ATP_RESPPKTSIZE {
  U16 size;                  // max size of response packets in bytes
} ATP_RESPPKTSIZE, *P_ATP_RESPPKTSIZE;
This file contains the API definition for the interface between the connections notebook and a generic service.

The connections notebook is, effectively, an option sheet. Because of this implementation choice, it is important to understand the option sheet protocol and messages, as defined in OPTION.H. The terminology chosen herein reflects the close association between the connections notebook and an option sheet.

The two default views that one gets, for disks and printers, in the connections notebook are each option sheets added as cards of the connections notebook option sheet. Other sheets or windows can be added to the connections notebook.

The connections notebook observes the well-known list theConnections. If an item is added to the list, the connections notebook calls that item with msgConnectionsAddSheet, with the P_ARGS being the main option sheet in the connections notebook. By using msgOptionAddCard to the object passed in the aforementioned call, a service can add a sheet or just a single window to the connections notebook. Once these items have been added, all responsibility for the user interface and functionality rests solely on the service.

Network disks and printers, however, are handled differently. There are already predefined windows for these two items. A network file-sharing system, for example, would add itself to the well-known list theVolumeServices. The connections notebook, which observes this list, would send the object on the list a msgConnectionsStartConversation and a msgConnectionsSetConnectionsApp to pass along the application context of the connections notebook from this time.

If the network file-sharing service were to remove itself from theVolumeServices, the connections notebook would send msgConnectionsEndConversation to the object.

The object on the list is expected to be able to respond to the various connections messages. If it has specified that it provides a UI, it will be asked for its network view when appropriate.

```
#ifndef CNCTIONS_INCLUDED
#define CNCTIONS_INCLUDED
#endif

#include <instlmgr.h>

Common #defines and typedefs

Warnings

#define stsConnectionsAlreadyConnected MakeWarning(clsConnections, 1)

Statuses

#define stsConnectionsPasswordFailed MakeStatus(clsConnections, 1)
#define stsConnectionsServiceDeinstalling MakeStatus(clsConnections, 2)
#define stsConnectionsNotConnected MakeStatus(clsConnections, 3)
```
**Typedefs**

```c
typedef struct CONNECTIONS_MENU_ITEM {
    P_CHAR pName;
    OBJECT netService;
    P_UNKNOWN netIdentifier;
    U32 reserved[2];
} CONNECTIONS_MENU_ITEM, *P_CONNECTIONS_MENU_ITEM;

typedef struct CONNECTIONS_ITEM {
    struct CONNECTIONS_ITEM *pNextConnectionsItem; // Next item
    P_UNKNOWN pItemID; // Service defined identifier
    TAG itemIconTag; // Item's icon tag
    TAG itemName; // Item tag
    P_CHAR name; // Item name
    P_CHAR serverName; // Item's server's name
    P_CHAR location; // Item's location
    P_CHAR type; // Item's type
    BOOLEAN connected; // Connected?
    BOOLEAN autoConnect; // Auto-connect enabled?
    BOOLEAN remember; // Remember (menu) enabled?
    // fill in some more information here
    P_UNKNOWN itemSpecificData; // volume or printer stuff
    U32 filler[4]; // reserved
} CONNECTIONS_ITEM, *P_CONNECTIONS_ITEM, **PP_CONNECTIONS_ITEM;
```

**Messages**

**msgConnectionsSetState:**

Sets the specified states in the service.

Takes P_CONNECTIONS_STATE, returns STATUS.

**Arguments**

```c
Enum16 ( CONNECTIONS_CONNECT_STATE ) {
    cnctManualConnections, // Connect only when asked to
    cnctAutoConnections, // Connect auto-connect items
    cnctPromiscuousConnections // Connect to everything
};

Enum16 ( CONNECTIONS_WARNINGS ) {
    cnctWarningNone = 0, // No warnings
    cnctWarningPermissionsFailure = flag0, // On permissions failure
    cnctWarningOnConnection = flag1, // On connection
    cnctWarningOnUnconnection = flag2 // On loss of connection
};

Enum16 ( CONNECTIONS_PASSWORDS ) {
    cnctPasswordNone = 0, // Do not save passwords
    cnctPasswordServer = flag0, // Save server passwords
    cnctPasswordItem = flag1, // Save item passwords
    cnctPasswordServerAndItem = flag2 // Save server and item passwords
};

Enum16 ( CONNECTIONS_PERMISSIONS ) {
    cnctPermissionsReadWrite, // Connect Read/Write
    cnctPermissionsReadOnly // Connect Read only
};
```

**typedef struct CONNECTIONS_STATE {
    BOOLEAN attached; // Attached
    BOOLEAN connectMores; // How to attach
    BOOLEAN connectWarning; // Level of warnings
    BOOLEAN connectPasswords; // What passwords
    BOOLEAN connectPermissions; // What permissions
    U32 reserved[4];```

```c
) CONNECTIONS_STATE, *P_CONNECTIONS_STATE;
```

```c
#define msgConnectionsSetState MakeMsg ( clsConnections, 1 )
```
**msgConnectionsGetState:**

Gets the specified states in the service.

Takes `P_CONNECTIONS_STATE`, returns `STATUS`.

```
#define msgConnectionsGetState MakeMsg ( clsConnections, 2 )

typedef struct CONNECTIONS_STATE {
    BOOLEAN attached;       // Attached
    CONNECTIONS_CONNECT_STATE connectMores; // How to attach
    CONNECTIONS_WARNINGS connectWarning; // Level of warnings
    CONNECTIONS_PASSWORDS connectPasswords; // What passwords
    CONNECTIONS_PERMISSIONS connectPermissions; // What permissions
    U32 reserved[4];
} CONNECTIONS_STATE, *P_CONNECTIONS_STATE;
```

**msgConnectionsEnumerateItems:**

Gets a list of the network items, per restrictions.

Takes `P_CONNECTIONS_ENUMERATE`, returns `STATUS`.

```
#define cnctAttribMatchLocation flag0 // Match on location
#define cnctAttribMatchServer flag1 // Match on server
#define cnctAttribMatchConnect flag2 // Match on connected state
#define cnctAttribMatchAutoConnect flag3 // Match on auto-connect state
#define cnctAttribMatchMenu flag4 // Match on menu
                                 // (remember) state

typedef struct ATTRIB {
    U32 flags;                   // various meanings -- complete match
                                  // match at beginning, match at end
                                  // connected, auto connect, remember
    P_CHAR restrictName;         // match this string
                                  // other possible characteristics -- type, characteristics, etc.
    PUNKNOWN matchID;           // restrict enumeration to this file
                                // server
    TAG tag;                    // Tag to match against
} ATTRIB, *P_ATTRIB;

#define cnctFlagLocationsOnly flag0 // Look only at locations
#define cnctFlagServersOnly flag1 // Look only at servers
#define cnctFlagOKFreeCIFields flag14 // Free the CI fields
#define cnctFlagOKFreeCI flag15 // Free the CI

typedef struct CONNECTIONS_ENUMERATE {
    ATTRIB attributes;
    U16 count;       // in = # of entries to return in list.
                      // out = # of valid entries in list.
    U16 next;        // in = 0 to start at beginning
                      // OR previous out value to pick up
                      // where we left off.
    P_CONNECTIONS_ITEM pEntry; // in = pNull.
                                // out = Link list of connections items.
    U16 flags;       // in = state flags to filter on.
                      // out = free state
} CONNECTIONS_ENUMERATE, *P_CONNECTIONS_ENUMERATE;

#define msgConnectionsEnumerateItems MakeMsg ( clsConnections, 3 )
```

**msgConnectionsEnumerateServers:**

Gets a list of the network servers, per restrictions.

Takes `P_CONNECTIONS_ENUMERATE`, returns `STATUS`.

```
#define msgConnectionsEnumerateServers MakeMsg ( clsConnections, 4 )
```
typedef struct CONNECTIONS_ENUMERATE {
    ATTRIB attributes;
    U16 count; // in = # of entries to return in list.
    // out = # of valid entries in list.
    U16 next; // in = 0 to start at beginning
    // OR previous out value to pick up
    // where we left off.
    P_CONNECTIONS_ITEM pEntry; // in = pNull.
    // out = Link list of connections items.
    U16 flags; // in = state flags to filter on.
    // out = free state
} CONNECTIONS_ENUMERATE, * P_CONNECTIONS_ENUMERATE;

Use CONNECTIONS_ITEM with restriction of cnctFlagServersOnly.

msgConnectionsEnumerateTags:

Gets a list of the known tags, per restrictions.

Takes P_CONNECTIONS_ENUMERATE, returns STATUS.

typedef struct CONNECTIONS_TAG {
    TAG tag;
} CONNECTIONS_TAG, * P_CONNECTIONS_TAG;

#define msgConnectionsEnumerateTags MakeMsg ( clsConnections, 5 )

typedef struct CONNECTIONS_ENUMERATE {
    ATTRIB attributes;
    U16 count; // in = # of entries to return in list.
    // out = # of valid entries in list.
    U16 next; // in = 0 to start at beginning
    // OR previous out value to pick up
    // where we left off.
    P_CONNECTIONS_ITEM pEntry; // in = pNull.
    // out = Link list of connections items.
    U16 flags; // in = state flags to filter on.
    // out = free state
} CONNECTIONS_ENUMERATE, * P_CONNECTIONS_ENUMERATE;

msgConnectionsGetNetworkView:

Each service is required to provide a window, which will be a client of a scrollwin, which will be set as the current (active) window when the network view is invoked. This window will be able to make use of msgConnections calls to manipulate attachments, et al.

Takes P_WIN, returns STATUS.

#define msgConnectionsGetNetworkView MakeMsg ( clsConnections, 6 )

msgConnectionsCompareItems:

Compares two pItemID values to see if they refer to the same item.

Takes P_CONNECTIONS_COMPARE, returns STATUS.

typedef struct CONNECTIONS_COMPARE {
    P_UNKNOWN item1; // First item
    P_UNKNOWN item2; // Second item
    BOOLEAN same; // Out: Are they the same?
    U32 forPublicUse; // if any one needs this
} CONNECTIONS_COMPARE, * P_CONNECTIONS_COMPARE;

#define msgConnectionsCompareItems MakeMsg ( clsConnections, 10 )
msgConnectionsTagItem:
Tags the indicated item.
Takes P_CONNECTIONS_TAG_ITEM, returns STATUS.

```c
typedef struct CONNECTIONS_TAG_ITEM {
    TAG tag; // Tag to set
    U32 flags; // Type
    P_UNKNOWN netAddress; // Item's address
    U32 userInformation;
} CONNECTIONS_TAG_ITEM, * P_CONNECTIONS_TAG_ITEM;
#define msgConnectionsTagItem MakeMsg ( clsConnections, 11 )
```

msgConnectionsGetServiceInfo:
Gets the service name and other information.
Takes P_CONNECTIONS_SERVICE_INFO, returns STATUS.

```c
typedef struct CONNECTIONS_SERVICE_INFO {
    CHAR serviceName[nameBufLength]; // Service name
    U16 reserved:15,
    uiProvided:1; // User interface provided
    U32 filler[2];
} CONNECTIONS_SERVICE_INFO, * P_CONNECTIONS_SERVICE_INFO;
#define msgConnectionsGetServiceInfo MakeMsg ( clsConnections, 12 )
```

msgConnectionsGetItemInfo:
Gets information for the specified item, specific to the service.
Takes P_UNKNOWN, returns STATUS.

```c
#define msgConnectionsGetItemInfo MakeMsg ( clsConnections, 13 )
```

msgConnectionsSetConnectionsApp:
Passes the connections notebook app object to the service.
Takes OBJECT, returns STATUS.

```c
#define msgConnectionsSetConnectionsApp MakeMsg ( clsConnections, 14 )
```

msgConnectionsUpdate:
Requests an update of the current network state.
Takes nothing, returns STATUS.

```c
#define msgConnectionsUpdate MakeMsg ( clsConnections, 15 )
```

msgConnectionsExpandCollapse:
Requests an expand/collapse (depending on the argument) of the current view of the network.
Takes BOOLEAN, returns STATUS.

```c
#define msgConnectionsExpandCollapse MakeMsg ( clsConnections, 16 )
```
msgConnectionsConnectItem:
Connect the specified item.
Takes P_CONNECTIONS_REQUEST, returns STATUS.

Arguments
typedef struct CONNECTIONS_REQUEST {
P kostenlos pItemID; // Item to connect
U32 response;
} CONNECTIONS_REQUEST, *P_CONNECTIONS_REQUEST;
#define msgConnectionsConnectItem MakeMsg ( clsConnections, 17 )

msgConnectionsUnconnectItem:
Unconnect the specified item.
Takes P_CONNECTIONS_REQUEST, returns STATUS.

#define msgConnectionsUnconnectItem MakeMsg ( clsConnections, 18 )

msgConnectionsRememberItem:
Remember the specified item.
Takes P_CONNECTIONS_REQUEST, returns STATUS.

#define msgConnectionsRememberItem MakeMsg ( clsConnections, 19 )

msgConnectionsForgetItem:
Forget the specified item.
Takes P_CONNECTIONS_REQUEST, returns STATUS.

#define msgConnectionsForgetItem MakeMsg ( clsConnections, 20 )

msgConnectionsAutoConnectItem:
Sets the auto connect state on for the specified item.
Takes P_CONNECTIONS_REQUEST, returns STATUS.

#define msgConnectionsAutoConnectItem MakeMsg ( clsConnections, 21 )
**msgConnectionsUnAutoConnectItem:**
Sets the auto connect state off for the specified item.

Takes P_CONNECTIONS_REQUEST, returns STATUS.

```c
#define msgConnectionsUnAutoConnectItem MakeMsg ( clsConnections, 22 )
```

**msgConnectionsAddSheet:**
Permits items on the connections to add items to the contents.

Takes OBJECT, returns STATUS.

```c
#define msgConnectionsAddSheet MakeMsg ( clsConnections, 23 )
```

**msgConnectionsAddCards:**
Sent to network views, when they are not the foremost view, to run the option protocol.

Takes P_OPTION_TAG, returns STATUS.

```c
#define msgConnectionsAddCards MakeMsg ( clsConnections, 24 )
```

**msgConnectionsSetSelection:**
Sent by the connections notebook to the appropriate service, informing the service what the currently selected item is.

Takes P_UNKNOWN, returns STATUS.

```c
#define msgConnectionsSetSelection MakeMsg ( clsConnections, 25 )
```

**msgConnectionsGetTopCard:**
Sent by the connections notebook to the appropriate service, inquiring of that service what the appropriate top card is to be.

Takes P_TAG, returns STATUS.

```c
#define msgConnectionsGetTopCard MakeMsg ( clsConnections, 26 )
```

**msgConnectionsStartConversation:**
Sent by the Connections Notebook to the appropriate service, informing that service that the Connections Notebook is planning on conversing with it. This message will be sent at first page turn and at restore (of the Connections Notebook) time.

Takes nothing, returns STATUS.

```c
#define msgConnectionsStartConversation MakeMsg ( clsConnections, 27 )
```
**msgConnectionsEndConversation:**
Sent by the Connections Notebook to the appropriate service, informing that service that the Connections Notebook is stopping conversing with it. This message will be sent at save (of the Connections Notebook) time.

Takes nothing, returns STATUS.

```c
#define msgConnectionsEndConversation MakeMsg(clsConnections, 28)
```

**msgConnectionsIsParent:**
Compares two pItemID values to see if item1 is a parent of item2.

Takes P_CONNECTIONS_COMPARE, returns STATUS.

```c
#define msgConnectionsIsParent MakeMsg(clsConnections, 31)
```

```c
typedef struct CONNECTIONS_COMPARE {
    P_UNKNOWN item1; // First item
    P_UNKNOWN item2; // Second item
    BOOLEAN same; // Out: Are they the same?
    U32 forPublicUse; // if any one needs this
} CONNECTIONS_COMPARE, *P_CONNECTIONS_COMPARE;
```

### Notification Messages

**msgConnectionsConnectedChanged:**
Sent by the appropriate service, indicating when an item has been connected to or unconnected from.

Takes P_CONNECTIONS_NOTIFY, returns STATUS.

```c
#define msgConnectionsConnectedChanged MakeMsg(clsConnections, 7)
```

```c
typedef struct CONNECTIONS_NOTIFY {
    OBJECT manager; // manager that sent notification
    IM_HANDLE handle; // handle to service
    OBJECT service; // service that sent notification
    P_UNKNOWN pItemID; // pointer to affected item
    UI6 reserved:13,
        server:1, // Unused
        uiProvided:1, // Unused
        state:1; // connected or unconnected
    UI6 notifyLength; // Length of notify info which follows
} CONNECTIONS_NOTIFY, *P_CONNECTIONS_NOTIFY;
```

**msgConnectionsAutoConnectChanged:**
Sent by the appropriate service, indicating when an item has had the auto connect state set or turned off for it.

Takes P_CONNECTIONS_NOTIFY, returns STATUS.

```c
#define msgConnectionsAutoConnectChanged MakeMsg(clsConnections, 8)
```

```c
typedef struct CONNECTIONS_NOTIFY {
    OBJECT manager; // manager that sent notification
    IM_HANDLE handle; // handle to service
    OBJECT service; // service that sent notification
    P_UNKNOWN pItemID; // pointer to affected item
    UI6 reserved:13,
        server:1, // Unused
        uiProvided:1, // Unused
        state:1; // connected or unconnected
    UI6 notifyLength; // Length of notify info which follows
} CONNECTIONS_NOTIFY, *P_CONNECTIONS_NOTIFY;
```
msgConnectionsRememberChanged:
Sent by the appropriate service, indicating when an item has had the remember state set or turned off for it.
Takes P_CONNECTIONS_NOTIFY, returns STATUS.

```c
#define msgConnectionsRememberChanged MakeMsg ( clsConnections, 9 )
```

typedef struct CONNECTIONS_NOTIFY {
    OBJECT manager;       // manager that sent notification
    HM_HANDLE handle;     // handle to service
    OBJECT service;       // service that sent notification
    P_UNKNOWN pItemID;    // pointer to affected item
    U16 reserved:13,
        server:1,      // Unused
        uiProvided:1,  // Unused
        state:1;       // connected or unconnected
    U16 notifyLength;     // Length of notify info which follows
} CONNECTIONS_NOTIFY, * P_CONNECTIONS_NOTIFY;

msgConnectionsItemChanged:
Sent by the appropriate service, indicating when an item has been noticed or lost.
Takes P_CONNECTIONS_NOTIFY, returns STATUS.

```c
#define msgConnectionsItemChanged MakeMsg ( clsConnections, 30 )
```

typedef struct CONNECTIONS_NOTIFY {
    OBJECT manager;       // manager that sent notification
    HM_HANDLE handle;     // handle to service
    OBJECT service;       // service that sent notification
    P_UNKNOWN pItemID;    // pointer to affected item
    U16 reserved:13,
        server:1,      // Unused
        uiProvided:1,  // Unused
        state:1;       // connected or unconnected
    U16 notifyLength;     // Length of notify info which follows
} CONNECTIONS_NOTIFY, * P_CONNECTIONS_NOTIFY;

msgConnectionsServiceChanged:
Sent by the appropriate service, indicating when it is available for use or unavailable.
Takes P_CONNECTIONS_NOTIFY, returns STATUS.

```c
#define msgConnectionsServiceChanged MakeMsg ( clsConnections, 32 )
```

typedef struct CONNECTIONS_NOTIFY {
    OBJECT manager;       // manager that sent notification
    HM_HANDLE handle;     // handle to service
    OBJECT service;       // service that sent notification
    P_UNKNOWN pItemID;    // pointer to affected item
    U16 reserved:13,
        server:1,      // Unused
        uiProvided:1,  // Unused
        state:1;       // connected or unconnected
    U16 notifyLength;     // Length of notify info which follows
} CONNECTIONS_NOTIFY, * P_CONNECTIONS_NOTIFY;
This file contains the API for clsDialEnv, clsDialEnvOptCard, and clsDialEnvField.

clsDialEnv inherits from clsService.

clsDialEnv maintains telephone dialing related information pertinent to a specific geographic location/environment.

The intent of clsDialEnv is to relieve client data communication programs of having to replicate the code for maintaining their own separate telephone dialing-related data and logic. clsDialEnv is designed to provide the “intelligence” and data needed for dialing from/to a variety of environments (to/from local in-house to/from international).

clsDialEnvOptCard inherits from clsCustomLayout.

clsDialEnvOptCard provides a default behavior of observing the dialing environment and refreshing dialing environment option cards when the dialing environment changes.

clsDialEnvField inherits from clsField.

clsDialEnvField alters the a default behavior of ancestor clsField by specifying a character list template for coercing its field input.

Dialing environments are a location type service and therefore managed by a service manager called theLocations. Each instance of a dialing environment is identified by the name of a location to which the dialing environment pertains (NOTE: for PenPoint 1.0 there is only a single location/dialing environment). Objects wishing to communicate with a dialing environment do so by sending messages to the current location service. The UID of the current location is obtained by querying theLocations via standard install manager and service manager messages. The following block of code provides one example of how a client might obtain dialing environment data.

```cpp
{  
  OBJECT handleCurrentLoc, theCurrentLocation;
  SM_QUERY_LOCK lock;
  SM_QUERY_UNLOCK unlock;
  DIALENV_COUNTRY country;
  IM_GET_SET_NAME getName;
  CHAR locationName[nameBufLength];

  // Get the handle and UID of the current location.
  // Lock the current location to guarantee exclusive access to 
  // location data.
  // Get the country code for the current location (from the dialing
  // environment for the current location).
  // Unlock the current location so that other clients may access it.
  // Get the name of the current location.
  //
  ObjCallJump(msgIMGetCurrent, theLocations, &handleCurrentLoc, s, Problem);
  lock.handle = unlock.handle = handleCurrentLoc;
```
ObjCallJmp(msgSMQueryLock, theLocations, &lock, s, Problem);
theCurrentLocation = lock.service;
ObjCallJmp(msgDialEnvGetCountry, theCurrentLocation, &country, s, Problem);
ObjectCall(msgSMQueryUnlock, theLocations, &unlock);
getName.handle = handleCurrentLoc;
getName.pName = locationName;
ObjCallJmp(msgIMGetName, theLocations, &getName, s, Problem);)

For PenPoint 1.0 an application or service requiring dialing environment services should install the
dialing environment dll via a SERVICE.INI file.

**** Future Direction Ideas ****

In a future release of PenPoint, dialing environments will be subsumed by a location service. The
location service will manage all of the objects which provide location-dependent behavior to the
PenPoint environment/applications. Current plans are for the user to access location services via the
configuration notebook. Because dialing environments will be a constituent of a location service it won’t
be necessary for a dialing environment to be included by an application’s or service’s SERVICE.INI file.
The location service will maintain the list of locations the user has created (GO may ship pre-configured
locations; however a user will be able to create and modify locations). A user will select a location by
name, and all of the unique properties regarding that location will take effect.

For each location there may be a dialing environment. Thus, whenever the user selects a new location, a
different dialing environment may take effect (it is possible that two different locations will share the
same dialing environment, or that a location doesn’t have a dialing environment). When a user creates a
new location, the user will be given the opportunity to specify a dialing environment for the new
location, or to select one of the currently available dialing environments and bind it to the new location.
The dialing environment will be enhanced to provide clients with information regarding valid city/area
codes and dialing rules for specific countries. This information can be presented to the user for UI
pick-lists, used to coerce input to only valid combinations of codes, and to enforce the rules which
national telephone systems impose on computer software which interacts with the public telephone
system.

**** End of Future Direction Ideas ****

clsDialEnvOptCard provides a default behavior of observing the dialing environment and refreshing
dialing environment option cards when the dialing environment changes. A client needn't provide any
special code support to have such option cards track dialing environment changes. Note: A client
shouldn't insert a dialing environment option card into an option sheet or any window tree with a
modal filter (e.g. option sheet with a style modality set to either osModalApp or osModalSystem).
The following block of code provides one example of creating a dialing environment option card.

{ //
  // Create an option card for dialing environment settings.
  //
  STATUS s;
  DIALENV_OPTCARD_NEW don;
  OBJECT handleCurrentLoc;
  IM_GET_SET_NAME getName;
  CHAR locationName[nameBufLength];
Get the handle and name of the current location. Create a dialing environment option card for the current location.

ObjCallRet(msgIMGetCurrent, theLocations, &handleCurrentLoc);
getName.handle = handleCurrentLoc;
getName.pName = locationName;
ObjCallRet(msgIMGetName, theLocations, &getName, S);
ObjCallRet(msgNewDefaults, clsDialEnvOptCard, &don, S);
don.win.tag = tagDialEnvOptionCard;
strcpy(don.dialenvOptCard.dialEnv.name, locationName);
ObjCallRet(msgNew, clsDialEnvOptCard, &don, s);
}

clsDialEnvField alters the default behavior of ancestor clsField by specifying a character list template for coercing its field input.

Defined within this header file.

- defines and typedefs for dial environment data. function prototypes, messages & status values.

```c
#ifndef DIALENV_INCLUDED
#define DIALENV_INCLUDED
#endif
#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif
#ifndef SERVICE_INCLUDED
#include <service.h>
#endif
#ifndef CLAYOUT_INCLUDED
#include <clayout.h>
#endif
#ifndef FIELD_INCLUDED
#include <field.h>
#endif

Defines and typedefs

Class UIDs for:

- clsDialEnv The dialing environment service.
- clsDialEnvOptCard Dialing environment option cards.
- clsDialEnvField Field for entering and coercing dialing codes/numbers.
- theLocations Service manager for dialing environments.

```c
#define clsDialEnv
#define clsDialEnvOptCard
#define clsDialEnvField
#define theLocations
```

Dialing Environment Quick Help.

- Quick help is stored in clsDialEnv resource list 0.
- Each quick help entry is located by its index/position within resource list 0.

```c
#define resListDialEnvQHelp 0
#define MakeDialEnvQHelpResId(x) MakeIndexedResId(clsDialEnv, resListDialEnvQHelp, x)
#define tagDialEnvOptCard MakeTag(clsDialEnv, 1)
#define hlpDialEnvOptCard MakeDialEnvQHelpResId(0)
#define tagDialEnvDialEnvTable MakeTag(clsDialEnv, 17)
#define hlpDialEnvDialEnv MakeDialEnvQHelpResId(0)
```
#define tagDialEnvCurrentLocLabel MakeTag(clsDialEnv, 18)
#define tagDialEnvCurrentLocTable MakeTag(clsDialEnv, 19)
#define hlpDialEnvCurrentLoc MakeDialEnvQHelpResld(6)
#define tagDialEnvDialLabel MakeTag(clsDialEnv, 24)
#define tagDialEnvDial MakeTag(clsDialEnv, 25)
#define tagDialEnvDialTone MakeTag(clsDialEnv, 26)
#define tagDialEnvDialPulse MakeTag(clsDialEnv, 27)
#define hlpDialEnvDial MakeDialEnvQHelpResld(7)
#define tagDialEnvAreaCityLabel MakeTag(clsDialEnv, 32)
#define tagDialEnvAreaCity MakeTag(clsDialEnv, 33)
#define hlpDialEnvAreaCity MakeDialEnvQHelpResld(9)
#define tagDialEnvCountryLabel MakeTag(clsDialEnv, 40)
#define tagDialEnvCountry MakeTag(clsDialEnv, 41)
#define hlpDialEnvCountry MakeDialEnvQHelpResld(8)
#define tagDialEnvOutsideLineLabel MakeTag(clsDialEnv, 48)
#define tagDialEnvOutsideLine MakeTag(clsDialEnv, 49)
#define hlpDialEnvOutsideLine MakeDialEnvQHelpResld(1)
#define tagDialEnvLongDistLabel MakeTag(clsDialEnv, 56)
#define tagDialEnvLongDist MakeTag(clsDialEnv, 57)
#define hlpDialEnvLongDist MakeDialEnvQHelpResld(2)
#define tagDialEnvIntlAccessLabel MakeTag(clsDialEnv, 64)
#define tagDialEnvIntlAccess MakeTag(clsDialEnv, 65)
#define hlpDialEnvIntlAccess MakeDialEnvQHelpResld(3)
#define tagDialEnvSuffixLabel MakeTag(clsDialEnv, 72)
#define tagDialEnvSuffix MakeTag(clsDialEnv, 73)
#define hlpDialEnvSuffix MakeDialEnvQHelpResld(4)
#define tagDialEnvMacroCodesLabel MakeTag(clsDialEnv, 80)
#define tagDialEnvMacroCodes MakeTag(clsDialEnv, 81)
#define hlpDialEnvMacroCodes MakeDialEnvQHelpResld(5)
#define tagDialEnvSetCodes MakeTag(clsDialEnv, 82)
#define tagDialEnvMacroCodeALabel MakeTag(clsDialEnv, 83)
#define tagDialEnvMacroCodeA MakeTag(clsDialEnv, 84)
#define tagDialEnvMacroCodeBLabel MakeTag(clsDialEnv, 85)
#define tagDialEnvMacroCodeB MakeTag(clsDialEnv, 86)
#define tagDialEnvMacroCodeCLabel MakeTag(clsDialEnv, 87)
#define tagDialEnvMacroCodeC MakeTag(clsDialEnv, 88)
#define tagDialEnvMacroCodesFrame MakeTag(clsDialEnv, 91)
#define deMaxMacroCodes 4

Exported function prototypes from dialenv.dll

None currently defined.

Message definitions

NOTE msg #1 is reserved for private use.

Observer Notification Messages

msgDialEnvChanged

Notification sent to observers to indicate a dialing environment change.

Takes OBJECT, returns STATUS. Category: observer notification.

#define msgDialEnvChanged MakeMsg(clsDialEnv, 2)
The pArgs indicates the object which initiated the change to the dialing environment. pArgs of objNull indicates that the dialing environment is being destroyed.

Observers which receive this message should refresh any local dialing environment information or view of such information.

Error Return Values: N/A.

### Action Messages

**msgDialEnvGetCountry**

Passes back the country code from the current dialing environment.

Takes P_DIALENV_COUNTRY, returns STATUS. Category: service action request.

```c
#define msgDialEnvGetCountry MakeMsg(clsDialEnv, 3)
```

**Arguments**

```c
typedef struct DIALENV_COUNTRY
{
    CHAR symbols[lenDialEnvCountry+1];
} DIALENV_COUNTRY, *P_DIALENV_COUNTRY;
```

**Comments**

Error Return Values: none, always returns stsOK.

**msgDialEnvIsCountryNorthAmerican**

Indicates whether or not the specified country code is North American.

Takes P_DIALENV_COUNTRY, returns STATUS. Category: service action request.

```c
#define msgDialEnvIsCountryNorthAmerican MakeMsg(clsDialEnv, 6)
```

**Arguments**

```c
typedef struct DIALENV_COUNTRY
{
    CHAR symbols[lenDialEnvCountry+1];
} DIALENV_COUNTRY, *P_DIALENV_COUNTRY;
```

**Comments**

NOTES: This message is provided so a client may alter its UI and/or enforce editing rules unique to North American phone numbers.

Returns stsOK if the specified country is North American, otherwise stsDialEnvNoMatch.

**msgDialEnvGetEnvironment**

Passes back the current dialing environment settings.

Takes P_DIALENV_ENVIRONMENT, returns STATUS. Category: service action request.

```c
#define msgDialEnvGetEnvironment MakeMsg(clsDialEnv, 4)
```

**Arguments**

```c
typedef TAG DIALENV_DIAL_MODE;
#define deTone tagDialEnvDialTone // Touch tone dialing.
#define dePulse tagDialEnvDialPulse // Pulse code dialing.
```
typedef struct DIALENV_INTL_ACCESS
{
    CHAR symbols[lenDialEnvIntIAccess+l];
} DIALENV_INTL_ACCESS, *P_DIALENV_INTL_ACCESS;

typedef struct DIALENV_LONG_DIST
{
    CHAR symbols[lenDialEnvLongDist+l];
} DIALENV_LONG_DIST, *P_DIALENV_LONG_DIST;

Symbols appended to a dialing string. Typically for credit card billing/call accounting purposes.

typedef struct DIALENV_SUFFIX
{
    CHAR symbols[lenDialEnvSuffix+l];
} DIALENV_SUFFIX, *P_DIALENV_SUFFIX;

Multi-purpose codes for specifying credit card #s, account billing codes, or altering environment dependent behavior. When a client requests to build a dial string, the symbols from a macro code get expanded into the resultant dial string.

typedef struct DIALENV_MACRO_CODE
{
    CHAR symbols[lenDialEnvMacroCode+l];
} DIALENV_MACRO_CODE, *P_DIALENV_MACRO_CODE;

typedef struct DIALENV_ENVIRONMENT
{
    DIALENV.DialMode; // Dial mode (tone/pulse).
    DIALENV.OutsideLine; // Outside line/net access.
    DIALENV.AreaCity; // Area/City call originates from.
    DIALENV.Country; // Country call originates from.
    DIALENV.IntlAccess; // International access code.
    DIALENV.LongDist; // Long distance access code.
    DIALENV.Suffix; // Suffix applied to dial strings.
    DIALENV_MACRO_CODE macroCode[numDialEnvMacroCodes]; // Macro/expand codes.
} DIALENV_ENVIRONMENT, *P_DIALENV_ENVIRONMENT;

Error Return Values: none, always returns stsOK.

Symbols prefixed to a dialing string to gain access to the general switched telephone network.

msgDialEnvBuildDialString

Construct a dial string based upon the current dialing environment.

Takes P_DIALENV_BUILD_DIALSTR, returns STATUS. Category: service action request.

#define msgDialEnvBuildDialString

typedef struct DIALENV_TELEPHONE_NUMBER
{
    CHAR country[lenDialEnvCountry+l]; // Cntry call originates from.
    CHAR areaCity[lenDialEnvAreaCity+l]; // Area/City call origs from.
    CHAR teleNumber[lenDialEnvTeleNumber+l]; // Destination telephone #.
    CHAR postConnect[lenDialEnvPostConnect+l]; // Post connect destination network navigation code.
} DIALENV_TELEPHONE_NUMBER, *P_DIALENV_TELEPHONE_NUMBER;

The resultant string of symbols a dialer sends to either clsModem, the phone network, or another server which performs the dialing.

typedef struct DIALENV_DIAL_STRING
{
    CHAR symbols[lenDialEnvDialString+l];
} DIALENV_DIAL_STRING, *P_DIALENV_DIAL_STRING;
typedef struct DIALENV_BUILD_DIALSTR
{
  P_DIALENV_TELEPHONE_NUMBER pTeleNumber; // In: Raw tel # to dial.
  P_DIALENV_DIAL_STRING pDialString; // Out: Resultant dial str.
} DIALENV_BUILD_DIALSTR, *P_DIALENV_BUILD_DIALSTR;

NOTE: The order in which macro codes are processed is significant. All like macro codes are expanded before the next macro code is expanded. Thus if expansion of macro code N results in symbols for a subsequent macro code (e.g. N+1) to be inserted into the dial string, such symbols will be interpreted as and expanded as macro codes.

Error Return Values:  stsDialEnvDialStrTooLarge

Class Messages

msgNew

Creates an instance of a dialing environment.

Takes P_DIALENV_NEW, returns STATUS. Category: class message.

typedef DIALENV_ENVIRONMENT DIALENV_NEW_ONLY, *P_DIALENV_NEW_ONLY;
#define dialenvNewFields \
  serviceNewFields \
  DIALENV_NEW_ONLY dialEnv;

Arguments
typedef struct DIALENV_NEW
{
  dialenvNewFields
} DIALENV_NEW, *P_DIALENV_NEW;

Comments

Error Return Values:  percolated up from other classes, none from clsDialEnv.

msgNewDefaults

Initializes the DIALENV_NEW structure to default values.

Takes P_DIALENV_NEW, returns STATUS. Category: class message.

Message

typedef struct DIALENV_NEW
{
  dialenvNewFields
} DIALENV_NEW, *P_DIALENV_NEW;

Arguments

Sets:

pArgs->svc.style.waitForTarget =
pArgs->svc.style.exclusiveOpen =
pArgs->svc.style.autoOwnTarget =
pArgs->svc.style.autoOpen =
pArgs->svc.style.autoMsgPass =
pArgs->svc.style.checkOwner = false;
pArgs->svc.pManagerList = pManagerList; // theLocations
pArgs->svc.numManagers = 1;
memset(&(pArgs->dialEnv), 0, sizeof(pArgs->dialEnv));
pArgs->dialEnv.dialMode = deTone; // Tone dialing.
  // All remaining struct dialEnv
  // fields are set to zero/null.

Error Return Values:  percolated up from other classes, none from clsDialEnv.
msgDialEnvGetMacroIds
Passes back a string of symbols which identify dialing macro codes.
Takes P_DIALENV_MACRO_IDS, returns STATUS. Category: class message.

#define msgDialEnvGetMacroIds MakeMsg(clsDialEnv, 6)

Arguments
typedef struct DIALENV_MACRO_IDS
  {
    CHAR symbols[numDialEnvMacroCodes+1];
  } DIALENV_MACRO_IDS, *P_DIALENV_MACRO_IDS;

Comments Error return values: percolated up from other classes, none from clsDialEnv.

\*clsDialEnv non-error status values
None currently defined

\*clsDialEnv error status values
The request sent to the dialing environment has been denied because the request isn't supported by this
dialing environment.

#define stsDialEnvRequestDenied MakeStatus(clsDialEnv, 1)
The request sent to the dialing environment specified an invalid country code.

#define stsDialEnvInvalidCountry MakeStatus(clsDialEnv, 2)
The request sent to the dialing environment contained data which didn't match the specified
constraints.

#define stsDialEnvNoMatch MakeStatus(clsDialEnv, 3)
The dial string resulting from msgDialEnvBuildDialString is too large to be contained within struct
DIALENV_DIAL_STRING.

#define stsDialEnvDialStrTooLarge MakeStatus(clsDialEnv, 4)

\*Message definitions ....

NOTE msg #1 reserved for private use.

\*Action Messages

msgDialEnvOptCardRefresh
Refreshes a dialing environment option card (self) with the current dialing environment settings.
Takes nothing, returns STATUS. Category: action request.

#define msgDialEnvOptCardRefresh MakeMsg(clsDialEnvOptCard, 2)

Comments A client should send msgDialEnvOptCardRefresh to a dialing environment option card when it
receives msgOptionRefreshCard and the card tag matches that assigned to the dialing environment
option card.

Error Return Values: percolated up from other classes, none from clsDialEnv.
**msgDialEnvOptCardApply**

Updates the dialing environment with current settings from a dialing environment option card (self).

Takes nothing, returns STATUS. Category: action request.

```c
#define msgDialEnvOptCardApply MakeMsg(clsDialEnvOptCard, 3)
```

Comments

A client should send **msgDialEnvOptCardApply** to a dialing environment option card when it receives **msgOptionApplyCard** and the card tag matches that assigned to the dialing environment option card.

Error Return Values: percolated up from other classes, none from clsDialEnv.

---

**Class Messages**

**msgNew**

Creates an instance of a dialing environment option card.

Takes P_DIALENV_OPTCARD_NEW, returns STATUS. Category: class message.

Arguments

```c
typedef struct LOCATION_NAME
{  
  CHAR name[nameBufLength];  // Name of a location.
} LOCATION_NAME, *P_LOCATION_NAME;
// Name of a dialing environment.

typedef LOCATION_NAME DIALENV_NAME, *P_DIALENV_NAME;
typedef struct DIALENV_OPTCARD_NEW_ONLY
{  
  DIALENV_NAME dialEnv;  // Name of DialEnv supplying info.
  U32 spare1;  // unused (reserved).
  U32 spare2;  // unused (reserved).
} DIALENV_OPTCARD_NEW_ONLY, *P_DIALENV_OPTCARD_NEW_ONLY;
#define dialenvOptCardNewFields  
  customLayoutNewFields  
  DIALENV_OPTCARD_NEW_ONLY dialenvOptCard;
typedef struct DIALENV_OPTCARD_NEW
{  
  dialenvOptCardNewFields
} DIALENV_OPTCARD_NEW, *P_DIALENV_OPTCARD_NEW;
```

Comments

A client may add the dialing environment option card to its stack of option cards, and create it in response to **msgOptionProvideCard** via this message. Clients may create multiple cards and insert them into any window. The cards needn't be part of an option card stack.

NOTES: It is possible for one or more clients to create multiple dial environment option cards. Because of this, dialing environment option cards observe the dialing environment. When the dialing environment changes, all dialing environment cards get refreshed with current dialing environment settings.

The requestor must fill in the pArgs->dialEnv with the name of the location which will supply the option card with dialing environment settings.

Error Return Values: percolated up from other classes, stsDialEnvOptCardBadEnvironment.
msgNewDefaults
Initializes the DIALENV_OPTCARD_NEW structure to default values.
Takes P_DIALENV_OPTCARD_NEW, returns STATUS. Category: class message.

Message Arguments
typedef struct DIALENV_OPTCARD_NEW
{
   dialenvOptCardNewFields
} DIALENV_OPTCARD_NEW, *P_DIALENV_OPTCARD_NEW;

Comments
Sets:
memset(pArgs->dialenvOptCard.dialEnv.name, Nil(CHAR),
    sizeof(pArgs->dialenvOptCard.dialEnv.name));

\* clsDialEnvOptCard non-error status values
None currently defined

\* clsDialEnvOptCard error status values
An internal system error was encountered creating an instance of clsDialEnvOptCard.
#define stsDialEnvOptCardProblem MakeStatus(clsDialEnvOptCard, 1)
The arguments specified via msgNew to clsDialEnvOptCard didn’t specify a dialing environment (from
which data for the option card is obtained).
#define stsDialEnvOptCardBadEnvironment MakeStatus(clsDialEnvOptCard, 2)
An internal system error was encountered unfiling clsDialEnvOptCard from a resource file.
#define stsDialEnvOptCardBadResFile MakeStatus(clsDialEnvOptCard, 3)
An internal system error was encountered when attempting to locate a window (containing option data)
within a dialing environment option card.
#define stsDialEnvOptCardNoSuchOption MakeStatus(clsDialEnvOptCard, 4)

\* Message definitions

msgNew
Creates an instance of a dialing environment field.
Takes P_DIALENV_FIELD_NEW, returns STATUS. Category: class message.
#define dialenvFieldNewFields
    fieldNewFields

Arguments
typedef struct DIALENV_FIELD_NEW
{
   dialenvFieldNewFields
} DIALENV_FIELD_NEW, *P_DIALENV_FIELD_NEW;

Comments
clsDialEnvField logic within its msgInit method:
DIALENV_MACRO_IDS macroIds;
CHAR fieldCharList[20+numDialEnvMacroCodes+1];
XTM ARGS template;
P_STRING fieldChars = "0123456789()\-,\#\!\";
// If the client hasn't modified the default field template value, // establish a template to coerce dialing environment field input. // Query clsDialEnv to obtain the symbols identifying macro // codes. Append them to base dialing type characters.

if (pArgs->field.xlate.pTemplate == pNull && pArgs->field.style.xlateType == fstXlateTemplate)
{
    macroIds.symbols[0] = Nil(CHAR);
    ObjCallWarn(msgDiaIEnvGetMacroIds, clsDialEnv, &macroIds);
    strcpy(fieldCharList, fieldChars);
    strcat(fieldCharList, macroIds.symbols);
    template.xtmType = xtmTypeCharList; // Char list type template.
    template.xtmMode = xtmModeDefault; // No special template mode.
    template.pXtmData = fieldCharList; // The character list.
    pArgs->field.xlate.pTemplate = &template;
}
// Call our ancestor to create the object.
return ObjectCallAncestor(msg, self, pArgs, ctx);

Error Return Values: percolated up from other classes,

msgNewDefaults

Initializes the DIALENV_FIELD_NEW structure to default values.

Takes P_DIALENV_FIELD_NEW, returns STATUS. Category: class message.

typedef struct DIALENV_FIELD_NEW
{
    dialenvFieldNewFields
} DIALENV_FIELD_NEW, *P_DIALENV_FIELD_NEW;

Sets:

// Establish defaults for an instance of clsDialEnvField.
// pArgs->field.style.veto =
// pArgs->field.style.noSpace =
// pArgs->field.style.upperCase = true;
// pArgs->field.style.xlateType = fstXlateTemplate;
// pArgs->field.xlate.pTemplate = &template;
// pArgs->label.style.numCols =
// pArgs->label.style.numRows = IsNumAbsolute;
// pArgs->label.cols = 12;
// pArgs->label.rows = 1;
// pArgs->border.style.edge = bsEdgeBottom;
// pArgs->border.style.topMargin =
// pArgs->border.style.bottomMargin = bsMarginMedium;
// pArgs->border.style.borderInk = bsInkGray66;
This file contains the API definition for clsFLAP.

clsFLAP inherits from clsMILService.

This mil service provides the interface between the ALAP mil device and the rest of Penpoint. This interface allows for the configuring of the ALAP mil device and for PenTops networking using the ALAP mil device. The flap mil service will typically only be accessed by link level drivers since the mil service is responsible for providing the lowest levels of the PenTops protocol stack.

This mil service responds to the messages defined in the link.h header file. Refer to link.h for message definitions.

You access this mil service by using the standard service access techniques. These techniques are described in servmgr.h.

The flap mil service is a member of the 'theLinkHandlers' service manager.

```c
ifndef FLAP_INCLUDED
#define FLAP_INCLUDED
ifndef MIL_SERVICE_INCLUDED
#include <milserv.h>
#endif
ifndef LINK_INCLUDED
#include <link.h>
#endif

msgNew creates a new flap object.

Takes P_FLAP_NEW, returns STATUS.

#define flapNewFields
  milServiceNewFields

typedef struct FLAP_NEW {
  flapNewFields
} FLAP_NEW, *P_FLAP_NEW;

STATUS EXPORTEDClsFLAPInit(void);
This file contains the definition and methods for clsALAPHighSpeed.

clsALAPHighSpeed inherits from clsLink (see link.h).

```c
#ifndef HSLINK_INCLUDED
#define HSLINK_INCLUDED
#define alapHighSpeedNewFields serviceNewFields
typedef struct ALAP_HSLINK_NEW
{
    alapHighSpeedNewFields
} ALAP_HSLINK_NEW, *P_ALAP_HSLINK_NEW;
STATUS EXPORTED CIsALAPHSLinkInit(void);
```
HSPKT.H

This file contains the API definition for clsHighSpeedPacket.

clsHighSpeedPacket inherits from clsService.

Provides a high speed packet transfer API.

```c
#ifndef HSPKT_INCLUDED
#define HSPKT_INCLUDED
#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef MILSERV_INCLUDED
#include <milserv.h>
#endif
#ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif
#ifndef DVHS_PKT_INCLUDED
#include <dvhspkt.h>
#endif
#endif
```

**Common #defines and typedefs**

```c
typedef struct HS_PACKET_METRICS {
    U16 version;       // version number
    U16 status;        // current status
    U32 asyncBaud;     // baud rate for async serial mode
    U16 parConnectChar; // connect character for connection
                         // testing (parallel mode only)
    U16 parConnectAckChar; // character to return upon reception
                           // of parConnectChar (parallel mode
                           // only)
    U16 leadInChar;    // default lead in character
    U16 dataAckChar;   // default acknowledgement character
                        // (return upon reception of 1st data
                        // byte or of packet lead in character
                        // if one is defined).
    MIL_HS_PACKET_DEVICE_TYPE deviceType; // device type (see dvhspkt.h)
} HS_PACKET_METRICS, *P_HS_PACKET_METRICS;
typedef OBJECT HS_PACKET, *pHS_PACKET;
#define stsHSPacketBusy MakeStatus(clsHighSpeedPacket, 1)
```

**High Speed Packet Class Messages**

**msgHSPacketStatus**

Returns the current status of the high speed packet device.

Takes P_HS_PACKET_STATUS, returns STATUS.

```c
#define msgHSPacketStatus MakeMsg(clsHighSpeedPacket, 3)
#define hsPktStsBusy flag0       // status
```
PENPOINT API REFERENCE
Part 10 / Connectivity

Arguments
typedef struct HS_PACKET_STATUS
{
  U16 status;
} HS_PACKET_STATUS, *P_HS_PACKET_STATUS;

msgHSHandlePacket
Sends one packet through high speed packet device.
Takes P_HS_PACKET_SEND_PACKET, returns STATUS.

#define msgHSHandlePacket MakeMsg(clsHighSpeedPacket, 9)

Arguments
typedef struct HS_PACKET_SEND_PACKET
{
  P_UNKNOW pBuf;
  U32 numBytes;
  U16 firstByte;
} HS_PACKET_SEND_PACKET, *P_HS_PACKET_SEND_PACKET;

Comments
If leadInChar (in metrics) is zero, firstByte is used as lead in character. If both are zero, no lead in character is sent.

msgHSHandlePacketSetCharHandler
Installs character receive handler.
Takes P_HS_PACKET_CHAR_HANDLER, returns STATUS.

#define msgHSHandlePacketSetCharHandler MakeMsg(clsHighSpeedPacket, 10)

Arguments
Function Prototype
typedef struct HS_PACKET_CHAR_HANDLER
{
  P_HS_PACKET_RX_HANDLER pRxHandler;
  U32 userData;
} HS_PACKET_CHAR_HANDLER, *P_HS_PACKET_CHAR_HANDLER;

Comments
HSHandle calls the user-defined function when a character is received. The called function must collect the provided character and return either true if the packet is complete, false otherwise.

userData in HS_PACKET_RX_HANDLER is the user-provided userData U32 in HS_PACKET_CHAR_HANDLER.

If leadInChar (in metrics) is zero, the first character received is contained in both the firstByte and the receivedByte parameters to P_HS_PACKET_RX_HANDLER().
The received character handler will not be installed if one already is. See msgHSHandlePacketFreeCharHandler.
The character handler is automatically freed when the service is closed.

msgHSHandlePacketFreeCharHandler
Deinstalls a previously installed character receive handler.
Takes P_HS_PACKET_CHAR_HANDLER, returns STATUS.

#define msgHSHandlePacketFreeCharHandler MakeMsg(clsHighSpeedPacket, 11)
Function prototypes

**msgHSPacketEnable**
Starts the continuous function which tests for connection and make ourselves "visible" to others.
Takes nothing, returns STATUS.

```c
#define msgHSPacketEnable MakeMsg(clsHighSpeedPacket, 12)
```

**msgHSPacketDisable**
Stops the continuous function (started by `msgHSPacketEnable`) which tests for connection and become "invisible".
Takes nothing, returns STATUS.

```c
#define msgHSPacketDisable MakeMsg(clsHighSpeedPacket, 13)
```

**msgNew**
Creates a new hspkt object.
Takes `P_HS_PACKET_NEW`, returns STATUS.

```c
#define hspktNewFields milServiceNewFields
```

```c
typedef struct HS_PACKET_NEW {
    hspktNewFields
} HS_PACKET_NEW, *P_HS_PACKET_NEW;
```

**Function prototypes**

```c
STATUS EXPORTED ClsHSPacketInit(void);
```
INBXSVC.H

This file contains the API definition for clsINBXService.

clsINBXService inherits from clsIOBXService.

Provides default behavior for Inbox Services.

```c
#ifndef INBXSVC_INCLUDED
#define INBXSVC_INCLUDED
#endif
#include <iobxsvc.h>
#endif
```

Introduction

In PenPoint, input operations are handled by a special class of services known as the "inbox services." While most input operations are triggered by an external event such as an incoming fax image from a remote fax machine, some input operations may require that the PenPoint computer be one that initiates the communication process. For example, a fax input service may wish to periodically "poll" a "store-and-forward" facility in order to receive a fax image. Thus, an inbox service implements the "deferred input" feature in PenPoint: This concept permits a user to specify input operations regardless of the readiness of input devices. If the input device (e.g., a data/fax modem, a LAN connection, etc.) is not available or not connected, the input process is deferred until the input device becomes ready.

Passive vs. Active Inbox Services

The simplest type of inbox services are those who passively wait for an input event to happen. That is, after the input operation is initiated by a remote agent such as a fax machine, the inbox service running on a PenPoint computer detects the input event and then receives the incoming data stream. This type of inbox services do not initiate an input operation by themselves. Typically, when such a service is enabled by the user, it simply becomes the owner of the I/O device. A simple fax inbox service, for example, becomes the owner of the fax modem and sets it up to start receiving fax images whenever a phone call comes in. While the inbox service owns the I/O device, no other services can transmit or receive data through the same device. (For more details on the notion of service ownership, see the service API in service.h.)

Some inbox services may want to actively "solicit" input from a remote agent. For example, a service that queries a remote database will have to establish the communication link between the PenPoint computer and the remote database server. For this type of services, clsINBXService provides default behaviors to manage the state of the I/O device (connected or disconnected), the permission to initiate input operation (whether the service is enabled or disabled), as well as automatic polling behavior similar to that of an outbox service. Thus, the user can "defer" the input operation until it becomes possible to establish a communication link with a remote agent. See the API for clsOBXService for a detailed discussion of the deferred input/output protocol. Note, however, that to enable such outbox-like behavior, the polling flag must be turned on when the service is created. I.e., in `msgNewDefaults`, you should set

```c
pArgs->iobxsvc.in.autoPoll = true;
```
Inbox Documents

Normally, documents can be automatically created in an inbox section as the end result of an input event. For example, a fax inbox section may create a document containing the fax images received in the fax modem. Such documents are normal PenPoint documents. Their contents have nothing to do with the input device or where the document came from.

Sometimes an inbox document contains not only data, but also some control information about the input operation to be performed. For example, taking advantage of the "deferred input" feature, the user may construct a specific query statement for an online database and put it into the appropriate inbox section before the PenPoint machine is physically connected to the remote database. When the input service becomes ready, the query statement is sent to the remote database, and the result is put into either another document or the same document containing the query statements. This type of inbox documents is very similar to the outbox document that controls the actual output operation. Again, for more information about the deferred input/output protocol, see obxsvc.h.

Note that the deferred I/O protocol implemented by clsINBXService assumes that an input operation is controlled by an inbox document: an assumption that may be too cumbersome and confusing for many services. If such is the case, an inbox service can simply store the input control information (e.g., a database query statement) with the service itself. When the service receives msgINBXSvcPollDocuments, it simply handles the input operation directly and bypasses the rest of the protocol.

Services that Handle Input and/or Output

clsINBXService deals only with input operations. For those services that want to handle output operations or both input and output at the same time, two other classes, clsOBXService and clsIOBXService, are provided by PenPoint. In fact, clsINBXService and clsOBXService are implemented as a subclass (hence a subset) of clsIOBXService.

Class Messages

msgNewDefaults

Initializes the P_INBXSVCSVC_NEW structure to default values.

Takes P_INBXSVCSVC_NEW, returns STATUS. Category: class message.

Arguments

typedef struct INBXSVCSVC_NEW { 
  OBJECT sectionClass; // class of the inbox section
    // This must be clsNBToc or a subclass of it.
  U32 unused1;
  U32 unused2;
  U32 unused3;
} INBXSVCSVC_NEW_ONLY, *P_INBXSVCSVC_NEW_ONLY;
#define inbxServiceNewFields
  ioSvcNewFields
  INBXSVCSVC_NEW_ONLY inbxsvc;

typedef struct INBXSVCSVC_NEW { 
  inbxServiceNewFields
  INBXSVCSVC_NEW_NEW, *P_INBXSVCSVC_NEW;
} INBXSVCSVC_NEW, *P_INBXSVCSVC_NEW;

Zeroes out pArgs->inbxsvc and sets...>inbxsvc.in.autoPoll = false; inbxsvc.sectionClass = clsNBToc;
### msgNew
Creates a new inbox service object.

Takes P_INBXSVC_NEW, returns STATUS. Category: class message.

**Message**

```
typedef struct INBXSVC_NEW {
    inbxServiceNewFields
} INBXSVC_NEW, *P_INBXSVC_NEW;
```

### msgINBXSvcSwitchIcon
Toggles the inbox icon (to empty or filled) if necessary.

Takes nothing, returns STATUS. Category: class message.

```
#define msgINBXSvcSwitchIcon msgIOBXSvcSwitchIcon
```

**Comments**

Check the content of the inbox notebook. Show the "filled" icon if any document is found. Show the "empty" icon otherwise.

### msgINBXDocGetService
Gets the service name.

Takes P_INBX_DOC_GET_SERVICE, returns STATUS. Category: class message.

```
#define msgINBXDocGetService msgIOBXDocGetService
```

**Arguments**

```
typedef struct INBX_DOC_GET_SERVICE {
    OBJECT document; // In: document uid
    CHAR svcName[nameBufLength]; // Out: service name
} INBX_DOC_GET_SERVICE, *P_INBX_DOC_GET_SERVICE;
```

**Comments**

Get the name of the service associated with an inbox document. If the document has not been placed into an inbox section, stsFailed is returned.

Note that the document must be at the top level of an inbox section. That is, if the document is embedded within another document, which is in an inbox section, stsFailed will be returned.

### msgINBXDocInInbox
Checks if a document is in a section in the Inbox.

Takes P_INBX_DOC_IN_INBOX, returns STATUS. Category: class message.

```
#define msgINBXDocInInbox msgIOBXDocInIOBox
```

**Arguments**

```
typedef struct INBX_DOC_IN_INBOX {
    UUID uuid; // In: document uid
    CLASS svcClass; // In: service class to check for
} INBX_DOC_IN_INBOX, *P_INBX_DOC_IN_INBOX;
```

**Comments**

This message can be sent to clsINBXService to check if a PenPoint document represented by pArgs->uuid is already in the input queue of an inbox service inheriting from pArgs->svcClass. stsOK is returned if it is, stsFailed otherwise. If pArgs->svcClass is objNull, stsOK is returned if the document is anywhere in the Inbox notebook.
Messages Sent to an Inbox Service Instance

msgINBXSvcMoveInDoc
Moves a document into the inbox section.
Takes P_INBXSVCEMOVE_COPY_DOC, returns STATUS.
#define msgINBXSvcMoveInDoc msgIOBXSvcMoveInDoc

typedef struct INBXSVCEMOVE_COPY_DOC {
FS_LOCATOR source; // In: Location of source document.
U16 sequence; // In: Sequence number to move/copy
// in front of.
} INBXSVCEMOVE_COPY_DOC, *P_INBXSVCEMOVE_COPY_DOC;

Comments
Superclass behavior is to move the document located at pArgs->source into the input queue associated with the inbox service. For example, set pArgs->sequence to 1 to move the document to the top of the queue. Set it to maxU16 to move the document to the bottom of the queue.

After the document is moved (or copied) to the input queue, it is considered to be in a state ready for input, even though the service may not be connected at the time. Client should not alter the document in any way once it has been moved to the input queue.

Subclasses can provide their own behavior if they wish. Remember to use the class message msgINBXSvcSwitchIcon to change the inbox icon.

msgINBXSvcCopyInDoc
Copies a document into the Inbox section.
Takes P_INBXSVCEMOVE_COPY_DOC, returns STATUS.
#define msgINBXSvcCopyInDoc msgIOBXSvcCopyInDoc

 typedef struct INBXSVCEMOVE_COPY_DOC {
FS_LOCATOR source; // In: Location of source document.
U16 sequence; // In: Sequence number to move/copy
// in front of.
} INBXSVCEMOVE_COPY_DOC, *P_INBXSVCEMOVE_COPY_DOC;

Comments
Same as msgINBXSvcMoveInDoc, except that the document is copied to the input queue.

msgINBXSvcGetTempDir
Passes back a handle for a temporary directory.
Takes P_OBJECT, returns STATUS.
#define msgINBXSvcGetTempDir msgIOBXSvcGetTempDir

Comments
This message is provided for clients who may want ot prepare their input document before moving it into the input queue. The handle of an "official" temporary directory is passed back and it can be used as temporary storage for documents, data, etc. Clients are responsible for deleting temporary files when they are done. The directory will be flushed after a warm boot.

msgINBXSvcPollDocuments
Polls all documents in an input queue and input those who are ready.
Takes nothing, returns STATUS.
#define msgINBXSvcPollDocuments msgIOBXSvcPollDocuments
This message tells the inbox service to look through its input queue and send out the first document ready for input. The service will first make sure that it is enabled and is connected to the designated input port. If these conditions are met, it will then self-send msgINBXSvcNextDocument to locate the next document ready for input.

If msgINBXSvcNextDocument returns stsOK, indicating that a document is ready for input, this message proceeds to self-send msgINBXSvcLockDocument to lock the document, and finally msgINBXSvcInputStart to initiate the input process.

If msgINBXSvcNextDocument returns stsINBXSvcDocReady, indicating that the section is not empty but none of the documents are ready for input, this message self-sends msgINBXSvcScheduleDocument to schedule the document passed back in pArgs at a later time.

Subclasses normally do not process this message.

msgINBXSvcNextDocument

Passes back the next document ready for input.

Takes P_INBXSVC_DOCUMENT, returns STATUS. Category: self-sent.

```c
#define msgINBXSvcNextDocument
typedef struct INBXSVC_DOCUMENT
{
    OBJECT uid; // uid of the doc
    OBJECT dir; // app dir of the doc
    OBJECT docClass; // class of the doc
    U16 sequence; // sequence of the doc
    CHAR pName[nameBufLength]; // name of this doc
    P_UNKNOWN pDocData; // subclass's private data
} INBXSVC_DOCUMENT, *P_INBXSVC_DOCUMENT;
```

Superclass behavior is to start from the top of the input queue and locate the first document ready for input. If one is found, information about the document is passed back in pArgs. The same pArgs will be passed to messages msgINBXSvcLockDocument and msgINBXSvcInputStart. By default, a document is ready for input when it is closed. If the document is open, it will receive msgINBXSvcInputStartOK and it should return stsOK to indicate that it is ready for input.

Subclasses can provide their own behavior if they wish. Return stsINBXSvcSectionEmpty to give the superclass an opportunity to change the inbox icon from filled to empty.

Return Value

- **stsOK** A document is ready for input.
- **stsINBXSvcSectionEmpty** The input queue is empty.
- **stsINBXSvcDocNotReady** No document in the input queue is ready.

Service-Specific Error Returns.

See Also

msgINBXSvcPollDocuments

msgINBXSvcLockDocument

Locks the document in preparation for input.

Takes P_INBXSVC_DOCUMENT, returns STATUS. Category: self-sent.

```c
#define msgINBXSvcLockDocument
```

```c
typedef struct INBXSVC_DOCUMENT
{
    OBJECT uid; // uid of the doc
    OBJECT dir; // app dir of the doc
    OBJECT docClass; // class of the doc
    U16 sequence; // sequence of the doc
    CHAR pName[nameBufLength]; // name of this doc
    P_UNKNOWN pDocData; // subclass's private data
} INBXSVC_DOCUMENT, *P_INBXSVC_DOCUMENT;
```
typedef struct INBXSVC_DOCUMENT {
    OBJECT uid; // uid of the doc
    OBJECT dir; // app dir of the doc
    OBJECT docClass; // class of the doc
    U16 sequence; // sequence of the doc
    CHAR pName[nameBufLength]; // name of this doc
    P_UNKNOWN pDocData; // subclass's private data
} INBXSVC_DOCUMENT, *P_INBXSVC_DOCUMENT;

This message is a place holder for subclasses that may require additional preparatory work to be performed on a document before it is ready for input. For example, a document may have to be "locked" so that it can not be opened during the input process. This message may be used for other purposes as well. For example, an inbox service may decide to store a light-weight "shadow" document (e.g., a report designator for a database application) in the input queue until it is chosen for input. The service then handles this message by converting the shadow document to a real one (e.g., the actual report).

The superclass behavior for this message is to stamp the document directory with the filesystem attribute iobxsvcDocInputInProgress. This stamp will prevent any gestures over the document from being processed. This means that once a document is locked for input it can not be deleted, renamed, etc. via gestures.

See Also
msgINBXSvcUnlockDocument

msgINBXSvcUnlockDocument

Unlocks a document that was previously locked.

Takes P_INBXSVC_DOCUMENT, returns STATUS. Category: self-sent.

#define msgINBXSvcUnlockDocument msgIOBXSvcUnlockDocument

typedef struct INBXSVC_DOCUMENT {
    OBJECT uid; // uid of the doc
    OBJECT dir; // app dir of the doc
    OBJECT docClass; // class of the doc
    U16 sequence; // sequence of the doc
    CHAR pName[nameBufLength]; // name of this doc
    P_UNKNOWN pDocData; // subclass's private data
} INBXSVC_DOCUMENT, *P_INBXSVC_DOCUMENT;

This message is a place holder for subclasses that may require additional "cleanup" work to be performed on a document before it is put back to the input queue.

The superclass behavior for this message is to remove the iobxsvcDocInputInProgress stamp on the document directory.

See Also
msgINBXSvcLockDocument

msgINBXSvcScheduleDocument

Schedules a document that is not ready for input

Takes P_INBXSVC_DOCUMENT, returns STATUS. Category: self-sent.

#define msgINBXSvcScheduleDocument msgIOBXSvcScheduleDocument

typedef struct INBXSVC_DOCUMENT {
    OBJECT uid; // uid of the doc
    OBJECT dir; // app dir of the doc
    OBJECT docClass; // class of the doc
    U16 sequence; // sequence of the doc
    CHAR pName[nameBufLength]; // name of this doc
    P_UNKNOWN pDocData; // subclass's private data
} INBXSVC_DOCUMENT, *P_INBXSVC_DOCUMENT;
This message is sent when msgINBXSvcNextDocument locates a document in the input queue but the document is not ready for input.

Subclasses should provide their own behavior. The default behavior is to release the ownership of the target service (i.e., become disabled), with the expectation that the user must manually schedule the document later on (by re-enabling the section.)

See Also
msgINBXSvcNextDocument

**msgINBXSvcInputStart**

Starts the input process for a document in the input queue.

Takes P_INBXSVC_DOCUMENT, returns STATUS. Category: self-sent.

```c
#define msgINBXSvcInputStart msgIOBXSvcIOStart
typedef struct INBXSVC_DOCUMENT {
    OBJECT uid;          // uid of the doc
    OBJECT dir;          // app dir of the doc
    OBJECT docClass;     // class of the doc
    U16 sequence;        // sequence of the doc
    CHAR pName[nameBufLength]; // name of this doc
    P_UNKNOWN pDocData;  // subclass's private data
} INBXSVC_DOCUMENT, *P_INBXSVC_DOCUMENT;
```

Comments
Superclass behavior is to activate the inbox document if it isn't already active, and then send msgINBXDocInputStart to the document instance.

Subclasses can provide their own behavior if they wish.

**msgINBXSvcInputCancel**

Cancels the input process.

Takes nothing, returns STATUS.

```c
#define msgINBXSvcInputCancel msgIOBXSvcIOCCancel
```

Comments
This message is sent to the service when the caller wishes to cancel any input operation in progress. The service responds to this message by sending msgINBXDocInputCancel to an active inbox document. After the document is cancelled, the service will post an error note to the user if there are other documents waiting to be processed. The user then decides whether or not the service should proceed to send the remaining documents.

Subclasses do not normally process this message.

**msgINBXSvcInputCleanUp**

Cleans up after the current input is done.


```c
#define msgINBXSvcInputCleanUp msgIOBXSvcIOCleanUp
Enum32 {INBX_DOC_EXIT_BEHAVIOR} {
    inbxDocExitDoNothing,
    inbxDocExitDelete,
    inbxDocExitMarkAsFailed,
    inbxDocExitMarkAsCancelled
};
```
typedef struct INBX_DOC_INPUT_DONE {
    INBX_DOC_EXIT_BEHAVIOR behavior;  // exit behavior
    P_UNKNOWN pDocData;              // Unused: document specific data
} INBX_DOC_INPUT_DONE, *P_INBX_DOC_INPUT_DONE;

This message is posted to self as a result of the service receiving msgINBXDocInputDone, which is sent by the inbox document when it finishes the input operation. The inbox document will be either deleted or marked as specified in pArgs, and when everything is properly cleaned up the service will post msgINBXSvcPollDocuments to self to see if anything else is waiting for input.

Subclasses do not normally process this message.

See Also
msgINBXDocInputDone

msgINBXSvcStateChanged
Tells observers that the service state just changed.

Takes OBJECT, returns STATUS. Category: observer notification.

#define msgINBXSvcStateChanged msgIOBXSvcStateChanged

Comments
Informs observers that the state of a service has just changed. pArgs is the UID of the service.

msgINBXSvcQueryState
Passes back the state of the service.

Takes P_INBXSVC_QUERY_STATE, returns STATUS.

#define msgINBXSvcQueryState msgIOBXSvcQueryState

Arguments
typedef struct {
    BOOLEAN enabled;               // true if the service is enabled.
    CHAR status[nameBufLength];   // text describing the status of
                                  // the service.
    CHAR docName[nameBufLength];  // document being processed
    P_UNKNOWN pStateData;         // subclass’s private data
} INBXSVC_QUERY_STATE, *P_INBXSVC_QUERY_STATE;

Comments
This message is typically used to query what state the service instance is in.

msgINBXSvcGetEnabled
Gets the enabled state of the service.

Takes P_BOOLEAN, returns STATUS.

#define msgINBXSvcGetEnabled msgIOBXSvcGetEnabled

Comments
Subclasses can override this message and redefine the notion of "enabled." The default behavior of the superclass is to equate "enabled" with the ownership of the target service (i.e., input device). That is, the service is "enabled" when it owns the target service. By appending to or replacing the default behavior, a subclass can define additional conditions which must be met before a service is considered enabled.

msgINBXSvcSetEnabled
Sets the enabled state of the service.

Takes BOOLEAN, returns STATUS.

#define msgINBXSvcSetEnabled msgIOBXSvcSetEnabled
This message is sent to the service in response to service notification messages `msgSvcOwnerAcquired` and `msgSvcOwnerReleased`. Subclasses can provide their own behavior and thereby redefine the notion of "enabled" for the service. If they do, they must pass this message up to the ancestor so that observers of the inbox service will be properly notified.

### Inbox Document Messages

**msgINBXDocInputStartOK**

Asks the inbox document if it is OK to start the input process

Takes nothing, returns STATUS.

```c
#define msgINBXDocInputStartOK  msgI0BXDocIOStartOK
```

**Comments**

When an inbox service finds an opened document in the inbox section, it sends this message to the document instance, asking whether it's OK to start the input operation while the document remains open. When the document receives this message, it should return `stsOK` to give the service permission to begin the input process. An error status, including `stsNotUnderstood`, is taken to mean that the document instance vetos the request and the service will not start the input process.

**msgINBXDocInputStart**

Tells an inbox document to start the input process.

Takes nothing, returns STATUS.

```c
#define msgINBXDocInputStart  msgI0BXDocIOStart
```

**Comments**

This message is sent by the inbox service to a document. The document should respond to this message by starting the input process.

**msgINBXDocInputDone**

Tells the inbox service that input is finished.

Takes `P_INBX_DOC_INPUT_DONE`, returns STATUS. Category: client responsibility.

```c
#define msgINBXDocInputDone  msgI0BXDocIODone
```

**Message Arguments**

```c
typedef struct_INBX_DOC_INPUT_DONE {
  INBX_DOC_EXIT_BEHAVIOR behavior; // exit behavior
  P_UNKNOWN  pDocData;  // Unused: document specific data
} INBX_DOC_INPUT_DONE, *P_INBX_DOC_INPUT_DONE;
```

**Comments**

When the input process is finished, the inbox document in charge of the input should send this message to the inbox service. This message must be sent even if the input process has been aborted. The `pArgs` for this message tells the inbox service what to do with the inbox document. If `inbxDocExitDelete` is specified, the document will be removed from the inbox. In all other cases the document will be unlocked and left in the inbox. If either `inbxDocExitMarkAsCancelled` or `inbxDocExitMarkAsFailed` are specified, the name of the document will be altered to provide visual indication for the user that the input process has not completed successfully.

**See Also**

`msgINBXDocGetService`
**msgINBXDocInputCancel**

Tells an inbox document to cancel the input process.

Takes nothing, returns STATUS.

```c
#define msgINBXDocInputCancel      msgIOBXDocIOCancel
```

**Comments**

This message is used by the inbox service to inform a document that it should cancel the input process. The document should handle this message by terminating its input operation and then sending `msgINBXDocInputDone` to the service with `pArgs->behavior` set to `inbxDocExistMarkAsCancelled`.

---

**msgINBXDocStatusChanged**

Tells the inbox service that the document status is changed.

Takes `P_INBX_DOC_STATUS_CHANGED`, returns STATUS. Category: client responsibility.

```c
#define msgINBXDocStatusChanged      msgIOBXDocStatusChanged
```

**Arguments**

```c
typedef struct INBX_DOC_STATUS_CHANGED {
    CHAR status[bufLength]; // Text describing document state
    P_UNKNOWN pDocData;     // Unused: document-specific data
} INBX_DOC_STATUS_CHANGED, *P_INBX_DOC_STATUS_CHANGED;
```

**Comments**

This message is sent by the inbox document to the service whenever its status has just changed. This status is displayed on Status column for the inbox section, in the Inbox notebook.
This file contains the API definition for clsIOBXService.

clsIOBXService inherits from clsService.

```c
#ifndef IOBXSVC_INCLUDED
#define IOBXSVC_INCLUDED
#endif

#ifndef CLSMGR_INCLUDED
#include "clsmgr.h"
#endif

#ifndef GO_INCLUDED
#include "go.h"
#endif

#ifndef SERVICE_INCLUDED
#include "service.h"
#endif

#ifndef AUXNBMGR_INCLUDED
#include "auxnbmgr.h"
#endif
```

**Introduction**

clsIOBXService implements most of the behavior of its two subclasses: clsOBXService (Outbox service class) and clsINBXService (Inbox service class). While its subclasses deal with either the system Inbox or the system Outbox, clsIOBXService allows a service to access both the Inbox and the Outbox at the same time. For details about the two subclasses of clsIOBXService, see inbxsvc.h and obxsvc.h.

**Choosing the Appropriate Superclass for Your Service**

An Outbox service is assigned a section in the system Outbox. Thus, if a service's primary function is to send data out of a PenPoint computer, it should probably be a subclass of clsOBXService. A good example for this type of services is a printer device driver. A very important behavior for an Outbox service is to hold the output data until the physical device is available. This "deferred output" feature allows any documents in an Outbox section to be sent only when the conditions are right for the output operation to commence. This is implemented as a series of messages associated with msgIOBXSvcPollDocuments, which basically "polls" the Outbox section looking for documents to be sent out. By default, all Outbox services inherit such auto polling behavior. (See the IOBXSVC_NEW structure defined in this API for inhibiting this behavior.)

Similarly, an Inbox service is associated with a section in the system Inbox and concerns itself with transferring data into a PenPoint computer. For example, the device driver for an optical scanner should probably be a subclass of clsINBXService. However, the notion of "deferred input" may not apply to most types of Inbox services. Therefore an Inbox service by default does not "poll" the documents in its Inbox section. When "deferred input" does make sense, as in the case of a stock quote service periodically downloading the latest stock prices from a host computer, the auto polling behavior can be easily enabled through the newArgs.

Some services may need to transfer data both into and out of the PenPoint computer. (E.g., an electronic mail service.) There are several alternatives to deal with this situation. First, such services can still
subclass from either clsINBXService or clsOBXService and avoid the complexity of dealing with two separate sections in the system Inbox and Outbox. Second, the input and output operations can be divided into two services, one inheriting from clsINBXService and one inheriting from clsOBXService. Third, the service can inherit directly from clsIOBXService and deal with both an Inbox section and an Outbox section at the same time. Both sections will have the same name as the service itself, and enabling one of them will automatically enable the other.

## Common #defines and typedefs

### Inbox/Outbox Service Status Codes

The inbox/outbox section associated with the service is empty. This status is returned by msgIOBXSvcNextDocument.

```c
#define stsIOBXSvcSectionEmpty MakeStatus(clsIOBXService, 101)
```

The outbox section associated with the service is not empty, but none of the document is ready for output. This status is returned by msgIOBXSvcNextDocument.

```c
#define stsIOBXSvcDocNotReady MakeStatus(clsIOBXService, 102)
```

### Outbox Service Standard Dialog Codes

```c
#define tagOBXSvcDocumentExists MakeDialogTag(clsOBXService, 0)
#define tagOBXSvcOutputPending MakeDialogTag(clsOBXService, 1)
```

### Inbox Service Standard Dialog Codes

```c
#define tagINBXSvcDocumentExists MakeDialogTag(clsINBXService, 0)
#define tagINBXSvcInputPending MakeDialogTag(clsINBXService, 1)
```

### Filesystem Attributes

The state of a document in the inbox/outbox.

```c
#define iobxsvcAttrDocState FSMakeFix32Attr(clsIOBXService, 1)
Enum32(IOBXVC_ATTR_DOC_STATE) {
    iobxsvcDocNotScheduled = 0, // Document hasn’t been scheduled
    iobxsvcDocOutputInProgress = 1, // Output started, not finished yet
    iobxsvcDocUserCancelled = 2, // Cancelled by user
    iobxsvcDocError = 3, // Unable to finish due to errors
    iobxsvcDocInputInProgress = 4, // Input started, not finished yet
    iobxsvcDocReserved5 = 5, // Reserved for future expansion
    iobxsvcDocReserved6 = 6, // Reserved for future expansion
    iobxsvcDocReserved7 = 7, // Reserved for future expansion
    iobxsvcDocReserved8 = 8, // Reserved for future expansion
    iobxsvcDocReserved9 = 9, // Reserved for future expansion
    iobxsvcDocReserved10 = 10, // Reserved for future expansion
    iobxsvcDocReserved11 = 11, // Reserved for future expansion
    iobxsvcDocReserved12 = 12, // Reserved for future expansion
    iobxsvcDocReserved13 = 13, // Reserved for future expansion
    iobxsvcDocReserved14 = 14, // Reserved for future expansion
    iobxsvcDocReserved15 = 15 // Reserved for future expansion
};
```
Class Messages

msgNewDefaults
Initializes the P_IOBXSVC_NEW structure to default values.
Takes P_IOBXSVC_NEW, returns STATUS. Category: class message.

Arguments

typedef struct IOBXSVC_SECTION_METRICS {
  BOOLEAN autoPoll; // True if svc should poll documents when
  // it's both enabled and connected.
  CLASS sectionClass; // Section Class. Must be clsNBToc or
  // a subclass of it, or objNull for none.
  U32 reserved[2]; // Reserved.
} IOBXSVC_SECTION_METRICS, *P_IOBXSVC_SECTION_METRICS;

typedef struct IOBXSVC_NEW_ONLY {
  IOBXSVC_SECTION_METRICS in; // Inbox section spec
  IOBXSVC_SECTION_METRICS out; // Outbox section spec
  U32 reserved[3];
} IOBXSVC_NEW_ONLY, *P_IOBXSVC_NEW_ONLY;

#define ioSvcNewFields
   serviceNewFields
   IOBXSVC_NEW_ONLY iobxsvc;

typedef struct IOBXSVC_NEW {
  ioSvcNewFields
} IOBXSVC_NEW, *P_IOBXSVC_NEW;

Zeroes out pArgs->iobxsvc.

msgNew
Creates a new inbox/outbox service object.
Takes P_IOBXSVC_NEW, returns STATUS. Category: class message.

Arguments

typedef struct IOBXSVC_NEW {
  ioSvcNewFields
} IOBXSVC_NEW, *P_IOBXSVC_NEW;

msgIOBXSvcSwitchIcon
Toggles the inbox or outbox icon (to empty or filled) if necessary.
Takes nothing, returns STATUS. Category: class message.

Comments

Check the content of the inbox or outbox notebook. For outbox, show the "filled" icon if any document is found. For inbox, show the "filled" icon if there is at least one document that has not been opened.

msgIOBXDocGetService
Gets the service name.
Takes P_IOBX_DOC_GET_SERVICE, returns STATUS. Category: class message.

Arguments

typedef struct IOBX_DOC_GET_SERVICE {
  OBJECT document; // In: document uid
  CHAR svcName[nameBufLength]; // Out: service name
} IOBX_DOC_GET_SERVICE, *P_IOBX_DOC_GET_SERVICE;

#define msgIOBXDocGetService MakeMsg(clsIOBXService, 2)
Get the name of the service associated with an inbox/outbox document. If the document has not been placed into an inbox/outbox section, stsFailed is returned.

Note that the document must be at the top level within an inbox/outbox section. That is, if the document is embedded in another document, stsFailed will be returned even if its embeddor is within an inbox/outbox section.

**msgIOBXDocInIOBox**

Checks if a document is in a section in the Inbox/Outbox notebook.

Takes P_IOBX_DOC_IN_IOBOX, returns STATUS. Category: class message.

```c
#define msgIOBXDocInIOBox MakeMsg(clsIOBXService, 3)
```

```c
typedef struct IOBX_DOC_IN_IOBOX {
    ANM_AUX_NOTEBOOK notebook; // In: Which notebook?
    UUID uid; // In: document uuid
    CLASS svcClass; // In: service class to check for
} IOBX_DOC_IN_IOBOX, *P_IOBX_DOC_IN_IOBOX;
```

**Messages Sent to an Outbox Service Instance**

**msgIOBXSvcMoveInDoc**

Moves a document into the outbox section.

Takes P_IOBXSVC_MOVE_COPY_DOC, returns STATUS.

```c
#define msgIOBXSvcMoveInDoc MakeMsg(clsIOBXService, 4)
```

```c
typedef struct IOBXSVC_MOVE_COPY_DOC {
    ANM_AUX_NOTEBOOK notebook; // In: Which notebook?
    FS_LOCATOR source; // In: Location of source document.
    U16 sequence; // In: Sequence number to move/copy // in front of.
} IOBXSVC_MOVE_COPY_DOC, *P_IOBXSVC_MOVE_COPY_DOC;
```

**Comments**

Superclass behavior is to move the document located at pArgs->source into the input/output queue associated with the inbox/outbox service. For example, set pArgs->sequence to 1 to move the document to the top of the queue. Set it to maxU16 to move the document to the bottom of the queue.

After the document is moved (or copied) to the input/output queue, it is considered to be in a state ready for input/output, even though the service may not be connected at the time. Client should not alter the document in any way once it has been moved to the input/output queue.

Subclasses can provide their own behavior if they wish. Remember to use the class message msgIOBXSvcSwitchIcon to change the inbox/outbox icon.

**msgIOBXSvcCopyInDoc**

Copies a document into the Inbox/Outbox section.

Takes P_IOBXSVC_MOVE_COPY_DOC, returns STATUS.

```c
#define msgIOBXSvcCopyInDoc MakeMsg(clsIOBXService, 5)
```
typedef struct IOBXSVC_MOVE_COPY_DOC {
    ANM_AUX_NOTEBOOK notebook; // In: Which notebook?
    FS_LOCATOR source; // In: Location of source document.
    U16 sequence; // In: Sequence number to move/copy in front of.
} IOBXSVC_MOVE_COPY_DOC, *P_IOBXSVC_MOVE_COPY_DOC;

Same as msgIOBXSvcMoveInDoc, except that the document is copied to the input/output queue.

**msgIOBXSvcGetTempDir**

Passes back a handle for a temporary directory.

Takes P_OBJECT, returns STATUS.

```c
#define msgIOBXSvcGetTempDir MakeMsg(clsIOBXService, 6)
```

This message is provided for clients who may want to prepare their input/output document before moving it into the input/output queue. The handle of an "official" temporary directory is passed back and it can be used as temporary storage for documents, data, etc. Clients are responsible for deleting temporary files they created when done. This temporary directory will be flushed after a warm boot.

**msgIOBXSvcPollDocuments**

Polls all documents waiting for input/output.

Takes nothing, returns STATUS.

```c
#define msgIOBXSvcPollDocuments MakeMsg(clsIOBXService, 7)
```

This message tells the inbox/outbox service to look through its queue and initiate the input/output process for the first document ready to do so. The service will first make sure that it is enabled and is connected to the designated input/output port. If these conditions are met, it will then self-send msgIOBXSvcNextDocument to locate the next document ready for input/output.

If msgIOBXSvcNextDocument returns stsOK, indicating that a document is ready, this message proceeds to self-send msgIOBXSvcLockDocument to lock the document, and finally msgIOBXSvcIOStart to initiate the input/output process.

If msgIOBXSvcNextDocument returns stsOXBSvcDocNot Ready, indicating that the section is not empty but none of the documents are ready for input/output, this message self-sends msgIOBXSvcScheduleDocument to schedule the document passed back in pArgs at a later time.

Subclasses normally do not process this message.

**See Also**

msgIOBXSvcNextDocument

**msgIOBXSvcNextDocument**

Passes back the next document ready for input/output.

Takes P_IOBXSVC_DOCUMENT, returns STATUS. Category: self-sent.

```c
#define msgIOBXSvcNextDocument MakeMsg(clsIOBXService, 8)
```

```c
typedef struct IOBXSVC_DOCUMENT {
    OBJECT uid; // uid of the doc
    OBJECT dir; // app dir of the doc
    OBJECT docClass; // class of the doc
    U16 sequence; // sequence of the doc
    CHAR pName[nameBufLength]; // name of this doc
    P_UNKNOWN pDocData; // subclass's private data
} IOBXSVC_DOCUMENT, *P_IOBXSVC_DOCUMENT;
```

If msgIOBXSvcNextDocument returns stsOK, indicating that a document is ready, this message proceeds to self-send msgIOBXSvcLockDocument to lock the document, and finally msgIOBXSvcIOStart to initiate the input/output process.

If msgIOBXSvcNextDocument returns stsOXBSvcDocNot Ready, indicating that the section is not empty but none of the documents are ready for input/output, this message self-sends msgIOBXSvcScheduleDocument to schedule the document passed back in pArgs at a later time.

Subclasses normally do not process this message.
Superclass behavior is to start from the top of the queue and locate the first document ready for input/output. If one is found, information about the document is passed back in `pArgs`. The same `pArgs` will be passed to messages `msgIOBXSvcLockDocument` and `msgIOBXSvcIOStart`. By default, a document is ready for input/output when it is closed. If the document is open, it will receive `msgIOBXSvcIOStartOK` and it should return `stsOK` to indicate that it is ready for input/output.

Subclasses can provide their own behavior if they wish. Return `stsOBXSvcSectionEmpty` to give the superclass an opportunity to change the inbox/outbox icon from filled to empty. Or refresh the look of the icon by sending `msgIOBXSvcSwitchIcon` to the service class.

**Return Value**
- `stsOK` A document is ready for input/output.
- `stsOBXSvcSectionEmpty` The input/output queue is empty.
- `stsOBXSvcDocNotReady` No document in the input/output queue is ready.

**See Also**
- `msgIOBXSvcPollDocuments`

---

**msgIOBXSvcLockDocument**

Locks the document in preparation for input/output.


```c
#define msgIOBXSvcLockDocument MakeMsg(clsIOBXService, 9)
```

**Message**

```c
typedef struct IOBXSVC_DOCUMENT {
    OBJECT uid;  // uid of the doc
    OBJECT dir;  // app dir of the doc
    OBJECT docClass;  // class of the doc
    UI6 sequence;  // sequence of the doc
    CHAR pName[nameBufLength];  // name of this doc
    P_UNKNOWN pDocData;  // subclass's private data
} IOBXSVC_DOCUMENT, *P_IOBXSVC_DOCUMENT;
```

**Comments**

This message is a place holder for subclasses that may require additional preparatory work to be performed on a document before it is ready for input/output. For example, a document may have to be "locked" so that it can not be opened during the input/output process. This message may be used for other purposes as well. For example, an inbox/outbox service may decide to store a light-weight "shadow" document (e.g., a report designator for a database application) in the input/output queue until it is chosen for input/output. The service then handles this message by converting the shadow document to a real one (e.g., the actual report).

The superclass behavior for this message is to stamp the document directory with the filesystem attribute `iobxsvcDocIOInProgress`. This stamp will prevent any gestures over the document from being processed. This means that once a document is locked for input/output it can not be deleted, renamed, etc. via gestures.

**See Also**
- `msgIOBXSvcUnlockDocument`

---

**msgIOBXSvcUnlockDocument**

Unlocks a document that was previously locked.


```c
#define msgIOBXSvcUnlockDocument MakeMsg(clsIOBXService, 10)
```
typedef struct IOBXSVC_DOCUMENT {
    OBJECT uid; // uid of the doc
    OBJECT dir; // app dir of the doc
    OBJECT docClass; // class of the doc
    U16 sequence; // sequence of the doc
    CHAR pName[nameBufLength]; // name of this doc
    P_UNKNOWN pDocData; // subclass's private data
} IOBXSVC_DOCUMENT, *P_IOBXSVC_DOCUMENT;

This message is a place holder for subclasses that may require additional "cleanup" work to be performed on a document before it is put back to the input/output queue.

The superclass behavior for this message is to remove the iobxsDocIOInProgress stamp on the document directory.

See Also
msgIOBXSvcLockDocument

---

**msgIOBXSvcScheduleDocument**

Schedules a document that is not ready for input/output

Takes P_IOBXSVC_DOCUMENT, returns STATUS. Category: self-sent.

#define msgIOBXSvcScheduleDocument MakeMsg(clsIOBXService, 11)

typedef struct IOBXSVC_DOCUMENT {
    OBJECT uid; // uid of the doc
    OBJECT dir; // app dir of the doc
    OBJECT docClass; // class of the doc
    U16 sequence; // sequence of the doc
    CHAR pName[nameBufLength]; // name of this doc
    P_UNKNOWN pDocData; // subclass's private data
} IOBXSVC_DOCUMENT, *P_IOBXSVC_DOCUMENT;

Comments
This message is sent when msgIOBXSvcNextDocument locates a document in the input/output queue but the document is not ready for input/output.

Subclasses should provide their own behavior. The default behavior is to release the ownership of the target service (i.e., become disabled), with the expectation that the user must manually schedule the document later on (by re-enabling the section.)

See Also
msgIOBXSvcNextDocument

---

**msgIOBXSvcIOStart**

Starts the input/output process for a document in the input/output queue.

Takes P_IOBXSVC_DOCUMENT, returns STATUS. Category: self-sent.

#define msgIOBXSvcIOStart MakeMsg(clsIOBXService, 12)

typedef struct IOBXSVC_DOCUMENT {
    OBJECT uid; // uid of the doc
    OBJECT dir; // app dir of the doc
    OBJECT docClass; // class of the doc
    U16 sequence; // sequence of the doc
    CHAR pName[nameBufLength]; // name of this doc
    P_UNKNOWN pDocData; // subclass's private data
} IOBXSVC_DOCUMENT, *P_IOBXSVC_DOCUMENT;

Comments
Superclass behavior is to activate the inbox/outbox document if it isn’t already active, and then send msgIOBXDocIOStart to the document instance.

Subclasses can provide their own behavior if they wish.
msgIOBXSvcIOCCancel
Cancels the input/output process.
Takes nothing, returns STATUS.
#define msgIOBXSvcIOCCancel MakeMsg(clsIOBXService, 13)

Comments
This message is sent to the service when the caller wishes to cancel any input/output operation in progress. The service responds to this message by sending msgIOBXDocOutuptCancel to an active inbox/outbox document. After the document is cancelled, the service will post an error note to the user if there are other documents waiting to be processed. The user then decides whether or not the service should proceed to send the remaining documents.

Subclasses do not normally process this message.

msgIOBXSvcIOCcleanUp
Cleans up after the current input/output is done.
Takes P_IOBX_DOC_OUTPUT_DONE, returns STATUS. Category: self-post..
#define msgIOBXSvcIOCcleanUp MakeMsg(clsIOBXService, 14)
Arguments
Enum32(IOBX_DOC_EXIT_BEHAVIOR) {
   iobxDocExitDoNothing = 0,
   iobxDocExitDelete = 1,
   iobxDocExitMarkAsFailed = 2,
   iobxDocExitMarkAsCancelled = 3
};
typedef struct IOBX_DOC_OUTPUT_DONE {
   IOBX_DOC_EXIT_BEHAVIOR behavior; // exit behavior
   P_UNKNOWN pDocData; // Unused: document specific data
} IOBX_DOC_OUTPUT_DONE, *P_IOBX_DOC_OUTPUT_DONE;

Comments
This message is posted to self as a result of the service receiving msgIOBXDocIODone, which is sent by the inbox/outbox document when it finishes the input/output operation. The inbox/outbox document will be either deleted or marked as specified in pArgs, and when everything is properly cleaned up the service will post msgIOBXSvcPollDocuments to self to see if anything else is waiting for input/output.

Subclasses do not normally process this message.

See Also
msgIOBXDocIODone

msgIOBXSvcStateChanged
Tells observers that the service state just changed.
Takes OBJECT, returns STATUS. Category: observer notification..
#define msgIOBXSvcStateChanged MakeMsg(clsIOBXService, 15)
Comments
Informs observers that the state of a service has just changed. pArgs is the UID of the service.

msgIOBXSvcQueryState
Passes back the state of the service.
Takes P_IOBXSVC_QUERY_STATE, returns STATUS.
#define msgIOBXSvcQueryState MakeMsg(clsIOBXService, 16)
typedef struct {
    BOOLEAN enabled;           // is the service enabled?
    CHAR status[nameBufLength]; // text describing the status of
                                // the service.
    CHAR docName[nameBufLength]; // document being processed
    PUNKNOWN pStateData;        // subclass's private data
} IOBXSVCS_QUERY_STATE, *P_IOBXSVCS_QUERY_STATE;

msgIOBXSvcGetEnabled

Gets the enabled state of the service.
Takes P_BOOLEAN, returns STATUS.
#define msgIOBXSvcGetEnabled MakeMsg(clsIOBXService, 17)

Subclasses can override this message and redefine the notion of "enabled." The default behavior of the superclass is to equate "enabled" with the ownership of the target service (i.e., input/output device). That is, the service is "enabled" when it owns the target service. By appending to or replacing the default behavior, a subclass can define additional conditions which must be met before a service is considered enabled.

msgIOBXSvcSetEnabled

Sets the enabled state of the service.
Takes BOOLEAN, returns STATUS.
#define msgIOBXSvcSetEnabled MakeMsg(clsIOBXService, 18)

This message is sent to the service in response to service notification messages msgSvcOwnerAcquired and msgSvcOwnerReleased. Subclasses can provide their own behavior and thereby redefine the notion of "enabled" for the service. If they do, they must pass this message up to the ancestor so that observers of the inbox/outbox service will be properly notified.

**Inbox/Outbox Document Messages**

msgIOBXDocIOStartOK

Asks the inbox/outbox document if it is OK to start the input/output process
Takes nothing, returns STATUS.
#define msgIOBXDocIOStartOK MakeMsg(clsIOBXService, 19)

When an inbox/outbox service finds an opened document in the inbox/outbox section, it sends this message to the document instance, asking whether it's OK to start the input/output operation while the document remains open. When the document receives this message, it should return stsOK to give the service permission to begin the input/output process. An error status, including stsNotUnderstood, is taken to mean that the document instance vetos the request and the service will not start the input/output process.

msgIOBXDocIOStart

Tells an inbox/outbox document to start the input/output process.
Takes nothing, returns STATUS.
#define msgIOBXDocIOStart MakeMsg(clsIOBXService, 20)
This message is sent by the inbox/outbox service to a document. The document should respond to this message by starting the input/output process.

### msgIOBXDocIODone

Tells the inbox/outbox service that input/output is finished.


```c
#define msgIOBXDocIODone MakeMsg(clsIOBXService, 21)
```

**Message**

```c
typedef struct IOBX_DOC_OUTPUT_DONE {
    IOBX_DOC_EXIT_BEHAVIOR behavior; // exit behavior
    P_UNKNOWN pDocData; // Unused: document specific data
} IOBX_DOC_OUTPUT_DONE, *P_IOBX_DOC_OUTPUT_DONE;
```

**Comments**

When the input/output process is finished, the inbox/outbox document in charge of the input/output should send this message to the inbox/outbox service. This message must be sent even if the input/output process has been aborted. The `pArgs` for this message tells the inbox/outbox service what to do with the inbox/outbox document. If `obxDocExitDelete` is specified, the document will be removed from the inbox/outbox. In all other cases the document will be unlocked and left in the inbox/outbox. If either `obxDocExitMarkAsCancelled` or `obxDocExitMarkAsFailed` are specified, the name of the document will be altered to provide visual indication for the user that the input/output process has not completed successfully.

**See Also**

msgIOBXDocGetService

### msgIOBXDocIOCancel

Tells an inbox/outbox document to cancel the input/output process.

Takes nothing, returns `STATUS`.

```c
#define msgIOBXDocIOCancel MakeMsg(clsIOBXService, 22)
```

**Comments**

This message is used by the inbox/outbox service to inform a document that it should cancel the input/output process. The document should handle this message by terminating its input/output operation and then sending `msgIOBXDocIODone` to the service with `pArgs->behavior` set to `obxDocExistMarkAsCancelled`.

### msgIOBXDocStatusChanged

Tells the inbox/outbox service that the document status is changed.


```c
#define msgIOBXDocStatusChanged MakeMsg(clsIOBXService, 23)
```

**Arguments**

```c
typedef struct IOBX_DOC_STATUS_CHANGED {
    CHAR status[nameBufLength]; // Text describing document state
    P_UNKNOWN pDocData; // Unused: document-specific data
} IOBX_DOC_STATUS_CHANGED, *P_IOBX_DOC_STATUS_CHANGED;
```

**Comments**

This message is sent by the inbox/outbox document to the service whenever its status has just changed. This status is displayed on Status column for the inbox/outbox section, in the Inbox/Outbox notebook.
Link layer API definition.

This file contains the interface definition for link layer protocols.

1. Link layer protocols must sub-class clsLink.
2. clsLink sub-classes clsService.

```c
#ifndef LINK_INCLUDED
#define LINK_INCLUDED
typedef struct {
    U16 addrSize; // size of address pointed to
    U8 addr[8];  // address
} ADDRESS, *P_ADDRESS;

The PROTOCOL_ADDRESS structure contains all the addressing information needed below the transport level. Unspecified addresses have null pointers.

typedef struct {
    } PROTOCOL_ADDRESS, *P_PROTOCOL_ADDRESS;

The PROTOCOL_INFO structures in the transmit and receive descriptors holds the following information.

typedef struct {
    PROTOCOL_ADDRESS src;
    PROTOCOL_ADDRESS dest;
} PROTOCOL_INFO;
#define sizeRxBuf 608
typedef struct RXBUFDESC {
    PROTOCOL_INFO info;
} RX_DESC, *P_RX_DESC;

typedef struct {
    U16 blockLen;
    U8 *pBlock;
} BLOCK, *P_BLOCK;
#define lnkMaxBlocks 8
#define sizeTxImmedData 32
typedef struct {
    PROTOCOL_INFO info;
    BLOCK txBlockTab[lnkMaxBlocks];
    U8 immedData[sizeTxImmedData];
} TX_DESC, *P_TX_DESC;
#define stsNoTxBuffer MakeStatus(clsLink, 1)
#define stsNoRxBuffer MakeStatus(clsLink, 2)
#define stsTxCollisionOrDefer MakeStatus(clsLink, 3)
#define stsTxTimeout MakeStatus(clsLink, 4)

// A power cycle has happened, the link should be closed and reinitialized
#define stsLinkPowerCycle MakeStatus(clsLink, 5)

// The link cable is not connected.
#define stsLinkNotConnected MakeStatus(clsLink, 6)
typedef U16 LINK_PROTOCOL_TYPE;
```
typedef enum
{
    linkMulticast = flag0,  // multicast transmit and receive
    linkBroadcast = flag1,  // broadcast transmit and receive
    linkPromiscuous = flag2, // promiscuous receive mode
    linkLoopback = flag3    // loopback of transmit to receive
} LINK SERVICES;

typedef struct
{
    U16 tableSize;
    U8  linkAddress[2];
} *P_BROADCAST_ADDR, *P_MULTICAST_ADDR;

typedef struct
{
    U16 tableSize;         // size of link Attributes table
    U8  typeName[32];     // ASCII name of LINK type: LocalTalk, Ethernet
                         // ASYNC, SDLC, etc.
    U16 linkAddrLen;      // length in bytes of link addresses
    U8  linkAddr[16];    // current link address of local station
    U32 linkSpeed;        // link communication speed in bits per second
    U16 maxDataSize;      // maximum amount of data that will fit in a link frame
    U16 maxFrameSize;     // maximum size of a link frame (including link header)
    U16 numBuffers;       // total number of available link buffers for this
                          // LINK services supported
    struct LINK SERVICES linkServices; // LINK services supported
    ADDRESS broadcast;    // broadcast address
    P_MULTICAST_ADDR pMulticastTable; // pointer to multicast address table
                          // add additional fields here
} LINK_ATTRIBUTES, *P_LINK_ATTRIBUTES;

typedef enum
{
    linkOperational,
    linkHardwareFailure,
    linkConfigurationFailure,
    linkHardwareNotInstalled
} LINK OPERATING_STATUS;

typedef struct
{
    LINK OPERATING STATUS linkStatus; // additional specific status info goes here
} LINK STATUS, *P_LINK_STATUS;

typedef void (EXPORTED * PF_PROTOCOL_HANDLER) (P_RX_DESC);

#pragma pack(l)      // byte boundary packing for protocol headers

typedef struct LINK_HEADER
{
    U8 destLinkAddr;
    U8 srcLinkAddr;
    U8 typeLink;
} *P_LINK_HEADER;

#define maxRxFrameSize sizeRxBuf

#pragma pack()       // back to command line stuff

#define lnkMaxShortFrameSize 10
typedef struct SHORT_TX_FRAME
{
  struct SHORT_TX_FRAME * link;
  BOOLEAN sent;
  U16 length;
  U32 physAddr;
  unsigned char buf[linkMaxShortFrameSize];
} SHORT_TX_FRAME, *P_SHORT_TX_FRAME;

msgLINKInstallProtocol
Install a link layer protocol handler to receive frames.
Takes P_INSTALL_PROTOCOL, returns STATUS.
#define msgLINKInstallProtocol MakeMsg(clslink, 1)
Arguments
typedef struct INSTALL_PROTOCOL{
  LINK_PROTOCOL_TYPE linkProtocolType;
  PF_PROTOCOL_HANDLER pNewHandler;
} INSTALL_PROTOCOL, *P_INSTALL_PROTOCOL;

msgLINKRemoveProtocol
Remove a link layer protocol handler.
Takes P_REMOVE_PROTOCOL, returns STATUS.
#define msgLINKRemoveProtocol MakeMsg(clslink, 2)
Arguments
typedef struct REMOVE_PROTOCOL{
  LINK_PROTOCOL_TYPE linkProtocolType;
} REMOVE_PROTOCOL, *P_REMOVE_PROTOCOL;

msgLINKTransmit
Transmit a packet.
Takes P_LINK_TRANSMIT, returns STATUS.
#define msgLINKTransmit MakeMsg(clslink, 5)
Arguments
typedef struct LINK_TRANSMIT {
  P_TX_DESC pTD;
} LINK_TRANSMIT, *P_LINK_TRANSMIT;

msgLINKBufferReturn
Return receive buffer to the link layer.
Takes P_BUFFER_RETURN, returns STATUS.
#define msgLINKBufferReturn MakeMsg(clslink, 6)
Arguments
typedef struct BUFFER_RETURN {
  P_RX_DESC pRD;
} BUFFER_RETURN, *P_BUFFER_RETURN;

msgLINKAttributesGet
Obtain the link layer attributes.
Takes P_ATTRIBUTES_GET, returns STATUS.
#define msgLINKAttributesGet MakeMsg(clslink, 7)
typedef struct ATTRIBUTES_GET {
    P_LINK_ATTRIBUTES pAttributes;
} ATTRIBUTES_GET, * P_ATTRIBUTES_GET;

**msgLINKStatusGet**

Obtain the link layer statistics.

Takes `P_STATUS_GET`, returns `STATUS`.

```c
#define msgLINKStatusGet MakeMsg( clsLink, 8 )
```

```c
typedef struct STATUS_GET {
    P_LINK_STATUS pStatus;
} STATUS_GET, * P_STATUS_GET;
```

**msgLINKAddressAcquire**

Acquire the link layer address.

Takes `P_ADDRESS_ACQUIRE`, returns `STATUS`.

```c
#define msgLINKAddressAcquire MakeMsg( clsLink, 9 )
```

```c
typedef struct ADDRESS_ACQUIRE {
    U16 linkAddrLen; // length in bytes of link addresses
    U8 linkAddr[16]; // current link address of local station
    BOOLEAN server; // acquire a server address
} ADDRESS_ACQUIRE, * P_ADDRESS_ACQUIRE;
```
This file contains the API for clsModem.

.clsModem inherits from clsService.

.clsModem provides the interface a client uses to communicate via a modem. The modem service is located, bound to, opened, and closed via standard PenPoint service messages.

The object which opens a modem service becomes its client. After opening a modem service, it is recommended that a client explicitly reset the modem firmware, initialize the modem I/O port settings, and then set the modem firmware to the desired state.

The modem firmware is reset by sending msgModemReset to an open modem service. Refer to msgModemReset below for a description of the state to which the modem firmware is reset.

A client obtains current modem I/O port settings by sending msgSioGetMetrics to a modem service. I/O port settings may be altered by sending msgSioSetMetrics to the modem service. These messages in addition to msgSioInit, msgSioBreakSend, msgSioControllInStatus, msgSioInputBufferSize, and msgSioInputBufferFlush are the only clsMILAsyncSIODevice messages which clsModem handles. Refer to file "sio.h" for a description of these messages.

After initializing the modem I/O port, a client may then send clsModem messages to initialize the modem to a desired state. Such initialization may be accomplished via discrete messages, or via msgSvcsSetMetrics.

Upon successfully initializing a modem, the client may then establish a connection, transmit data and/or receive data via the connection, and finally terminate the connection. Clients send clsStream messages to read/write data from/to the modem. Refer to file "stream.h" for a description of clsStream messages.

**** PLEASE NOTE ****

In a future release of PenPoint, the clsModem API will be augmented. Compatibility with the clsModem API described herein shall be maintained for at least one release.

Defined within this header file for the clsModem API

** Defines and Typedefs **

See Also

"service.h", "stream.h".

 ifndef MODERIM_INCLUDED
 define MODERIM_INCLUDED
 ifndef GO INCLUDED
 include <go.h>
 endif
 ifndef CLSMGR_INCLUDED
 include <clsmgr.h>
 endif
 ifndef SERVICE INCLUDED
 include <service.h>
 endif
 ifndef UID INCLUDED
 include <uid.h>
 endif
 ifndef DIALENV INCLUDED
 include <dialenv.h>
 endif

Observer Notification Messages

msgModemActivity
Notification sent to observers signifying changes in modem activity.
Takes MODEM_ACTIVITY, returns N/A. Category: observer notification.

```c
#define msgModemActivity MakeMsg(clsModem, 1)
```

Arguments
Enum32 (MODEM_ACTIVITY) { // The current modem activity/state.
    mdmOpened, // Modem service has been opened for use. *
    mdmResetting, // Currently being reset.
    mdmDialing, // Dialing a phone number. *
    mdmAwaitingConnection, // Awaiting a connection/answer.
    mdmConnected, // Connected with remote node. *
    mdmNegotiating, // Negotiating session/link parms.
    mdmSending, // Sending data.
    mdmReceiving, // Receiving data.
    mdmAnswering, // Answering a call. *
    mdmHangingUp, // Terminating the connection. *
    mdmDisconnected, // Connection terminated. *
    mdmClosed // Modem service has been closed. *
};
```

Comments
NOTE: A modem service needn't implement all observer notifications listed below. Those marked with an asterisk are the required minimum.

Client Notification Messages

msgModemResponse
Provides the modem's response to a command.
Takes MODEM_RESPONSE_INFO, returns N/A. Category: client notification.

```c
#define msgModemResponse MakeMsg(clsModem, 2)
```

Arguments
Enum32 (MODEM_RESPONSE) { // Modem response indications.
    mdmResOK, // OK - command accepted.
    mdmResUnrecognized, // Error - Unrecognized response from modem.
    mdmResError, // Error - Error response from modem.
    mdmResNoCarrier, // Error - No line carrier detected.
    mdmResNoDialTone, // Error - No phone dial tone detected.
    mdmResPhoneBusy, // Error - Phone line busy signal detected.
    mdmResNoAnswer, // Error - No one answered at the other end.
    mdmResInvalidFrame, // Error - Invalid frame detected.
    mdmResCRCError, // Error - Cyclic redundancy check error.
    mdmResRing, // Ring indication signal detected.
    mdmResConnect, // Connection established.
    mdmResConnect300, // 300 baud connection established.
    mdmResConnect600, // 600 baud connection established.
    mdmResConnect1200, // 1200 baud connection established.
    mdmResConnect2400, // 2400 baud connection established.
    mdmResConnect4800, // 4800 baud connection established.
    mdmResConnect9600, // 9600 baud connection established.
    mdmResConnect19200, // 19200 baud connection established.
    mdmResConnectReserved01, // Reserved for future expansion.
    mdmResConnectReserved02, // Reserved for future expansion.
    mdmResConnectReserved03, // Reserved for future expansion.
    mdmResConnectMNP, // MNP connection has been established.
    mdmResConnect1200MNP, // 1200 baud MNP connection established.
    mdmResConnect2400MNP, // 2400 baud MNP connection established.
    mdmResConnect1200LAPM, // 1200 baud LAPM connection established.
    mdmResConnect2400LAPM, // 2400 baud LAPM connection established.
    mdmResConnectReserved04, // Reserved for future expansion.
    mdmResConnectReserved05 // Reserved for future expansion.
};
```
typedef struct {
    U8 symbols[mdmSizeMaxResponse+1]; // Symbols comprising a response.
    MODEM_RESPONSE response; // Response meaning as enumerated above.
    U32 spare; // Reserved for future expansion.
} MODEM_RESPONSE_INFO, *P_MODEM_RESPONSE_INFO;

Provides the response to a previous modem request/command. msgModemResponse is only sent to the modem service’s client if the response behavior has been set to mdmResponseViaMessage (RE: msgModemSetResponseBehavior).

If a desired response isn’t available, then please contact GO Corporation to see that it gets added as a standard modem response. Thank you.

NOTE: The modem service depends upon the order in which the responses are defined.

---

**msgModemConnected**
Notification sent to the client indicating the modem has connected with a remote node modem.
Takes nothing, returns N/A. Category: client notification.

```
#define msgModemConnected MakeMsg(clsModem, 3)
```

Comments
A client may obtain information regarding the connection via msgModemGetConnectionInfo.

---

**msgModemDisconnected**
Notification sent to the client indicating that the current connection has been terminated.
Takes nothing, returns N/A. Category: client notification.

```
#define msgModemDisconnected MakeMsg(clsModem, 4)
```

---

**msgModemRingDetected**
Notification sent to the client indicating that a ring indication has been received from the modem.
Takes nothing, returns N/A. Category: client notification.

```
#define msgModemRingDetected MakeMsg(clsModem, 5)
```

---

**msgModemTransmissionError**
Notification sent to the client indicating that an error has been detected during transmission (sending or receiving) of data.
Takes nothing, returns N/A. Category: client notification.

```
#define msgModemTransmissionError MakeMsg(clsModem, 6)
```

Comments
This unsolicited message is typically sent as a result of detecting a data framing error, or other low-level modem link protocol generated error condition.

---

**msgModemErrorDetected**
Notification sent to the client indicating that an unexpected error indication has been received from the modem.
Takes nothing, returns N/A. Category: client notification.

```
#define msgModemErrorDetected MakeMsg(clsModem, 7)
```
### Action Messages

**msgModemSetResponseBehavior**

Set the modem response mode, and command-to-response time-out values.

Takes P_MODEM_RESPONSE_BEHAVIOR, returns STATUS. Category: modem service request.

```c
#define msgModemSetResponseBehavior MakeMsg(clsModem, 16)
```

**Arguments**

```c
Enum32 (MODEM_RESPONSE_MODE) { // Mode for conveying modem responses.
  mdmResponseViaStatus, // Report via status (Default).
  mdmResponseTransparent // Don't intercept and process modem responses.
};
#define mdmDefaultCommandTimeout 2500 // 2 1/2 second command timeout.
#define mdmDefaultConnectTimeout 30000 // 30 second connect timeout.
typedef struct {
  OS_MILLISECONDS timeoutCommand; // Command-to-response timeouts.
  // Timeout for all commands
  // except connect requests
  // (default of 2.5 seconds).
  OS_MILLISECONDS timeoutConnect; // Timeout for connect requests
  // (default of 30 seconds).
} MODERM_TIMEOUT, *P_MODEM_TIMEOUT;
```

**Comments**

Response mode mdmResponseViaStatus causes the modem service to block and await a response from the modem. If the modem doesn't return a response within the specified time-out duration, stsTimeOut is returned.

Response mode mdmResponseViaMessage is useful for clients that wish to ObjectPostAsync their modem service requests, and hence not block until completion (or timeout) of the request. Modem responses are reported to the client via msgModemResponse.

Response mode mdmResponseTransparent disables the modem service response processing sub-system. Modem command responses are left unaltered within the input data stream. The client assumes responsibility for processing modem responses. All commands successfully sent to the modem return a status of stsOK.

**NOTE:** Once a client switches to transparent mode (or sends modem register altering commands via msgModemSendCommand) they are responsible for the integrity of clsModem. Therefore, it is the client's responsibility to ensure that the clsModem (and the modem) are reset to a known state upon switching from transparent mode to a different response mode.

**msgModemGetResponseBehavior**

Passes back the current modem response mode, and the current command-to-response time-out values.

Takes P_MODEM_RESPONSE_BEHAVIOR, returns STATUS. Category: modem service request.

```c
#define msgModemGetResponseBehavior MakeMsg(clsModem, 17)
```

**Arguments**

```c
typedef struct {
  // Modem command-response handling behavior.
  // Mode for conveying responses
  // Command-to-response timeouts.
} MODERM_TIMEOUT, *P_MODEM_TIMEOUT;
```
msgModemSendCommand

Sends a specified command to the modem.

Takes P_MODEM_SEND_COMMAND, returns STATUS. Category: modem service request.

```c
#define msgModemSendCommand MakeMsg(clsModem, 18)
#define mdmSizeMaxCommand 80 /* Max command size is 80 bytes. */
```

**Arguments**

typedef struct {
    P U8 pCmdStr; /* In: Ptr to command string (null terminated). */
    OS_MILLISECONDS timeout; /* In: Timeout for cmd response. */
    MODEM_RESPONSE_INFO responseInfo; /* Out: The response to the cmd. */
} MODEM_SEND_COMMAND, *P_MODEM_SEND_COMMAND;

**Comments**

The timeout value specified within MODEM_SEND_COMMAND supersedes that specified via msgModemSetResponseBehavior.

NOTE: Clients should only use msgModemSendCommand to perform modem actions unavailable via the clsModem API described herein.

NOTE: Clients that send commands that alter modem registers are responsible for the integrity of clsModem. Therefore, it is the client's responsibility to ensure that such commands will not adversely affect clsModem.

msgModemGetConnectionInfo

Passes back information about the current connection.

Takes P_MODEM_CONNECTION_INFO, returns STATUS. Category: modem service request.

```c
#define msgModemGetConnectionInfo MakeMsg(clsModem, 19)
```

**Arguments**

```c
Enum32 (MODEM_CONNECTION) {
    mdmConnectionNone, /* None; Disconnected. */
    mdmConnectionStandard, /* Standard data. */
    mdmConnectionMNP, /* MNP. */
    mdmConnectionLAPM /* LAPM. */
};
```

```c
Enum32 (MODEM_LINK_CONTROL) {
    mdmLinkControlMNPClass1_4 = flag0, /* MNP Levels 1 through 4. */
    mdmLinkControlMNPClass5 = flag1, /* MNP Level 5 data compression. */
    mdmLinkControlMNPClass6 = flag2, /* MNP Level 6. */
    mdmLinkControlMNPClass7 = flag3, /* MNP Level 7 data compression. */
    mdmLinkControlV42 = flag4, /* Physical level error detection and correction (LAPM link control). */
    mdmLinkControlV42bis = flag5 /* V42 data compression. */
};
```

typedef struct {
    MODEM_CONNECTION connection; /* The type of connection. */
    MODEM_LINK_CONTROL linkControl; /* Link control in use, if any/known. */
    S32 baudRate; /* Baud rate of connection. */
    U32 spare[2]; /* Reserved for future expansion. */
} MODEM_CONNECTION_INFO, *P_MODEM_CONNECTION_INFO;

msgModemReset

Resets the modem firmware, I/O port, and service state.

Takes nothing, returns STATUS. Category: modem service request.

```c
#define msgModemReset MakeMsg(clsModem, 20)
```
NOTE: The modem I/O port baud rate is reset to the highest supported data mode baud rate. Therefore not all implementations will reset the baud rate to 2400. The client may elect to subsequently change the baud rate for auto-baud detecting modems.

Reset I/O port state:

```cpp
type baud = 2400;
line.dataBits = sioEightBits;
line.stopBits = sioOneStopBit;
line.parity = sioNoParity;
controlOut.rts = true;
controlOut.dtr = true;
flowChar.xonChar = 0x11;
flowChar.xoffChar = 0x13;
flowType.flowControl = sioNoFlowControl;
```

Reset modem firmware state:

Speaker control on until carrier detected (*). volume medium (*). detection enabled (*). detection enabled (*). mode from dialing environment. disabled. on ring zero. character echo disabled. command result codes. verbal result codes (words). carrier upon connect. code + (ASCII 43). termination code carriage return (ASCII 13).

(*) or set as per current modem option card setting.

---

### `msgModemOffHook`

Picks up the phone line.

Takes nothing, returns STATUS. Category: modem service request.

```cpp
#define msgModemOffHook (clsModem, 21)
```

### `msgModemOnline`

Forces the modem online into data mode.

Takes nothing, returns STATUS. Category: modem service request.

```cpp
#define msgModemOnline (clsModem, 22)
```

### `msgModemSetDialType`

Establishes the mode for dialing telephone numbers.

Takes MODEM_DIAL_MODE, returns STATUS. Category: modem service request.

```cpp
#define msgModemSetDialType (clsModem, 23)
```

```cpp
Enum32 (MODEM_DIAL_MODE) {
  mdlDialPulseDialing,
  mdlDialTouchtoneDialing,
  mdlDialStringDialing,
  mdlDialEnvironmentDialing
};
```
**msgModemDial**

Performs dialing and attempts to establish a connection.

Takes `P_MODEM_DIAL`, returns `STATUS`. Category: modem service request.

```c
#define msgModemDial MakeMsg(clsModem, 24)
```

**Arguments**

typedef struct {
  DIALENV.DialString dialString;  // Dialing and connection type.
  U32 spare[2];  // In: Phone number to dial.
  } MODEM_DIAL, *P_MODEM_DIAL;

**msgModemSetAutoAnswer**

Disables or enables the modem auto-answer feature.

Takes `P_MODEM_SET_AUTO_ANSWER`, returns `STATUS`. Category: modem service request.

```c
#define msgModemSetAutoAnswer MakeMsg(clsModem, 25)
```

**Arguments**

```c
Enum32 (MODEM_AUTO_ANSWER) {
  mdmAutoAnswerDisabled, // Disable auto-answer (Default).
  mdmAutoAnswerEnabled // Enable auto-answer.
};
```

```c
typedef struct {
  MODEM.AUTO_ANSWER autoAnswer; // Enable/disable auto-answer.
  S32 rings; // Number of rings before answer.
} MODEM_SET_AUTO_ANSWER, *P_MODEM_SET_AUTO_ANSWER;
```

**Comments**

NOTE: For some modems a value of 0 for rings disables auto-answer.

**msgModemSetAnswerMode**

Filters the type of calls to answer and connection reporting.

Takes `MODEM_ANSWER_MODE`, returns `STATUS`. Category: modem service request.

```c
#define msgModemSetAnswerMode MakeMsg(clsModem, 26)
```

```c
Enum32 (MODEM_ANSWER_MODE) {
  mdmAnswerDataMode, // Answer in data mode.
  mdmAnswerFaxMode, // Answer in fax mode.
  mdmAnswerVoiceMode, // Answer in voice mode.
};
```

**Comments**

NOTE: Not all modems are capable of discriminating between the type of incoming call.

**msgModemAnswer**

Immediately answers a telephone call.

Takes nothing, returns `STATUS`. Category: modem service request.

```c
#define msgModemAnswer MakeMsg(clsModem, 27)
```

**msgModemHangUp**

Hang-ups and disconnects to terminate a connection.

Takes nothing, returns `STATUS`. Category: modem service request.

```c
#define msgModemHangUp MakeMsg(clsModem, 28)
```
msgModemSetSignallingModes

Establishes/restricts the modem to use specific signalling modes/standards.

Takes P_MODEM_SIGNALLING_MODES, returns STATUS. Category: modem service request.

```
#define msgModemSetSignallingModes MakeMsg(clsModem, 29)
```

**Arguments**

```
Enum32 (MODEM_SIGNALLING_VOICEBAND) { // Voice-band signalling standards.
  mdmVoiceBandBell103J = flag0, // 300 BPS.
  mdmVoiceBandBell212A = flag1, // 1200 BPS.
  mdmVoiceBandV21 = flag2, // 300 BPS duplex modem on GSTN.
  mdmVoiceBandV22 = flag3, // 1200 BPS duplex modem on GSTN
  or P-P leased two-wire circuits.
  mdmVoiceBandV22bis = flag4, // 2400 BPS duplex modem on GSTN
  or P-P two-wire leased circuits.
  mdmVoiceBandV23 = flag5, // 600/1200 BPS modem on GSTN.
  mdmVoiceBandV26 = flag6, // 2400 BPS modem on four-wire
  or P-P two-wire leased circuits.
  mdmVoiceBandV26bis = flag7, // 2400/1200 BPS modem on GSTN.
  mdmVoiceBandV26ter = flag8, // 2400 BPS modem on four-wire
  or P-P two-wire leased circuits.
  mdmVoiceBandV27 = flag9, // 4800 BPS on leased circuits.
  mdmVoiceBandV27bis = flag10, // 2400/4800 BPS on leased circuits.
  mdmVoiceBandV27ter = flag11, // 4800/2400 BPS modem on GSTN.
  mdmVoiceBandV29 = flag12, // 9600 BPS FDX or HDX modem on
  or P-P four-wire leased circuits.
  mdmVoiceBandV32 = flag13, // 9600/4800 BPS duplex modem on
  GSTN or leased circuits.
  mdmVoiceBandV33 = flag14, // 14400 BPS modem on P-P
  four-wire leased circuits.
};
Enum32 (MODEM_SIGNALLING_WIDEBAND) { // Wide-band signalling standards.
  mdmWideBandV35 = flag0, // 48 Kbps data transmission on
  mdmWideBandV36 = flag1, // 48-72 Kbps sync data transmission
  mdmWideBandV37 = flag2 // 72-168 Kbps sync data transmission
  on 60-108 KHz group band circuits.
};
typedef struct {
  // Modem modulation/signalling modes.
  MODEM_SIGNALLING_VOICEBAND voiceBand; // Voice band signalling.
  MODEM_SIGNALLING_WIDEBAND wideBand; // Wide band signalling.
} P_MODEM_SIGNALLING_MODES;
```

**Comments**

NOTE: Not all modems provide support for selecting signalling modes.

msgModemSetToneDetection

Enables or disables busy tone and/or dial tone detection.

Takes MODEM_TONE_DETECTION, returns STATUS. Category: modem service request.

```
#define msgModemSetToneDetection MakeMsg(clsModem, 30)
```

**Arguments**

```
Enum32 (MODEM_TONE_DETECTION) { // Busy and dial tone carrier signal on/off.
  mdmToneDetectDisable, // Detect neither busy tone or dial tone.
  mdmToneDetectBusyOnly, // Detect busy tone, but not dial tone.
  mdmToneDetectDialOnly, // Detect dial tone, but not busy tone.
  mdmToneDetectBusyAndDial // Detect dial tone and busy tone (Default).
};
```
**msgModemSetSpeakerControl**

Enables, disables and controls modem speaker behavior.

Takes `MODEM_SPEAKER_CONTROL`, returns STATUS. Category: modem service request.

```c
#define msgModemSetSpeakerControl MakeMsg(clsModem, 31)
```

```c
Enum32 (MODEM_SPEAKER_CONTROL) { // Specifies the modem speaker behavior.
    mdmSpeakerOn, // Speaker is always on.
    mdmSpeakerOff, // Speaker is always off.
    mdmSpeakerOnConnectOff // Speaker on until carrier detected (Default).
};
```

**msgModemSetSpeakerVolume**

Sets the volume of the modem speaker.

Takes `MODEM_SPEAKER_VOLUME`, returns STATUS. Category: modem service request.

```c
#define msgModemSetSpeakerVolume MakeMsg(clsModem, 32)
```

```c
Enum32 (MODEM_SPEAKER_VOLUME) { // Specifies the modem speaker volume.
    mdmSpeakerVolumeWhisper, // Lowest volume level.
    mdmSpeakerVolumeLow, // Low/reasonable volume level.
    mdmSpeakerVolumeMedium, // Normal/average volume level (Default).
    mdmSpeakerVolumeHigh // Highest volume level.
};
```

**msgModemSetCommandState**

Sets the modem into command mode.

Takes nothing, returns STATUS. Category: modem service request.

```c
#define msgModemSetCommandState MakeMsg(clsModem, 33)
```

**msgModemSetDuplex**

Sets the duplex mode for inter-modem communications while on-line.

Takes `MODEM_DUPLEX_MODE`, returns STATUS. Category: modem service request.

```c
#define msgModemSetDuplex MakeMsg(clsModem, 34)
```

```c
Enum32 (MODEM_DUPLEX_MODE) { // Indicates data transmission line duplex.
    mdmDuplexHalf, // Data transmitted in one direction at a time (the line must be turned around).
    mdmDuplexFull // Data can be transmitted in both directions simultaneously (Default).
};
```

**msgModemSetMNPMode**

Sets the MNP mode of operation.

Takes `MODEM_MNP_MODE`, returns STATUS. Category: modem service request.

```c
#define msgModemSetMNPMode MakeMsg(clsModem, 35)
```
Part 10 / Connectivity

Enum32 (MODEM_MNP_MODE) {
    mdmMNPModeDirect, // MNP mode in which modem is to operate.
    mdmMNPModeReliable, // Disable MNP mode (default).
    mdmMNPModeAutoReliable, // Both modems must support MNP levels
    // 1-4 (5 if enabled) before a connection
    // can be made.
    mdmMNPModeLAPM, // Attempt to establish an MNP connection; if
    // it fails establish a direct connection.
};

NOTE: Not all modems provide MNP support.

msgModemSetMNPCompression
Sets MNP class 5 compression on or off.
Takes MODEM_MNP_COMPRESSION, returns STATUS. Category: modem service request.

#define msgModemSetMNPCompression MakeMsg(clsModem, 36)

Enum32 (MODEM_MNP_COMPRESSION) { // Type of compression to use in MNP mode.
    mdmMNPCompressionOff, // Disable MNP level 5 compression (default).
    mdmMNPCompressionOn // Enable MNP level 5 compression.
};

msgModemSetMNPBreakType
Specify how a break is handled in MNP mode.
Takes MODEM_MNP_BREAK_TYPE, returns STATUS. Category: modem service request.

#define msgModemSetMNPBreakType MakeMsg(clsModem, 37)

Enum32 (MODEM_MNP_BREAK_TYPE) { // How breaks are handled in MNP mode.
    mdmMNPSendNoBreak, // Do not send break to remote modem.
    mdmMNPEmptyBuffersThenBreak, // Empty data buffers before sending break.
    mdmMNPImmediatelySendBreak, // Send break when received (default).
    mdmMNPSendBreakInSequence // Send break relative to data to be sent.
};

msgModemSetMNPFlowControl
Specify the flow control to use in MNP mode.
Takes MODEM_MNP_FLOW_CONTROL, returns STATUS. Category: modem service request.

#define msgModemSetMNPFlowControl MakeMsg(clsModem, 38)

Enum32 (MODEM_MNP_FLOW_CONTROL) { // Indicates the flow control for MNP mode.
    mdmMNPFlowControlDisable, // No flow control used (default).
    mdmMNPFlowControlXonXoff, // Use Xon/Xoff flow control.
    mdmMNPFlowControlHardware // Use RTS/CTS flow control.
};
Superclass Messages

msgSvcGetMetrics

Passes back the current modem metrics.

Takes P_SVC_GET_SET_METRICS, returns STATUS. Category: superclass message.

Arguments
typedef struct MODEM_METRICS {
    MODEM.DialMode mdrnDialMode;
    MODEM.DuplexMode mdrnDuplexMode;
    MODEM.SpeakerControl mdrnSpeakerControl;
    MODEM.SpeakerVolume mdrnSpeakerVolume;
    MODEM.ToneDetection mdrnToneDetection;
    MODEM.AnswerMode mdrnAnswerMode;
    MODEM.AutoAnswer mdrnAutoAnswer;
    U32 mdrnAutoAnswerRings;
    MODEM.MNPMode mdrnMNPMode;
    MODEM.MNPCompression mdrnMNPCompression;
    MODEM.MNPBreakType mdrnMNPBreakType;
    MODEM.MNPFlowControl mdrnMNPFlowControl;
} MODEM_METRICS, *P_MODEM_METRICS;

Comments
The pMetrics field of SVC_GET_SET_METRICS is expected to point to a buffer capable of receiving MODEM_METRICS as described below.

msgSvcSetMetrics

Sets current modem metrics, and re-initializes the modem with specified metrics.

Takes P_SVC_GET_SET_METRICS, returns STATUS. Category: superclass message.

Comments
The pMetrics field of SVC_GET_SET_METRICS is expected to point to a buffer containing MODEM_METRICS as described above.

msgSvcCharacteristicsRequested

Passes back the characteristics of the modem service.

Takes P_SVC_CHARACTERISTICS, returns STATUS. Category: superclass message.

#define mdmHWManufactureNameLength 15
#define mdmHWModelNameLength 15

typedef struct {
    CHAR name[mdmHWManufactureNameLength+1];
} MODEM_HARDWARE_MANUFACTURER, *P_MODEM_HARDWARE_MANUFACTURER;

typedef struct {
    CHAR name[mdmHWModelNameLength+1];
} MODEM_HARDWARE_MODEL, *P_MODEM_HARDWARE_MODEL;

Enum32 (MODEM_HARDWARE_FEATURES) {
    mdrnHWCapAutoDial = flag0, // Auto dialing.
    mdrnHWCapAutoAnswer = flag1, // Auto answer.
    mdrnHWCapAutoBaudDetect = flag2, // Auto baud detection.
    mdrnHWCapCallTypeDiscrimination = flag3, // Call type discrimination
    mdrnHWCapPhoneJackConnectDetect = flag4, // Phone jack connect and disconnect event reporting.
    mdrnHWCapRingSignalMachineWakeUp = flag5 // Ring signal detection
};
typedef struct {
  S32 sizeInputBuffer;  // Size of internal modem I/O buffers.
  S32 sizeOutputBuffer;  // Input buffer size.
  MODM_HARDWARE_BUFFERS, *P_MODM_HARDWARE_BUFFERS;
} Enum32(MODM_DCE_CONTROL) {
  mdmDCEControlAT = flag0  // Firmware DCE protocol/command sets.
                           // Hayes 'AT' commands.
};

typedef struct MODEM_CHARACTERISTICS {  // Modem hw & sw characteristics.
  MODM_HARDWARE_MANUFACTURER hardwareManufacturer;
  MODM_HARDWARE_MODEL hardwareModel;
  MODM_HARDWARE_FEATURES hardwareFeature;
  MODM_HARDWARE_BUFFERS hardwareBuffer;
  MODM_DCE_CONTROL dceControl;
  MODM_SIGNALLING_MODES signallingMode;
  MODM_LINK_CONTROL linkControl;
  U32 spare[4];
} MODEM_CHARACTERISTICS, *P_MODEM_CHARACTERISTICS;

Comments
The pBuf field of SVC_CHARACTERISTICS is expected to point to a buffer capable of receiving
MODEM_CHARACTERISTICS as described below.
Implementors of clsModem services that wish to provide capabilities not described within
MODEM_CHARACTERISTICS should contact GO Corporation to ensure such clsModem enhancements
are standardized and noted within MODEM_CHARACTERISTICS. Thank you.

Class Messages

msgNew
Creates a new instance of a modem service.
Takes P_MODEM_NEW, returns STATUS. Category: class message.

#define modemNewFields serviceNewFields
typedef struct MODEM_NEW {
  modemNewFields
  } MODEM_NEW, *P_MODEM_NEW;

Arguments
typedef struct MODEM_NEW {
  modemNewFields
  } MODEM_NEW, *P_MODEM_NEW;

Comments
Error Return Values: percolated up from other classes,

msgNewDefaults
Initializes the MODEM_NEW structure to default values.
Takes P_MODEM_NEW, returns STATUS. Category: class message.

Message
typedef struct MODEM_NEW {
  modemNewFields
  } MODEM_NEW, *P_MODEM_NEW;

Comments
Sets:
  pArgs->svc.style.autoOption =
  pArgs->svc.style.exclusiveOpen = true;
  pArgs->svc.style.waitForTarget = false;
  pArgs->svc.pManagerList = pManagerList;
  pArgs->svc.numManagers = sizeof(pManagerList)/sizeof(OBJECT);
  static OBJECT pManagerList[] =
    {
      theModems   // clsModem is one of theModems.
    };

static OBJECT pManagerList[] =
  {
    theModems   // clsModem is one of theModems.
  };

// clsModem is one of theModems.
**clsModem error status values**

This modem service doesn't (or cannot) support the current request due to hardware or firmware constraints.

```c
#define stsModemNotSupported MakeStatus(clsModem, 1)
```

A request to the modem service contained a parameter that is invalid.

```c
#define stsModemBadParameter MakeStatus(clsModem, 2)
```

The size of the buffer supplied to get/set modem service metrics or characteristics is incorrect.

```c
#define stsModemBufferSizeError MakeStatus(clsModem, 3)
```

The modem service was unable to find and/or open its target service.

```c
#define stsModemTargetException MakeStatus(clsModem, 4)
```

The modem service is not open. The current request requires that it be open.

```c
#define stsModemNotOpen MakeStatus(clsModem, 5)
```

The modem has responded to a modem command with an error response.

```c
#define stsModemErrorResponse MakeStatus(clsModem, 6)
```

The modem has responded to a modem command with a response that was unrecognized.

```c
#define stsModemUnrecognizedResponse MakeStatus(clsModem, 7)
```

The modem responded with a notification of carrier loss after dialing, attempting to go online, or being online.

```c
#define stsModemNoCarrier MakeStatus(clsModem, 8)
```

The modem didn't detect a dial tone while dialing to establish a connection.

```c
#define stsModemNoDialTone MakeStatus(clsModem, 9)
```

The modem didn't detect an answer tone after dialing to establish a connection.

```c
#define stsModemNoAnswer MakeStatus(clsModem, 10)
```

The modem has been unable to successfully transmit a data frame to the remote node.

```c
#define stsModemTransmitError MakeStatus(clsModem, 11)
```

The modem has been unable to successfully receive a data frame to the remote node.

```c
#define stsModemReceiveError MakeStatus(clsModem, 12)
```

The modem has detected a cyclic redundancy check error within a data frame received from the remote node.

```c
#define stsModemCRCError MakeStatus(clsModem, 13)
```

The modem has detected a busy signal after dialing a telephone number.

```c
#define stsModemLineBusy MakeStatus(clsModem, 14)
```

The modem service could not locate a window within one of its option cards. This is an internal error.

```c
#define stsModemNoSuchWindow MakeStatus(clsModem, 255)
```

**clsModem non-error status values**

None currently defined
This file contains the API definition for clsOBXService. clsOBXService inherits from clsIOBXService. Provides default behavior for Outbox Services.

```c
#ifndef OBXSVC_INCLUDED
#define OBXSVC_INCLUDED
#ifndef IOBXSVC_INCLUDED
#include <iobxsvc.h>
#endif
#endif
```

## 1. Introduction

In PenPoint, output operations are handled by a special class of services known as the "outbox services." An outbox service implements the "deferred output" feature in PenPoint: This concept permits a user to specify output operations regardless of the readiness of output devices. If the output device (e.g., a printer, a phone plug, a LAN connection, etc.) is not available or not connected, documents waiting for output will be placed into an "output queue" associated with the output service. (This output queue is a special section in the system Outbox notebook.) Thus, the actual output process is deferred until the output device becomes ready.

### The Target of an Outbox Service

PenPoint expects that the PenPoint computer will not always be attached to most output devices. Therefore, the output process for any PenPoint documents will be deferred until a connection is established. The software controlling an input/output device is often implemented as an I/O service. In most cases, an outbox service will make such an I/O service as its "target." (See service.h for more information about target services in general.) Examples of I/O services include drivers for serial ports, parallel ports, data and/or fax modems, and LAN servers. By making an I/O service its target, an outbox service is notified whenever the physical output device becomes connected or disconnected. When an outbox service is not actively sending out a document, the connection status of the device is displayed in the "Status" column of the Outbox notebook Table of Contents.

### Enabling and Disabling an Outbox Service

An outbox service must be "enabled" before its output process can begin. This enabled state is represented by a checkbox in the "Enabled?" column of the Outbox notebook TOC. Typically, an output device permits only exclusive access. If multiple outbox services are connected to the same output device, only one can be enabled at a time. Enabling an outbox service causes it to become the "owner" of its target service. The service remains "enabled" until either it is manually disabled by the user (i.e., by unchecking the "Enabled?" box); or until it willingly releases ownership of the device so that another service can become the new owner. For more details on the notion of service ownership, see the service API in service.h.
The concept of enabling or disabling an outbox service also provides a convenient mechanism for the user to manage an output device that can not automatically determine whether or not it becomes connected or disconnected. Because the outbox service will not be informed when its target service is connected or disconnected, its status will always remain "Connected" regardless of the connection status of the physical device. Such services can be explicitly disabled to prevent documents from being sent to a device that is not ready for output.

Managing the Output Process via the Outbox Service Protocol

Each instance of an outbox service has a corresponding section in the system Outbox notebook. The name of the service and the name of the section are the same. For example, the user may create two instances of an outbox service class named "DotMatrix," say "Engineering Printer" and "Upstairs." Each instance will have its own output queue, implemented as a section called "Engineering Printer" and "Upstairs" in the outbox notebook. The primary function of an outbox service is to manage the output queue for each service instance. This function is implemented by a standard outbox protocol consisting of 8 inter-related messages, as summarized below:

The client of an outbox service first sends msgOBXSvcMoveInDoc or msgOBXSvcCopyInDoc to the outbox service instance, telling it to add an existing PenPoint document to its output queue. Once a document is added to the outbox, msgOBXSvcPollDocuments informs an outbox service that it should check to see if conditions are right to start an output process. Other events may also cause the outbox service to receive msgOBXSvcPollDocument. For example, an outbox service will self-send this message when the service has just been enabled. If the service is enabled and the output device is connected, the service sends msgOBXSvcNextDocument to self to locate the next document ready for output. If a document exists in the output queue but is not ready for output, the service self-sends msgOBXSvcScheduleDocument to reschedule output at a later time. If a document is ready for output, the service will lock the document with msgOBXSvcLockDocument, and kick off the output process with msgOBXSvcOutputStart. At the end of the output process, the document being sent will send msgOBXDocOutputDone to the outbox service. Finally, if the output finished normally, the service self-sends msgOBXSvcUnlockDocument to restore the document to its "pre-output" state.

Outbox Documents

The primary focus of an outbox service is to manage its output queue. An output queue is essentially a collection of documents located in an outbox section. The primary focus of an outbox document is to manage a single output job.

An outbox document can be any PenPoint document, i.e., an instance of an application inheriting from clsApp. It can be created, opened, and closed just like a regular page in the notebook. An example of an outbox document would be an "address envelope" for an electronic mail service.

An outbox document is also responsible for interacting with the outbox service and controlling the output process, such as sending out an electronic mail message through a communication link. Thus, in addition to responding to clsApp messages, an outbox document also understand the following clsOBXService messages:

msgOBXDocOutputStartOK

For details see the description for each message.
Writing Your Own Outbox Service

clsOBXService is an abstract class. You should always create a subclass of it. This is because clsOBXService only manages the output queue, it does not actually cause the output to happen. Typically, your outbox service will inherit the output queue management behavior from clsOBXService, and add any service-specific behaviors for the communication protocol or devices you need to handle.

The default behavior of the outbox service does not support sophisticated scheduling algorithms that may be required by some services. However, it is not difficult to replace some default behaviors with new ones. The messages you may want to handle on your own include:

msgOBXSvcMoveInDoc

For example, the default behavior of msgOBXSvcNextDocument treats the output queue as a simple, First-In-First-Out queue. If this is not sufficient for the service you wish to develop, you can provide your own behavior and pass back a document not on the top of the queue, or even a document not located in the Outbox notebook if it makes sense for the service.

Another example would be msgOBXSvcLockDocument and msgOBXSvcUnlockDocument. Their default behavior is to mark the document so that gestures over the document icon will not be recognized while output is in progress. A msgOBXSvcUnlockDocument typically indicates that the output has been aborted for some reason. You may wish to add to the default behavior, such as notifying your observers that some error has just occurred.

For details see the description for each message.

Working with Existing Outbox Services

As explained before, all output operations should be performed through an outbox service in order to take advantage of the "deferred output" feature of PenPoint. An application or a service can "bypass" the standard outbox protocol only if the output device is always present or is rarely detached from the PenPoint computer.

The key to working with an existing outbox service is to conceptually break up the output process into two distinct phases. The first phase is either adding an existing PenPoint document to the output queue, or creating a special document of some sort in temporary storage and then move it into the output queue. The second phase is the actual output process, during which a device-specific data stream is sent out via some communication link. clsOBXService provides a framework for managing the transition from one phase to another.

The separation of these two phases of output operation has an additional benefit. In many cases, an application developer can avoid writing a new outbox service in order to handle application-specific output functions. It is often sufficient to handle only one of the two phases of the output operation.

There are several options, as explained below:

One inexpensive solution is to have the application export the data into a format that is easier to output under an existing outbox service. For example, a database document can generate a report as an ASCII file or a word processor document and move it into a printer, fax or e-mail outbox section. Similarly, a spreadsheet document can export its pie chart into a popular drawing program document and move it to the outbox for output.

Another approach is to allow the database or spreadsheet document itself to be moved or copied into the output queue. When the document receives msgOBXSvcOutputStart, it knows that the output device is ready. It then proceeds to perform the output operation the old-fashioned way. This alternative may be an attractive one if we wish to port an existing PC application to PenPoint. Such applications already
have sophisticated output capabilities, and we only need to ensure not to start the output process until the device is ready. The obvious disadvantage of this approach is that it requires additional memory if we have to make a copy of the document in order to put it into the outbox.

A third approach represents a compromise between the two. During the first phase of the output operation, a "surrogate" document, rather than the real one, is copied into the output queue. This surrogate document not only understands the outbox output protocol, but also knows how to communicate with the original document. It is effectively a "pointer" back to the original document. When the output process begins, the surrogate document communicates with the original one to cause the device-specific data stream to be sent to the correct output port.

**Services that Handle Input and/or Output**

clsOBXService deals only with output operations. For those services that want to handle input operations, a similar class clsINBXService is provided by PenPoint. If a service (e.g., an electronic mail service) wants to handle both input and output, another abstract class, clsIOBXService, is provided. clsIOBXService associates the service with both an input queue and an output queue. (The input queue is a section in the system Inbox notebook.) The service, the inbox section, and the outbox section all have the same name. In fact, clsOBXService is implemented as a subclass (hence a subset) of clsIOBXService.

**Class Messages**

**msgNewDefaults**

Initializes the P OBXSVC.NEW structure to default values.

Takes P OBXSVC.NEW, returns STATUS. Category: class message.

```
typedef struct OBXSVC.NEWONLY {
    OBJECT sectionClass; // class of the outbox section (for output queue)
    // This must be clsNBToc or a subclass of it.
    U32 unused1;
    U32 unused2;
    U32 unused3;
} OBXSVC.NEWONLY, *P_OBXSVC.NEWONLY;
#define obxServiceNewFields
    ioSvcNewFields \nbX newFields
    OBXSVC.NEWONLY obxsvc;
typedef struct OBXSVC.NEW {
    obxServiceNewFields
} OBXSVC.NEW, *P_OBXSVC.NEW;
Zeroes out pArgs->obxsvc and sets...>iobxsvc.out.autoPoll = true;>obxsvc.sectionClass = clsNBToc;
```

**msgNew**

Creates a new outbox service object.

Takes P OBXSVC.NEW, returns STATUS. Category: class message.

```
typedef struct OBXSVC.NEW {
    obxServiceNewFields
} OBXSVC.NEW, *P_OBXSVC.NEW;
```
**msgOBXSvcSwitchIcon**

Toggles the outbox icon (to empty or filled) if necesssary.

Takes nothing, returns STATUS. Category: class message.

```c
#define msgOBXSvcSwitchIcon msgIOBXSvcSwitchIcon
```

**Comments**

Check the content of the outbox notebook. Show the "filled" icon if any document is found. Show the "empty" icon otherwise.

**msgOBXDocGetService**

Gets the service name.

Takes P_OBX_DOC_GET_SERVICE, returns STATUS. Category: class message.

```c
#define msgOBXDocGetService msgIOBXDocGetService
```

**Arguments**

typedef struct OBX_DOC_GET_SERVICE {
  OBJECT document; // In: document uid
  CHAR  svcName[nameBufLength]; // Out: service name
} OBX_DOC_GET_SERVICE, *P_OBX_DOC_GET_SERVICE;

**Comments**

Get the name of the service associated with an outbox document. If the document has not been placed into an outbox section, stsFailed is returned.

Note that the document must be at the top level of an outbox section. That is, if the document is embedded within another document which is in an outbox section, stsFailed will be returned because the document is not at the top level of an outbox section.

**msgOBXDocInOutbox**

Checks if a document is in a section in the Outbox.

Takes P_OBX_DOC_IN_OUTBOX, returns STATUS. Category: class message.

```c
#define msgOBXDocInOutbox msgIOBXDocInIOBox
```

**Arguments**

typedef struct OBX_DOC_IN_OUTBOX {
  UUID    uuid;  // In: document uid
  CLASS   svcClass;  // In: service class to check for
} OBX_DOC_IN_OUTBOX, *P_OBX_DOC_IN_OUTBOX;

**Comments**

This message can be sent to clsOBXService to check if a PenPoint document represented by pArgs->uuid is already in the output queue of an outbox service inheriting from pArgs->svcClass. stsOK is returned if it is, stsFailed otherwise. If pArgs->svcClass is objNull, stsOK is returned if the document is anywhere in the Outbox notebook.

**Messages Sent to an Outbox Service Instance**

**msgOBXSvcMoveInDoc**

Moves a document into the outbox section.

Takes P_OBXSVC_MOVE_COPY_DOC, returns STATUS.

```c
#define msgOBXSvcMoveInDoc msgIOBXSvcMoveInDoc
```
typedef struct OBXSVC_MOVE_COPY_DOC {
    FS_LOCATOR source; // In: Location of source document.
    U16 sequence; // In: Sequence number to move/copy in
                  // front of.
} OBXSVC_MOVE_COPY_DOC, *P_OBXSVC_MOVE_COPY_DOC;

Superclass behavior is to move the document located at pArgs->source into the output queue associated with the outbox service. For example, set pArgs->sequence to 1 to move the document to the top of the queue. Set it to maxU16 to move the document to the bottom of the queue.

After the document is moved (or copied) to the output queue, it is considered to be in a state ready for output, even though the service may not be connected at the time. Client should not alter the document in any way once it has been moved to the output queue.

Subclasses can provide their own behavior if they wish. Remember to use the class message msgOBXSvcSwitchIcon to change the outbox icon.

msgOBXSvcCopyInDoc
Copies a document into the Outbox section.
Takes P_OBXSVC_MOVE_COPY_DOC, returns STATUS.

#define msgOBXSvcCopyInDoc msgIOBXSvcCopyInDoc

typedef struct OBXSVC_MOVE_COPY_DOC {
    FS_LOCATOR source; // In: Location of source document.
    U16 sequence; // In: Sequence number to move/copy in
                  // front of.
} OBXSVC_MOVE_COPY_DOC, *P_OBXSVC_MOVE_COPY_DOC;

Same as msgOBXSvcMoveInDoc, except that the document is copied to the output queue.

msgOBXSvcGetTempDir
Passes back a handle for a temporary directory.
Takes P_OBJECT, returns STATUS.

#define msgOBXSvcGetTempDir msgIOBXSvcGetTempDir

This message is provided for clients who may want to prepare their output document before moving it into the output queue. The handle of an "official" temporary directory is passed back and it can be used as temporary storage for documents, data, etc. Clients are responsible for deleting temporary files when they are done. The directory will be flushed after a warm boot.

msgOBXSvcPollDocuments
Polls all documents in an output queue and output those who are ready.
Takes nothing, returns STATUS.

#define msgOBXSvcPollDocuments msgIOBXSvcPollDocuments

This message tells the outbox service to look through its output queue and send out the first document ready for output. The service will first make sure that it is enabled and is connected to the designated output port. If these conditions are met, it will then self-send msgOBXSvcNextDocument to locate the next document ready for output.

If msgOBXSvcNextDocument returns stsOK, indicating that a document is ready for output, this message proceeds to self-send msgOBXSvcLockDocument to lock the document, and finally msgOBXSvcOutputStart to initiate the output process.
If msgOBXSvcNextDocument returns stsOBXSvcDocReady, indicating that the section is not empty but none of the documents are ready for output, this message self-sends msgOBXSvcScheduleDocument to schedule the document passed back in pArgs at a later time.

Subclasses normally do not process this message.

See Also

msgOBXSvcNextDocument

msgOBXSvcNextDocument

Passes back the next document ready for output.

Takes P_OBX SVC _DOCUMENT, returns STATUS. Category: self-sent.

#define msgOBXSvcNextDocument msgIOBXSvcNextDocument

typedef struct OBXSVC_DOCUMENT {
  OBJECT uid; // uid of the doc
  OBJECT dir; // app dir of the doc
  OBJECT docClass; // class of the doc
  U16 sequence; // sequence of the doc
  CHAR pName[nameBufLength]; // name of this doc
  P_UNKNOWN pDocData; // subclass's private data
} OBXSVC_DOCUMENT, *P_OBXSVC DOCUMENT;

Comments

Superclass behavior is to start from the top of the output queue and locate the first document ready for output. If one is found, information about the document is passed back in pArgs. The same pArgs will be passed to messages msgOBXSvcLockDocument and msgOBXSvcOutputStart. By default, a document is ready for output when it is closed. If the document is open, it will receive msgOBXDocOutputStartOK and it should return stsOK to indicate that it is ready for output.

Subclasses can provide their own behavior if they wish. Return stsOBXSvcSectionEmpty to give the superclass an opportunity to change the outbox icon from filled to empty.

Return Value

stsOK A document is ready for output.

stsOBXSvcSectionEmpty The output queue is empty.

stsOBXSvcDocNotReady No document in the output queue is ready.

Service-Specific Error Returns.

See Also

msgOBXSvcPollDocuments

msgOBXSvcLockDocument

Locks the document in preparation for output.

Takes P_OBX SVC _DOCUMENT, returns STATUS. Category: self-sent.

#define msgOBXSvcLockDocument msgIOBXSvcLockDocument

typedef struct OBXSVC_DOCUMENT {
  OBJECT uid; // uid of the doc
  OBJECT dir; // app dir of the doc
  OBJECT docClass; // class of the doc
  U16 sequence; // sequence of the doc
  CHAR pName[nameBufLength]; // name of this doc
  P_UNKNOWN pDocData; // subclass's private data
} OBXSVC_DOCUMENT, *P_OBXSVC DOCUMENT;

Comments

This message is a place holder for subclasses that may require additional preparatory work to be performed on a document before it is ready for output. For example, a document may have to be "locked" so that it can not be opened during the output process. This message may be used for other
purposes as well. For example, an outbox service may decide to store a light-weight "shadow" document (e.g., a report designator for a database application) in the output queue until it is chosen for output. The service then handles this message by converting the shadow document to a real one (e.g., the actual report).

The superclass behavior for this message is to stamp the document directory with the filesystem attribute iobservDocOutputInProgress. This stamp will prevent any gestures over the document from being processed. This means that once a document is locked for output it can not be deleted, renamed, etc. via gestures.

See Also msgOBXSvcUnlockDocument

msgOBXSvcUnlockDocument

Unlocks a document that was previously locked.

Takes P_OBXSVC_DOCUMENT, returns STATUS. Category: self-sent.


typedef struct OBXSVC_DOCUMENT {
  OBJECT uid;  // uid of the doc
  OBJECT dir;  // app dir of the doc
  OBJECT docClass;  // class of the doc
  U16 sequence;  // sequence of the doc
  CHAR pName[nameBufLength];  // name of this doc
  P_UNKNOWN pDocData;  // subclass's private data
} OBXSVC_DOCUMENT, *P_OBXSVC_DOCUMENT;

Comments

This message is a place holder for subclasses that may require additional "cleanup" work to be performed on a document before it is put back to the output queue.

The superclass behavior for this message is to remove the iobservDocOutputInProgress stamp on the document directory.

See Also msgOBXSvcLockDocument

msgOBXSvcScheduleDocument

Schedules a document that is not ready for output

Takes P_OBXSVC_DOCUMENT, returns STATUS. Category: self-sent.


typedef struct OBXSVC_DOCUMENT {
  OBJECT uid;  // uid of the doc
  OBJECT dir;  // app dir of the doc
  OBJECT docClass;  // class of the doc
  U16 sequence;  // sequence of the doc
  CHAR pName[nameBufLength];  // name of this doc
  P_UNKNOWN pDocData;  // subclass's private data
} OBXSVC_DOCUMENT, *P_OBXSVC_DOCUMENT;

Comments

This message is sent when msgOBXSvcNextDocument locates a document in the output queue but the document is not ready for output.

Subclasses should provide their own behavior. The default behavior is to release the ownership of the target service (i.e., become disabled), with the expectation that the user must manually schedule the document later on (by re-enabling the section.)

See Also msgOBXSvcNextDocument
msgOBXSvcOutputStart

Starts the output process for a document in the output queue.

Takes P_OBXSVC_DOCUMENT, returns STATUS. Category: self-sent.

```c
#define msgOBXSvcOutputStart

typedef struct OBXSVC_DOCUMENT {
    OBJECT uid;       // uid of the doc
    OBJECT dir;       // app dir of the doc
    OBJECT docClass;  // class of the doc
    U16 sequence;     // sequence of the doc
    CHAR pName[nameBufLength]; // name of this doc
    P_UNKNOWN pDocData; // subclass's private data
} OBXSVC_DOCUMENT, *P_OBXSVC_DOCUMENT;
```

Comments

Superclass behavior is to activate the outbox document if it isn’t already active, and then send msgOBXSvcOutputStart to the document instance.

Subclasses can provide their own behavior if they wish.

msgOBXSvcOutputCancel

Cancels the output process.

Takes nothing, returns STATUS.

```c
#define msgOBXSvcOutputCancel
```

Comments

This message is sent to the service when the caller wishes to cancel any output operation in progress. The service responds to this message by sending msgOBXDocOutputCancel to an active outbox document. After the document is cancelled, the service will post an error note to the user if there are other documents waiting to be processed. The user then decides whether or not the service should proceed to send the remaining documents.

Subclasses do not normally process this message.

msgOBXSvcOutputCleanUp

Cleans up after the current output is done.

Takes P_OBX_DOC_OUTPUT_DONE, returns STATUS. Category: self-post..

```c
#define msgOBXSvcOutputCleanUp

Enum32 OBX_DOC_EXIT_BEHAVIOR) {
    obxDocExitDoNothing = 0,
    obxDocExitDelete = 1,
    obxDocExitMarkAsFailed = 2,
    obxDocExitMarkAsCancelled = 3
};
```

```c
typedef struct OBX_DOC_OUTPUT_DONE {
    OBX_DOC_EXIT_BEHAVIOR behavior; // exit behavior
    P_UNKNOWN pDocData; // Unused: document specific data
} OBX_DOC_OUTPUT_DONE, *P_OBX_DOC_OUTPUT_DONE;
```

Comments

This message is posted to self as a result of the service receiving msgOBXDocOutputDone, which is sent by the outbox document when it finishes the output operation. The outbox document will be either deleted or marked as specified in pArgs, and when everything is properly cleaned up the service will post msgOBXSvcPollDocuments to self to see if anything else is waiting for output.

Subclasses do not normally process this message.

See Also

msgOBXDocOutputDone
**msgOBXSvcStateChanged**

Tells observers that the service state just changed.

Takes OBJECT, returns STATUS. Category: observer notification...

```
#define msgOBXSvcStateChanged msgIOBXSvcStateChanged
```

**Comments**

Informs observers that the state of a service has just changed. `pArgs` is the UID of the service.

---

**msgOBXSvcQueryState**

Passes back the state of the service.

Takes `P_OBXSVC_QUERY_STATE`, returns STATUS.

```
#define msgOBXSvcQueryState msgIOBXSvcQueryState
```

**Arguments**

```
typedef struct {
    BOOLEAN enabled; // true if the service is enabled.
    CHAR status[nameBufLength]; // text describing the status of
    // the service.
    CHAR docName[nameBufLength]; // document being processed
    P_UNKNOWN pStateData; // subclass’s private data
} OBXSVC_QUERY_STATE, *P_OBXSVC_QUERY_STATE;
```

**Comments**

This message is typically used to query what state the service instance is in.

---

**msgOBXSvcGetEnabled**

Gets the enabled state of the service.

Takes `P_BOOLEAN`, returns STATUS.

```
#define msgOBXSvcGetEnabled msgIOBXSvcGetEnabled
```

**Comments**

Subclasses can override this message and redefine the notion of "enabled." The default behavior of the superclass is to equate "enabled" with the ownership of the target service (i.e., output device). That is, the service is "enabled" when it owns the target service. By appending to or replacing the default behavior, a subclass can define additional conditions which must be met before a service is considered enabled.

---

**msgOBXSvcSetEnabled**

Sets the enabled state of the service.

Takes BOOLEAN, returns STATUS.

```
#define msgOBXSvcSetEnabled msgIOBXSvcSetEnabled
```

**Comments**

This message is sent to the service in response to service notification messages `msgSvcOwnerAcquired` and `msgSvcOwnerReleased`. Subclasses can provide their own behavior and thereby redefine the notion of "enabled" for the service. If they do, they must pass this message up to the ancestor so that observers of the outbox service will be properly notified.
Outbox Document Messages

**msgOBXDocOutputStartOK**

Asks the outbox document if it is OK to start the output process.

Takes nothing, returns STATUS.

```c
#define msgOBXDocOutputStartOK             msgIOBXDocIOStartOK
```

**Comments**

When an outbox service finds an opened document in the outbox section, it sends this message to the document instance, asking whether it's OK to start the output operation while the document remains open. When the document receives this message, it should return **stsOK** to give the service permission to begin the output process. An error status, including **stsNotUnderstood**, is taken to mean that the document instance vetos the request and the service will not start the output process.

**msgOBXDocOutputStart**

Tells an outbox document to start the output process.

Takes nothing, returns STATUS.

```c
#define msgOBXDocOutputStart             msgIOBXDocIOStart
```

**Comments**

This message is sent by the outbox service to a document. The document should respond to this message by starting the output process.

**msgOBXDocOutputDone**

Tells the outbox service that output is finished.

Takes **P_OBX_DOC_OUTPUT_DONE**, returns STATUS. Category: client responsibility.

```c
#define msgOBXDocOutputDone           msgIOBXDocIODone
```

**Message**

typedef struct OBX_DOC_OUTPUT_DONE {
    OBX_DOC_EXIT_BEHAVIOR behavior; // exit behavior
    PUNKNOWN pDocData; // Unused: document specific data
} OBX_DOC_OUTPUT_DONE, *P_OBX_DOC_OUTPUT_DONE;

**Comments**

When the output process is finished, the outbox document in charge of the output should send this message to the outbox service. This message must be sent even if the output process has been aborted. The **pArgs** for this message tells the outbox service what to do with the outbox document. If **obxDocExitDelete** is specified, the document will be removed from the outbox. In all other cases the document will be unlocked and left in the outbox. If either **obxDocExitMarkAsCancelled** or **obxDocExitMarkAsFailed** are specified, the name of the document will be altered to provide visual indication for the user that the output process has not completed successfully.

**See Also**

msgOBXDocGetService

**msgOBXDocOutputCancel**

Tells an outbox document to cancel the output process.

Takes nothing, returns STATUS.

```c
#define msgOBXDocOutputCancel          msgIOBXDocIOCancel
```
This message is used by the outbox service to inform a document that it should cancel the output process. The document should handle this message by terminating its output operation and then sending `msgOBXDocOutputDone` to the service with `pArgs->behavior` set to `obxDocExistMarkAsCancelled`.

**msgOBXDocStatusChanged**

Tells the outbox service that the document status is changed.

Takes `P_OBX_DOC_STATUS_CHANGED`, returns `STATUS`. Category: client responsibility.

```c
#define msgOBXDocStatusChanged msgIOBXDocStatusChanged
```

```c
typedef struct OBX_DOC_STATUS_CHANGED {
    CHAR status[nameBufLength]; // Text describing document state
    P.Unknown pDocData; // Unused: document-specific data
} OBX_DOC_STATUS_CHANGED, *P_OBX_DOC_STATUS_CHANGED;
```

This message is sent by the outbox document to the service whenever its status has just changed. This status is displayed on Status column for the outbox section, in the Outbox notebook.
This file contains the API definition for clsOpenServiceObject.

clsOpenServiceObject inherits from clsStream.

Provides default behavior for open service objects.

All open service object classes must be a subclass of clsOpenServiceObject. This superclass forwards all clsService messages to the actual service instance. It also allows a subclass to easily get the service instance that it is associated with.

```c
#ifndef OPENSERV_INCLUDED
#define OPENSERV_INCLUDED
#endif

#ifndef STREAM_INCLUDED
#include <stream.h>
#endif

Messages

msgNew

Creates a new service object.

Takes P_OBJECT, returns STATUS. Category: class message.

```c
typedef struct OSO_NEW_ONLY {
    OBJECT serviceInstance;  // This is filled in by clsService at open time.
    U32 unused1;
    U32 unused2;
    U32 unused3;
    U32 unused4;
} OSO_NEW_ONLY, *P_OSO_NEW_ONLY, OSO_METRICS, *P_OSO_METRICS;
#define openServiceObjectNewFields \
    streamNewFields \
    OSO_NEW_ONLY openServiceObject;
```n
msgOSOGetServiceInstance

Returns the service instance that this object is associated with.

Takes P_OBJECT, returns STATUS.

```c
#define msgOSOGetServiceInstance MakeMsg(clsOpenServiceObject, 1)
```
This file contains the API definition for clsParallelPort.

clsParallelPort inherits from clsMILService.

This mil service provides the interface between the parallel printer mil device and the rest of Penpoint. This interface allows for the configuring of the parallel printer mil device and for printing using the parallel printer mil device. The pport mil service will typically only be accessed by printer drivers since they are responsible for rendering an image for printing.

You access this mil service by using the standard service access techniques. These techniques are described in servmgr.h.

The pport mil service is a member of the 'theParallelDevices' and 'thePrinterDevices' service managers.

```c
#ifndef PPORT_INCLUDED
#define PPORT_INCLUDED
#ifndef GO_INCLUDED
# include <go.h>
#endif
#ifndef CLSMGR_INCLUDED
# include <clsmgr.h>
#endif
#ifndef MIL_SERVICE_INCLUDED
# include <milserv.h>
#endif

Common #defines and typedefs

typedef OBJECT PPORT, *P_PPORT;
#define stsPPortBusy MakeStatus(clsParallelPort, 1)
#define stsPPortOutOfPaper MakeStatus(clsParallelPort, 2)
#define stsPPortOffline MakeStatus(clsParallelPort, 3)
#define stsPPortNoPrinter MakeStatus(clsParallelPort, 4)
#define stsPPortPrinterErr MakeStatus(clsParallelPort, 5)

typedef struct PPORT_METRICS
{
    U16 version;       // version number of pport
    U16 devFlags;      // device flags (none defined)
    U16 unitFlags;     // unit flags (see dvparall.h)
    U32 initDelay;     // time in microSeconds init signal
                         // is applied to printer
    U32 interruptTimeOut; // the printer should be ready to accept
                           // another character within this time
                           // period (in milliseconds)
} PPORT_METRICS, *P_PPORT_METRICS;
```
**Parallel Port Class Messages**

**msgPPortStatus**
returns the current hardware status of the printer.

Takes P_PPORT_STATUS, returns STATUS.

```c
#define msgPPortStatus MakeMsg(clsParallelPort, 3)
#define pportStsBusy flag7 // printer is busy
#define pportStsAcknowledge flag6 // printer acknowledged char.
#define pportStsEndOfPaper flag5 // printer out of paper
#define pportStsSelected flag4 // printer on line
#define pportStsIOError flag3 // printer error occurred
#define pportStsInterruptHappened flag2 // printer interrupt occurred
```

typedef struct PPORT_STATUS
{
    U16 pportStatus;
} PPORT_STATUS, *P_PPORT_STATUS;

'pportStatus' is the contents of the parallel port status register.

**msgPPortInitialize**
initializes the printer.

Takes P_NULL, returns STATUS.

```c
#define msgPPortInitialize MakeMsg(clsParallelPort, 4)
```

The printer is initialized by asserting the control "Initialize" to the printer for initDelay microseconds.

**msgPPortAutoLineFeedOn**
inserts a line feed after each carriage return.

Takes P_NULL, returns STATUS.

```c
#define msgPPortAutoLineFeedOn MakeMsg(clsParallelPort, 5)
```

The auto line feed signal to the printer is set active.

**msgPPortAutoLineFeedOff**
disables inserting a line feed after each carriage return.

Takes P_NULL, returns STATUS.

```c
#define msgPPortAutoLineFeedOff MakeMsg(clsParallelPort, 6)
```

The auto line feed signal to the printer is set inactive.

**msgPPortGetTimeDelays**
gets the initialization and interrupt time out intervals.

Takes P_PPORT_TIME_DELAYS, returns STATUS.

```c
#define msgPPortGetTimeDelays MakeMsg(clsParallelPort, 7)
```

```c
typedef struct PPORT_TIME_DELAYS
{
    U32 initDelay;       // initialization delay
    U32 interruptTimeOut; // interrupt time out
} PPORT_TIME_DELAYS, *P_PPORT_TIME_DELAYS;
```
The initialization time period is the time the initialization pulse asserted to the printer in microseconds. The interrupt time out is the maximum time the printer will assert busy before being to accept another character in milliseconds.

**msgPPortSetTimeDelays**

sets the initialization and interrupt time out intervals.

Takes P_PPOR(TETIME_DELAYS, returns STATUS.

```c
#define msgPPortSetTimeDelays MakeMsg(clsParallelPort, 8)
```

```c
typedef struct PPORT_TIME_DELAYS {
    U32 initDelay; // initialization delay
    U32 interruptTimeOut; // interrupt time out
} PPORT_TIME_DELAYS, *P_PPORT_TIME_DELAYS;
```

Neither value can be zero. It's best to get the present before changing the time intervals.

**msgPPortCancelPrint**

cancels the printing of the buffer currently being printed.

Takes P_NULL, returns STATUS.

```c
#define msgPPortCancelPrint MakeMsg(clsParallelPort, 9)
```

**msgNew**

creates a new pport object.

Takes P_PPORT_NEW, returns STATUS.

```c
#define pportNewFields milServiceNewFields
```

```c
typedef struct PPORT_NEW {
    pportNewFields
} PPORT_NEW, *P_PPORT_NEW;
```

STATUS EXPORTED CIsParallelPortInit(void);
SENDSERV.H

This file contains the class definition and methods for clsSendableService.

clsSendableService inherits from clsService.

Provides the API for the services which appear on the Document Send menu.

clsSendableService is an abstract superclass which defines the sendable services protocol. This protocol is used by the Send Manager and the address book to interact with services on the SendableServices service manager. All services on this list *must* implement this protocol.

```c
#ifndef SENDSERV_INCLUDED
#define SENDSERV_INCLUDED
#ifndef ADDRBOOK_INCLUDED
#include <addrbook.h>
#endif
#endif
```

### Common #defines and typedefs

Data window fields are empty.

```c
#define stsSendServAddrWinEmpty MakeWarning(clsSendableService, 1)
```

### Messages

**msgSendServCreateAddrWin**

Converts address data into a window displaying the data.

Takes P_SEND_SERV_ADDR_WIN, returns STATUS.

```c
#define msgSendServCreateAddrWin MakeMsg(clsSendableService, 1)
```

**Arguments**

```c
typedef struct SEND_SERV_ADDR_WIN {
    U16 numAttrs;
    P_ADDR_BOOK_ATTR attrs;
    P_STRING addrSummary;
    BOOLEAN errNote;
    OBJECT win;
} SEND_SERV_ADDR_WIN, *P_SEND_SERV_ADDR_WIN;
```

**Comments**

This message is sent to a sendable service by the address book. A sendable service should create a display window(pArgs->win). The sendable service should wait for msgSendServFillAddrWin before it fills in the fields in the window.

**Parameters:**

- `pArgs->numAttrs` In: number of attributes in the attrs array.
- `pArgs->attrs` In: an array of size `pArgs->numAttrs`. `pArgs->attrs[x].value` contains what the sendable service needs to display.
- `pArgs->win` Out: sendable-service-created display window.
**msgSendServGetAddrSummary**

Given pArgs->attrs, set pArgs->addrSummary to be a displayable string that sums up the address.

Takes P_SEND_SERV_ADDR_WIN, returns STATUS.

```c
#define msgSendServGetAddrSummary MakeMsg(clsSendableService, 9)
```

**Message Arguments**

```c
typedef struct SEND_SERV_ADDR_WIN {
    U16 numAttrs;
    P_ADDR_BOOK_ATTR attrs;
    P_STRING addrSummary;
    BOOLEAN errNote;
    OBJECT win;
} SEND_SERV_ADDR_WIN, *P_SEND_SERV_ADDR_WIN;
```

**Comments**

- **pArgs->numAttrs** In: number of attributes in the attrs array.
- **pArgs->attrs** In: an array of size pArgs->numAttrs.
- **pArgs->addrSummary** Out: a string that sums up the address information described in attribute-value form in pArgs->attrs.

**msgSendServFillAddrWin**

Sendable service refreshes pArgs->win with information in pArgs->attrs.

Takes P_SEND_SERV_ADDR_WIN, returns STATUS.

```c
#define msgSendServFillAddrWin MakeMsg(clsSendableService, 8)
```

**Message Arguments**

```c
typedef struct SEND_SERV_ADDR_WIN {
    U16 numAttrs;
    P_ADDR_BOOK_ATTR attrs;
    P_STRING addrSummary;
    BOOLEAN errNote;
    OBJECT win;
} SEND_SERV_ADDR_WIN, *P_SEND_SERV_ADDR_WIN;
```

**Comments**

An address book sends a sendable service this message to refresh the window that contains information described in pArgs->attrs.

- **pArgs->numAttrs** In: number of attributes in the attrs array. If 0, then clear all fields.
- **pArgs->attrs** In: an array of size pArgs->numAttrs. pArgs->attrs[x].value contains what the sendable service needs to display.
- **pArgs->win** In: uid of sendable-service-created display window.

**msgSendServEncodeAddrWin**

Converts a window which displays address data into data.

Takes P_SEND_SERV_ADDR_WIN, returns STATUS.

```c
#define msgSendServEncodeAddrWin MakeMsg(clsSendableService, 2)
```

**Message Arguments**

```c
typedef struct SEND_SERV_ADDR_WIN {
    U16 numAttrs;
    P_ADDR_BOOK_ATTR attrs;
    P_STRING addrSummary;
    BOOLEAN errNote;
    OBJECT win;
} SEND_SERV_ADDR_WIN, *P_SEND_SERV_ADDR_WIN;
```
The service must convert the window into an array of attribute-values, as described in ADDR_BOOK_SERVICE_DESC. Storage for this array should be created by the sendable service from a global heap. The caller client is responsible for freeing this storage.

Parameters:

- `pArgs->numAttrs` Out: Number of elements in the `.attrs` array
- `pArgs->attrs` Out: fill in the values of each attribute.
- `pArgs->errNote` In: if TRUE, then the service should display some kind of note on the screen when error occurs during data collection and validation.
- `pArgs->win` In: the window to get the data from. Presumably the sendable service created this window in response to a previous msgSendServCreateAddrWin.

Return Value

- `stsServiceDataWinEmpty` All data element fields are empty.
- `stsFailed` Some error occurs during data collection and validation.

### msgSendServEncodeAddrData

Converts service-specific data into ASCII byte array.

Takes `P_SEND_SERV_CONVERT_ADDR_DATA`, returns STATUS.

```c
#define msgSendServEncodeAddrData MakeMsg(clsSendableService, 3)
```

**Arguments**

```c
typedef struct SEND_SERV_CONVERT_ADDR_DATA {
    P_U8 pBuf;        // In/Out: Encoded addressing data
    U16 bufLen;       // In/Out: Length of pBuf
    U16 numAttrs;
    P_ADDR_BOOK_ATTR attrs;
} SEND_SERV_CONVERT_ADDR_DATA;
```

**Comments**

*** This message is obsolete ***

The resulting data is put into `.attrs` and update the attribute count in `.numAttrs`.

### msgSendServDecodeAddrData

Converts ASCII data into service-specific data.

Takes `P_SEND_SERV_CONVERT_ADDR_DATA`, returns STATUS.

```c
#define msgSendServDecodeAddrData MakeMsg(clsSendableService, 4)
```

**Arguments**

```c
typedef struct SEND_SERV_CONVERT_ADDR_DATA {
    P_U8 pBuf;        // In/Out: Encoded addressing data
    U16 bufLen;       // In/Out: Length of pBuf
    U16 numAttrs;
    P_ADDR_BOOK_ATTR attrs;
} SEND_SERV_CONVERT_ADDR_DATA;
```

**Comments**

*** This message is obsolete ***

The resulting data is put into `.attrs` and update the attribute count in `.numAttrs`.
msgAppExecute
Displays a UI for obtaining addressing info and executing the send.
Takes P_APP_EXECUTE, returns STATUS.
Comments
This message is a standard clsApp message which is forwarded to the service the user has selected from
the standard "Send" menu. The service should create and display their UI for obtaining addressing
information from the user.
Declaration for the APP_EXECUTE data structure can be found in app.h

msgSendServGetAddrDesc
Responsibility of a sendable service to return its service attribute-value pairs that describe its service
address
Takes P_ADDR_BOOK_SVC_DESC, returns STATUS.
#define msgSendServGetAddrDesc MakeMsg(clsSendableService, 7)
Comments
An address book usually send this message to a sendable service as part of of initialization to find out the
service address description.
This file contains the definition and methods for clsALAPSerial

```c
#ifndef SERLINK_INCLUDED
#define SERLINK_INCLUDED

ALAP_SERIAL_NEW_ONLY, *P_ALAP_SERIAL_NEW_ONLY;
#define alapSerialNewFields
    serviceNewFields
    ALAP_SERIAL_NEW_ONLY    alapSerial;
ALAP_SERIAL_NEW, *P_ALAP_SERIAL_NEW;
STATUS EXPORTED ClsSerLinkInit(void);
```

SIO.H

This file contains the API for clsMILAsyncSIODevice.

clsMILAsyncSIODevice inherits from clsStream.

Provides the serial port interface, see also stream.h for the stream messages.

```c
#ifndef SIO_INCLUDED
#define SIO_INCLUDED
#include <go.h>
#include <clsmgr.h>
#include <milserv.h>

Common #defines and typedefs
#define stsSioPortInUse
#define milDefaultBaudRate 9600
#define milDefaultXonChar 0x11
#define milDefaultXoffChar 0x13
#define milDefaultModemControl milDataTerminalReady | milRequestToSend
#define milDefaultStopBits milOneStopBit
#define milDefaultParityType milNoParity
#define milDefaultWordLength milEightBitWord
#define milDefaultXonTimeout (U32)30000
#define milDefaultLineToSend milRequestToSend

typedef OBJECT SIO;
typedef SIO * P_SIO;

Enum16(SIO_EVENT_MASK) {
    sioEventCTS = flag0, // CTS line has changed state
    sioEventDSR = flag1, // DSR line has changed state
    sioEventDCD = flag2, // DCD line has changed state
    sioEventRI = flag3, // RI line has changed state
    sioEventRxChar = flag4, // Rx buffer has become not empty.
                     // Note: The receive buffer must be empty for a received character
                     // to generate this event!
    sioEventRxBreak = flag5, // Break condition has been received
    sioEventTxBufferEmpty = flag6, // Tx buffer has become empty
    sioEventRxError = flag7, // parity, framing, or overrun error
    sioAllEvents = flag0 | flag1 | flag2 | flag3 | flag4
                     | flag5 | flag6 | flag7
};
```

Asynchronous SIO Class Messages

msgSioBaudSet
Sets the serial port baud rate.
Takes U32, returns STATUS.

```c
#define msgSioBaudSet MakeMsg(clsMILAsyncSIODevice, 4)
```

Comments
Maximum possible setting 115200. Actual baud rate = (115200/((U32)(115200/baudRate))) Default setting 9600 baud
msgSioLineControlSet

Sets serial port data bits per character, stop bits, and parity.

Takes P_SIO_LINE_CONTROL_SET, returns STATUS.

```c
#define msgSioLineControlSet MakeMsg(clsMILAsyncSIODevice,5)
```

**Arguments**

```c
Enum16 (SIO_DATA_BITS) {
  sioSixBits = 6,
  sioSevenBits = 7,
  sioEightBits = 8
};
Enum16 (SIO_STOP_BITS) {
  sioOneStopBit = 0,
  sioOneAndAHalfStopBits = 1,
  sioTwoStopBits = 2
};
Enum16 (SIO_PARITY) {
  sioNoParity = 0,
  sioOddParity = 1,
  sioEvenParity = 2
};
typedef struct {
  SIO_DATA_BITS dataBits;
  SIO_STOP_BITS stopBits;
  SIO_PARITY parity;
} SIO_LINE_CONTROL_SET, *P_SIO_LINE_CONTROL_SET;
```

**Comments**

Default setting 8 bits, 1 stop bit, no parity.

msgSioControlOutSet

Controls serial port output lines dtr and rts.

Takes P_SIO_CONTROL_OUT_SET, returns STATUS.

```c
#define msgSioControlOutSet MakeMsg(clsMILAsyncSIODevice,6)
```

**Arguments**

```c
typedef struct {
  BOOLEAN dtr;     // true activates, false deactivates
  BOOLEAN rts;     // true activates, false deactivates
  BOOLEAN out1;    // true activates, false deactivates
  BOOLEAN out2;    // true activates, false deactivates
} SIO_CONTROL_OUT_SET, *P_SIO_CONTROL_OUT_SET;
```

**Comments**

Default setting dtr active, rts active.

msgSioControlInStatus

Reads the current state of the serial port input control lines.

Takes P_SIO_CONTROL_IN_STATUS, returns STATUS.

```c
#define msgSioControlInStatus MakeMsg(clsMILAsyncSIODevice,7)
```

**Arguments**

```c
typedef struct {
  BOOLEAN cts;     // out = true = active (Clear To Send)
  BOOLEAN dsr;     // out = true = active (Data Set Ready)
  BOOLEAN dcd;     // out = true = active (Data Carrier Detect)
  BOOLEAN ri;      // out = true = active (Ring Indicator)
} SIO_CONTROL_IN_STATUS, *P_SIO_CONTROL_IN_STATUS;
```

**Comments**

Default setting rlsd = dcd
**msgSioFlowControlCharSet**

Defines serial port XON/XOFF flow control characters.

Takes P_SIO_FLOW_CONTROL_CHAR_SET, returns STATUS.

```c
#define msgSioFlowControlCharSet MakeMsg(clsMILAsyncSIODevice, 8)
```

**Arguments**

typedef struct {
    U8 xonChar;  // xon character (default control-Q)
    U8 xoffChar; // xoff character (default control-S)
} SIO_FLOW_CONTROL_CHAR_SET, *P_SIO_FLOW_CONTROL_CHAR_SET;

**Comments**

Valid only if xon-xoff flow control is enabled.

Default xon character 0x11 (control-q), default xoff character 0x13 (control-s).

**msgSioBreakSend**

Sends a break for the specified duration.

Takes P_SIO_BREAK_SEND, returns STATUS.

```c
#define msgSioBreakSend MakeMsg(clsMILAsyncSIODevice, 11)
```

**Arguments**

typedef struct {
    OS_MILLISECONDS milliseconds;  // break duration
} SIO_BREAK_SEND, *P_SIO_BREAK_SEND;

**Comments**

Constant 0's transmitted on the serial line for the specified duration. Typical durations are around 200-400 milliseconds.

**msgSioBreakStatus**

Sends back the number of breaks received so far.

Takes P_SIO_BREAK_STATUS, returns STATUS.

```c
#define msgSioBreakStatus MakeMsg(clsMILAsyncSIODevice, 13)
```

**Arguments**

typedef struct {
    U32 breaksReceived;  // out
} SIO_BREAK_STATUS, *P_SIO_BREAK_STATUS;

**Comments**

Also clears the internal break counter.

**msgSioReceiveErrorsStatus**

Sends back the number of receive errors and the number of dropped bytes (due to buffer overflows).

Takes P_SIO_RECEIVE_ERRORS_STATUS, returns STATUS.

```c
#define msgSioReceiveErrorsStatus MakeMsg(clsMILAsyncSIODevice, 36)
```

**Arguments**

typedef struct {
    U32 droppedBytes;  // out
    U32 receiveErrors; // out
} SIO_RECEIVE_ERRORS_STATUS, *P_SIO_RECEIVE_ERRORS_STATUS;

**Comments**

Also clears the internal counters.
msgSioInputBufferStatus
Provides input buffer status.
Takes P_SIO_INPUT_BUFFER_STATUS, returns STATUS.
#define msgSioInputBufferStatus MakeMsg(clsMILAsyncSIODevice,16)

Arguments
ttypedef struct {
    U32      bufferChars; // out, number of chars in buffer
    S32      bufferRoom;  // out, amount of empty room in buffer
    BOOLEAN  receiverFrozen; // out, is receive frozen?
} SIO_INPUT_BUFFER_STATUS, * P_SIO_INPUT_BUFFER_STATUS;

Comments
Sends back the number of characters in the input buffer and the amount of empty room in the input buffer.

msgSioOutputBufferStatus
Provides output buffer status.
Takes P_SIO_OUTPUT_BUFFER_STATUS, returns STATUS.
#define msgSioOutputBufferStatus MakeMsg(clsMILAsyncSIODevice,17)

Arguments
ttypedef struct {
    U32      bufferChars; // out, number of chars in buffer
    S32      bufferRoom;  // out, amount of empty room in buffer
    BOOLEAN  transmitterFrozen; // out, is transmit frozen?
} SIO_OUTPUT_BUFFER_STATUS, * P_SIO_OUTPUT_BUFFER_STATUS;

Comments
Sends back the number of characters in the output buffer and the amount of empty room in the output buffer.

msgSioInputBufferFlush
Flushes the contents of the input buffer.
Takes pNull, returns STATUS.
#define msgSioInputBufferFlush MakeMsg(clsMILAsyncSIODevice,18)

msgSioOutputBufferFlush
Flushes the contents of the output buffer.
Takes pNull, returns STATUS.
#define msgSioOutputBufferFlush MakeMsg(clsMILAsyncSIODevice,19)

msgSioFlowControlSet
Selects flow control type.
Takes P_SIO_FLOW_CONTROL_SET, returns STATUS.
#define msgSioFlowControlSet MakeMsg(clsMILAsyncSIODevice,20)
Enum16(SIO_FLOW_TYPE) {
    sioNoFlowControl        = 0x11,
    sioXonXoffFlowControl   = 0x22,
    sioHardwareFlowControl  = 0x44,
    // To independently set receive and transmit flow control OR together
    // one from each of the following two sets.
    // i.e., flowControl = sioRxXonXoff | sioTxHardware;
    // YOU MUST SET BOTH THE TX AND RX FLOW CONTROL!
    // Transmit flow control
    sioTxNone                = 0x01,
    sioTxXonXoff             = 0x02,
    sioTxHardware            = 0x04,
    // Receive flow control
    sioRxNone                = 0x10,
    sioRxXonXoff             = 0x20,
    sioRxHardware            = 0x40
};
typedef struct {
    SIO_FLOW_TYPE  flowControl;
} SIO_FLOW_CONTROL_SET, *P_SIO_FLOW_CONTROL_SET;

Flow control types: no flow control, XON/XOFF flow control, or hardware flow control. Default: XON/XOFF flow control.

---

### msgSioEventStatus

Sends back current state of event word, and then clears the event word.

Takes `P_SIO_EVENT_STATUS`, returns `STATUS`.

```c
#define msgSioEventStatus MakeMsg(clsMILAsyncSIODevice, 21)
```

---

### msgSioEventSet

Enables event notification.

Takes `P_SIO_EVENT_SET`, returns `STATUS`.

```c
#define msgSioEventSet MakeMsg(clsMILAsyncSIODevice, 22)
```

---

### msgSioEventGet

Gets the current sio event setting.

Takes `P_SIO_EVENT_SET`, returns `STATUS`.

```c
#define msgSioEventGet MakeMsg(clsMILAsyncSIODevice, 29)
```
typedef struct {
    SIO_EVENT_MASK eventMask; // in, events to respond to
    OBJECT client;            // object to inform when event happens
} SIO_EVENT_SET, *P_SIO_EVENT_SET;

msgSioEventHappened
Notifies client of event occurrence.
Takes P_SIO_EVENT_HAPPENED, returns STATUS.
#define msgSioEventHappened MakeMsg(clsMILAsyncSIODevice, 23)

typedef struct {
    SIO_EVENT_MASK eventMask; // out, bits set indicate event happened.
    OBJECT self;              // object which generated message.
} SIO_EVENT_HAPPENED, *P_SIO_EVENT_HAPPENED;

msgSioInit
Initializes the serial device to its default state.
Takes P_SIO_INIT, returns STATUS.
#define msgSioInit MakeMsg(clsMILAsyncSIODevice, 26)

typedef struct {
    U32 inputSize; // size of the input buffer
    U32 outputSize; // size of the output buffer
} SIO_INIT, *P_SIO_INIT;

msgSioGetMetrics
Sends back the sio metrics.
Takes P_SIO_METRICS, returns STATUS.
#define msgSioGetMetrics MakeMsg(clsMILAsyncSIODevice, 24)

typedef struct {
    SIO_LINE_CONTROL_SET baud; // out/in
    SIO_LINE_CONTROL_SET line; // out/in
    SIO_CONTROL_OUT_SET controlOut; // out/in
    SIO_FLOW_CONTROL_CHAR_SET flowChar; // out/in
    SIO_FLOW_CONTROL_SET flowType; // out/in
    // Changing the bufferSize fields causes reinitialization of serial
    // chip!
    SIO_INIT bufferSize; // out/in
    U8 spare[12];
} SIO_METRICS, *P_SIO_METRICS;

msgSioSetMetrics
Sets the sio metrics.
Takes P_SIO_METRICS, returns STATUS.
#define msgSioSetMetrics MakeMsg(clsMILAsyncSIODevice, 25)
typedef struct {
    U32 baud; // out/in
    SIO_LINE_CONTROL_SET line; // out/in
    SIO_CONTROL_OUT_SET controlOut; // out/in
    SIO_FLOW_CONTROL_CHAR_SET flowChar; // out/in
    SIO_FLOW_CONTROL_SET flowType; // out/in
} SIO_METRICS, *P_SIO_METRICS;

---

msgSioSetReplaceCharProc

Replaces the built in receive character interrupt routine.

Takes P_SIO_REPLACE_CHAR, returns STATUS.

#define msgSioSetReplaceCharProc MakeMsg(clsMILAsyncSIODevice, 72)

typedef struct SIO_REPLACE_CHAR
{
    P_SIO_CHAR_HANDLER pRxHandler; // address of character handler
    U32 handle; // user data (meaningless to clsMILAsyncSIO)
} SIO_REPLACE_CHAR, *P_SIO_REPLACE_CHAR;

Comments

This message calls the user defined function when a character is received. The procedure has the option to filter the character or to return and have the character processed normally. The user defined function returns a BOOLEAN indicating whether the function filtered the character or not.

msgNew

Creates a new clsMILAsyncSIODevice object.

Takes P_SIO_NEW, returns STATUS.

typedef struct SIO_NEW
{
    milServiceNewFields
} SIO_NEW, *P_SIO_NEW;

---

Asynchronous SIO Option Card Tags

#define sioTagOptionCard MakeTag(clsMILAsyncSIODevice, 19) // Card tag
#define sioTagName MakeTag(clsMILAsyncSIODevice, 20)
#define sioTagBaud MakeTag(clsMILAsyncSIODevice, 21)
#define sioTagFlowControl MakeTag(clsMILAsyncSIODevice, 22)
#define sioTagParity MakeTag(clsMILAsyncSIODevice, 23)
#define sioTagDataBits MakeTag(clsMILAsyncSIODevice, 24)
#define sioTagStopBits MakeTag(clsMILAsyncSIODevice, 25)
#define sioTagBaud300 MakeTag(clsMILAsyncSIODevice, 26)
#define sioTagBaud600 MakeTag(clsMILAsyncSIODevice, 27)
#define sioTagBaud1200 MakeTag(clsMILAsyncSIODevice, 28)
#define sioTagBaud2400 MakeTag(clsMILAsyncSIODevice, 29)
#define sioTagBaud4800 MakeTag(clsMILAsyncSIODevice, 30)
#define sioTagBaud9600 MakeTag(clsMILAsyncSIODevice, 31)
#define sioTagBaud19200 MakeTag(clsMILAsyncSIODevice, 32)
#define sioTagBaud38400 MakeTag(clsMILAsyncSIODevice, 33)
#define sioTagBaud57600 MakeTag(clsMILAsyncSIODevice, 34)
#define sioTagBaud115200 MakeTag(clsMILAsyncSIODevice, 35)

#define sioTagFlowNone MakeTag(clsMILAsyncSIODevice, 55)
#define sioTagFlowXonXoff MakeTag(clsMILAsyncSIODevice, 56)
#define sioTagFlowHardware MakeTag(clsMILAsyncSIODevice, 57)
#define sioTagParityNone MakeTag(clsMILAsyncSIODevice, 60)
#define sioTagParityOdd MakeTag(clsMILAsyncSIODevice, 61)
#define sioTagParityEven MakeTag(clsMILAsyncSIODevice, 62)
#define sioTagBits7 MakeTag(clsMILAsyncSIODevice, 65)
#define sioTagBits8 MakeTag(clsMILAsyncSIODevice, 66)
#define sioTagStopBitsOne MakeTag(clsMILAsyncSIODevice, 70)
#define sioTagStopBitsTwo MakeTag(clsMILAsyncSIODevice, 71)

Function prototypes

Function Prototype

STATUS EXPORTED ClsSioInit(void);
void EXPORTED SioSemaClear(P_UNKNOWN pHandle);
This file contains the class definition and methods for clsTransport. clsTransport inherits from clsOpenServiceObject.

Provides the API for replaceable transport layer network protocols.

```c
#ifndef TP_INCLUDED
#define TP_INCLUDED
#ifndef OPENSERV_INCLUDED
#include <openserv.h>
#endif
#endif
```

```c
typedef U8 TP_QUEUE_SIZE;
typedef U8 TP_ADDRESS, *P_TP_ADDRESS;
typedef U8 TP_OPTIONS, *P_TP_OPTIONS;
typedef U8 TP_BUFFER, *P_TP_BUFFER;
```

Service Types

```c
#define tpReliableService 1
#define tpDatagramService 2
#define tpTransactionService 3
```

### msgNew

Creates a transport (socket) handle object.

Takes P_TP_NEW, returns STATUS.

**Arguments**

```c
typedef struct TP_NEW_ONLY {
    TP_SERVICE service;
} TP_NEW_ONLY, *P_TP_NEW_ONLY;
```

```c
typedef struct TP_NEW {
    OSO_NEW oso;
    TP_NEW_ONLY tp;
} TP_NEW, *P_TP_NEW;
```

### msgDestroy

Destroys a transport handle object.

Takes OBJ_KEY, returns STATUS.

### msgTPAccept

Accepts a connection request from a remote process.

Takes P_TP_ACCEPT, returns STATUS.

```c
#define msgTPAccept MakeMsg(clsTransport, 1)
```

**Arguments**

```c
typedef struct TP_ACCEPT {
    OBJECT newHandle; // Out: uid of transport handle
    P_TP_ADDRESS pAddress; // ptr to protocol dependent address
} TP_ACCEPT, *P_TP_ACCEPT;
```
**msgTPBind**
Binds a transport handle to a transport address.
Takes P_TP_BIND, returns STATUS.

```c
#define msgTPBind MakeMsg( clsTransport, 2 )
```

**Arguments**

typedef struct TP_BIND {
    P_TP_ADDRESS pAddress;  // ptr to protocol dependent address
} TP_BIND, *P_TP_BIND;

---

**msgTPConnect**
Establishes a connection with a remote process.
Takes P_TP_CONNECT, returns STATUS.

```c
#define msgTPConnect MakeMsg( clsTransport, 3 )
```

**Arguments**

typedef struct TP_CONNECT {
    P_TP_ADDRESS pAddress;  // ptr to protocol dependent address
} TP_CONNECT, *P_TP_CONNECT;

---

**msgTPListen**
Allocates space for a queue of incoming connection requests.
Takes P_TP_LISTEN, returns STATUS.

```c
#define msgTPListen MakeMsg( clsTransport, 4 )
```

**Arguments**

typedef struct TP_LISTEN {
    TP_QUEUE_SIZE queueSize;  // max number of connection requests
} TP_LISTEN, *P_TP_LISTEN;

---

**msgTPRecv**
Receives a message.
Takes P_TP_RECV, returns STATUS.

```c
#define msgTPRecv MakeMsg( clsTransport, 5 )
```

**Arguments**

typedef struct TP_RECV {
    P_TP_BUFFER pBuffer;  // ptr to receive data buffer
    U16 length;  // size of receive buffer in bytes
    U16 count;  // number of bytes received
    P_TP_OPTIONS pOptions;  // ptr to protocol dependent options
} TP_RECV, *P_TP_RECV;

---

**msgTPRecvFrom**
Receives a datagram.
Takes P_TP_RECVFROM, returns STATUS.

```c
#define msgTPRecvFrom MakeMsg( clsTransport, 6 )
```

**Arguments**

typedef struct TP_RECVFROM {
    P_TP_BUFFER pBuffer;  // ptr to receive data buffer
    U16 length;  // size of receive buffer in bytes
    U16 count;  // number of bytes received
    P_TP_ADDRESS pAddress;  // ptr to protocol dependent address
    P_TP_OPTIONS pOptions;  // ptr to protocol dependent options
} TP_RECVFROM, *P_TP_RECVFROM;
msgTPSend
Sends a message.
Takes P_TP_SEND, returns STATUS.

#define msgTPSend MakeMsg( clsTransport, 7 )

typedef struct TP_SEND {
  P_TP_BUFFER pBuffer;  // ptr to send data buffer
  U16 count;            // number of bytes to send
  P_TP_OPTIONS pOptions;   // ptr to protocol dependent options
} TP_SEND, *P_TP_SEND;

msgTPSendTo
Sends a datagram.
Takes P_TP_SENDTO, returns STATUS.

#define msgTPSendTo MakeMsg( clsTransport, 8 )

typedef struct TP_SENDTO {
  P_TP_BUFFER pBuffer;  // ptr to send data buffer
  U16 count;            // number of bytes to send
  P_TP_OPTIONS pOptions;   // ptr to protocol dependent options
  P_TP_ADDRESS pAddress;   // ptr to protocol dependent address
} TP_SENDTO, *P_TP_SENDTO;

msgTPSendRecvTo
Sends a request and waits for a response. For transaction service only.
Takes P_TP_SENDRECVTO, returns STATUS.

#define msgTPSendRecvTo MakeMsg( clsTransport, 9 )

typedef struct TP_SENDRECVTO {
  P_TP_BUFFER pSendBuffer; // ptr to send data buffer
  U16 sendCount;           // number of bytes to send
  P_TP_BUFFER pRecvBuffer; // ptr to receive data buffer
  U16 recvLength;         // size of receive buffer in bytes
  U16 recvCount;          // number of bytes received
  P_TP_OPTIONS pOptions;  // ptr to protocol dependent options
  P_TP_ADDRESS pAddress;  // ptr to protocol dependent address
} TP_SENDRECVTO, *P_TP_SENDRECVTO;

Status Codes
#define stsTPnotSupported MakeStatus(clsTransport,1)
#define stsTPtooMany MakeStatus(clsTransport,2)
#define stsTPbadUser MakeStatus(clsTransport,3)
#define stsTPmaxUsers MakeStatus(clsTransport,4)
#define stsTPnoUser MakeStatus(clsTransport,5)
#define stsTPbadService MakeStatus(clsTransport,6)
#define stsTPnoSocket MakeStatus(clsTransport,7)
#define stsTPnoMemory MakeStatus(clsTransport,8)
#define stsTPLength MakeStatus(clsTransport,9)
#define stsTPnoTransaction MakeStatus(clsTransport,10)
#define stsTPddpLength MakeStatus(clsTransport,11)
#define stsTPnoBridge MakeStatus(clsTransport,12)
#define stsTPbadNetwork MakeStatus(clsTransport,13)
#define stsTPbadNode MakeStatus(clsTransport,14)
#define stsTPsocketInUse MakeStatus(clsTransport,15)
#define stsTPpending MakeStatus(clsTransport,16)
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stsTPddpQ</code></td>
<td><code>#define</code> MakeStatus(clsTransport,17)</td>
</tr>
<tr>
<td><code>stsTPoverflow</code></td>
<td><code>#define</code> MakeStatus(clsTransport,18)</td>
</tr>
<tr>
<td><code>stsTPbadParm</code></td>
<td><code>#define</code> MakeStatus(clsTransport,19)</td>
</tr>
<tr>
<td><code>stsTPfailed</code></td>
<td><code>#define</code> MakeStatus(clsTransport,20)</td>
</tr>
<tr>
<td><code>stsTPnameNotFound</code></td>
<td><code>#define</code> MakeStatus(clsTransport,21)</td>
</tr>
<tr>
<td><code>stsTPnameInUse</code></td>
<td><code>#define</code> MakeStatus(clsTransport,22)</td>
</tr>
<tr>
<td><code>stsTPnewSocket</code></td>
<td><code>#define</code> MakeStatus(clsTransport,23)</td>
</tr>
<tr>
<td><code>stsTPnoRoom</code></td>
<td><code>#define</code> MakeStatus(clsTransport,24)</td>
</tr>
<tr>
<td><code>stsTPnoLink</code></td>
<td><code>#define</code> MakeStatus(clsTransport,25)</td>
</tr>
</tbody>
</table>
Part 11 / Resources
Next up: 28
This file contains the API definition for clsPreferences.

clsPreferences inherits from clsObject.

clsPreferences provides a shell to access system preferences.

theSystemPreferences is a well-known instance of clsPreferences. theSystemPreferences provides access to read and write system wide preferences.

clsPreferences supports a set of preferences. Preferences are stored as resources in the "current" system preferences resource file. An instance of clsPreferences, known as theSystemPreferences, is created at boot time. This should be the only instance of clsPreferences in the system.

Preferences are named by well known resource id's (RES_ID's). This header file contains some predefined preference id's to simplify things. When defining new preferences, use the class that originated the preferences.

Clients can get and set preferences by accessing the well known object theSystemPreferences.

Preferences are stored in a resource file. Any request to read or write a preference will force a read or write to a file. This minimizes the amount of space required to store preferences. theSystemPreferences will respond to any resource file message defined in resfile.h and process them appropriately.

Remember, to read and write system preferences simply use the messages msgResReadData and msgResWriteData (or msgResUpdateData). theSystemPreferences forwards the msg to the current system preferences resource file.

As an example of reading a system preference:

```c
U16 lineHeight;
RES_READ_DATA read;
read.resId = prLineHeight;
read.heap = 0;
read.pData = &lineHeight;
read.length = SizeOf(U16);
ObjectCall(msgResReadData, theSystemPreferences, &read);
```

An example of writing a system preference:

```c
U16 lineHeight;
RES_WRITE_DATA write;
write.resId = prLineHeight;
write.pData = &lineHeight;
write.length = SizeOf(U16);
write.agent = resDefaultResAgent;
ObjectCall(msgResWriteData, theSystemPreferences, &write);
```

classPreferences "knows" about certain preferences (listed in this file below) and performs whatever interaction is required to activate the new preference. It also handles certain system wide notification and actions when certain preferences change. For example, clsPreferences will cause the system to be re-drawn and re-fonted when the system preference for the font changes.
dsPreferences will notify all observers when a preference has (potentially) changed. This will allow various objects to observe the SystemPreferences, and react to the preference changes.

Whenever a number of preferences are being changed, clients may wish to send msgPrefsWritingMany, followed by the preference writes, and then msgPrefsWritingDone. dsPreferences will use these messages to delay any layout that may occur as a result of writing preferences that cause layout.

clsPreferences will also send these messages to observers, allowing them to delay expensive operations until the preference changes are complete. As an example, when the preference set changes, msgPrefsWritingMany, followed by msgPrefsPreferenceChanged for each preference, followed by msgPrefsWritingDone is sent to the observers.

clsPreferences supports the concept of different sets of preferences. A set of preferences is stored in a single resource file in a well-known preferences directory managed by the InstalledPreferences.

clsPreferences supports messages to change the current preference set to another one that is already filed. In addition, clsPreferences allows a preference set to start "clean". When PenPoint first starts up (or during a warm boot), the SystemPreferences will contain the set of preferences associated with the "current" preference set managed by the InstalledPreferences. If no current set exists, the SystemPreferences will start with a "clean" preference set. When a preference set changes, clsPreferences will notify the observers of the changed preferences. This is because clsPreferences is notified via msgIMCurrentChanged from the install manager (see instlmgr.h).

To change the set of preference set programmatically, one must communicate with the InstallManager. An example code fragment to change a preference set. See instlmgr.h for details:

```c
IM_INSTALL install;
install.locator.uid = theBootVolume;
rn.fs.locator.pPath = n\PenPoint\prefs\PREFERENCESET";
install.exist = imExistReactivate;
install.listAttrLabel = 0;
install.listHandle = 0;
ObjectCall(msgIMInstall, theInstalledPrefs, &install);
ObjectCall(msgIMSetCurrent, theInstalledPrefs, install.handle);
```

---

**Known Preferences in the System**

The following are the predefined resource names, the data that reading and writing will return, and some predefined return values for certain preferences.

**System Font**

prSystemFont is the resource id for the system font. Reads and writes of this id use a P_PREF_SYSTEM_FONT. This resource will affect the returned value from PrefsSysFontInfo.
Known Preferences in the System

Changing this resource (via msgResWriteData) will cause the system to layout after notification of observers, which is expensive. This is done by doing an ObjectPost of msgPrefsLayoutSystem to self. As a result, clsPreferences will compare this resource to the previous value to prevent layout and observer notification if the write did not change the value.

#define tagPrSystemFont MakeWknResId(clsPreferences, 1)
#define prSystemFont tagPrSystemFont

Field Font

prUserFont is the resource id for the field (user) font. Reads and writes of this id use a P_PREF_SYSTEM_FONT. This preference will affect the returned data from PrefsSysFontInfo.

Changing this resource (via msgResWriteData) will cause the system to layout after notification of observers, which is expensive. This is done by doing an ObjectPost of msgPrefsLayoutSystem to self. As a result, clsPreferences will compare this resource to the previous value to prevent layout and observer notification if the write did not change the value.

#define tagPrUserFont MakeWknResId(clsPreferences, 2)
#define prUserFont tagPrUserFont

This data structure is the what is read in and written when reading and writing when the resId is prSystemFont or prUserFont. It contains a font specification, and a font scale to use.

typedef struct PREF_SYSTEM_FONT {
    SYSDC_FONT_SPEC spec; // Font spec
    SCALE scale; // Scale: same for system and user font
} PREF_SYSTEM_FONT, *P_PREF_SYSTEM_FONT;

Orientation

prOrientation is the resource id for the screen orientation. Reads and writes of this id use a P_U8, whose values are defined below.

Changing this resource (via msgResWriteData) will cause the system to layout after notification of observers, which is expensive. This is done by doing an ObjectPost of msgPrefsLayoutSystem to self. As a result, clsPreferences will compare this resource to the previous value to prevent layout and observer notification if the write did not change the value.

#define tagPrOrientation MakeWknResId(clsPreferences, 3)
#define prOrientation tagPrOrientation
#define prPortrait 0 // Portrait mode
#define prLandscape 1 // Landscape mode
#define prPortraitReversed 2 // Portrait mode (rotated 180 degrees)
#define prLandscapeReversed 3 // Landscape mode (rotated 180 degrees)

Bell

prBell is the resource id for ringing the warning bell. It reads and writes a P_U8, whose values are defined below. prBell is

#define tagPrBell MakeWknResId(clsPreferences, 5)
#define prBell tagPrBell
#define prBellOn 0 // Ring the bell
#define prBellOff 1 // Don’t ring the bell
Writing Style

`prWritingStyle` is the resource id for the handwriting preference style. Reads and writes of this id use a `P_U8`, whose values are defined below.

```c
#define tagPrWritingStyle MakeWknResId(clsPreferences, 6)
#define prWritingStyle tagPrWritingStyle
#define prMixedCase 0 // Mixed case writer
#define prCapsOnly 1 // All caps writer
```

Date Format

`prDateFormat` is the resource id for the desired date format. Reads and writes use a `P_U8`, whose values are defined below. This preference will affect the format of the string returned from `PrefsDateToString`.

```c
#define tagPrDateFormat MakeWknResId(clsPreferences, 7)
#define prDateFormat tagPrDateFormat
#define prDateMDYFull 0 // January 15, 1990
#define prDateMDYAbbre 1 // Jan. 15, 1990
#define prDateMDYSlash 2 // 1/15/90
#define prDateMDYHyphe 3 // 1-15-90
#define prDateMDYDot 8 // 1.15.90
#define prDateDMYFull 4 // 15 January 1990
#define prDateDMYAbbre 5 // 15 Jan. 1990
#define prDateDMYSlash 6 // 15/1/90
#define prDateDMYHyphe 7 // 15-1-90
#define prDateDMYDot 9 // 15.1.90
```

Gesture Timeout

`prGestureTimeout` is the resource id for the gesture timeout, and is measured in 1/100's of a second. Reads and writes of this id use a `P_U16` whose meaning is 1/100's of a second.

```c
#define tagPrGestureTimeout MakeWknResId(clsPreferences, 9)
#define prGestureTimeout tagPrGestureTimeout
```

Line Height

`prLineHeight` is the resource id for the ruled line writing line height in edit pads. Reads and writes of this id use a `P_U16`, whose meaning is 1/100th's of an inch. Changing this preference only affects newly created ruled pads.

```c
#define tagPrLineHeight MakeWknResId(clsPreferences, 10)
#define prLineHeight tagPrLineHeight
```

Auto Suspend

`tagPrAutoSuspend` is the resource id for auto suspend timeout. Reads and writes of this id use a `P_U16`, whose units are minutes. If the value is 0, the machine will not be auto suspended.

Machines that do not support auto suspend use the auto suspend preference for the auto shutdown timeout.

```c
#define tagPrAutoSuspend MakeWknResId(clsPreferences, 11)
```
**Known Preferences in the System**

**Auto Shutdown**

tagPrAutoShutdown is the resource id for auto shutdown timeout. Reads and writes of this id use a P_U16, whose units are hundredths of hours. If the value is 0, the machine will not auto shutdown.

Machines that do not support auto suspend use the auto suspend timeout preference for auto shutdown.

#define tagPrAutoShutdown MakeWknResId(clsPreferences, 28)

**Power Management**

prPowerManagement is the resource id that indicates if PenPoint should attempt to limit the computer's power consumption by turning off inactive devices.

#define tagPrPowerManagement MakeWknResId(clsPreferences, 27)
#define prPowerManagement 0 // power management not attempted
#define prPowerManagementOn 1 // power management attempted

**Floating Allowed**

prDocFloating is the resource id that indicates if documents can be floated. Reads and writes of this id use a P_U8, whose meaning is defined below.

#define tagPrDocFloating MakeWknResId(clsPreferences, 12)
#define prDocFloating 0 // document floating not allowed
#define prDocFloatingOn 1 // document floating allowed

**Zooming Allowed**

prDocZooming is the resource id that indicates if documents can be zoomed. Reads and writes of this id use a P_U8, whose meaning is defined below.

#define tagPrDocZooming MakeWknResId(clsPreferences, 13)
#define prDocZoomingOn 1 // document zooming allowed

**Left/Right Handed**

prHandPreference is the resource id that indicates a left handed or right handed user. Reads and writes of this id use a P_U8, whose meaning is defined below.

Changing this resource (via msgResWriteData) will cause the system to layout after notification of observers, which is expensive. This is done by doing an ObjectPost of msgPrefsLayoutSystem to self. As a result, clsPreferences will compare this resource to the previous value to prevent layout and observer notification if the write did not change the value.

#define tagPrHandPreference MakeWknResId(clsPreferences, 14)
#define prHandPreference 0 // Left Handed writer
#define prRightHanded 1 // Right Handed writer

**Scroll Margins Style**

prScrollMargins is the resource id that indicates a "full" vs. "light" scroll bars. Reads and writes of this id use a P_U8, whose meaning is defined below.
Changing this resource (via msgResWriteData) will cause the system to layout after notification of observers, which is expensive. This is done by doing an ObjectPost of msgPrefsLayoutSystem to self. As a result, clsPreferences will compare this resource to the previous value to prevent layout and observer notification if the write did not change the value.

#define tagPrScrollMargins
#define prScrollMargins
#define prScrollMarginsFull 0
#define prScrollMarginsLight 1

Character Box Width

prCharBoxWidth is the resource indicating the width of char boxes for boxed writing fields. Reads and writes of this id use a P_U8, whose meaning is the width of the box in points. This preference only affects newly created character boxes.

#define tagPrCharBoxWidth MakeWknResId(clsPreferences, 15)
#define prCharBoxWidth tagPrCharBoxWidth

Character Box Height

prCharBoxHeight is the resource id indicating the height of char boxes for boxed writing fields. Reads and writes of this id use a P_U8, whose meaning is the height of the char box in points. This preference only affects newly created character boxes.

#define tagPrCharBoxHeight MakeWknResId(clsPreferences, 16)
#define prCharBoxHeight tagPrCharBoxHeight

Hand Writing Timeout

prHWXTimeout is the resource id indicating the handwriting timeout. Reads and writes of this id use a P_U16 whose meaning is 1/100’s of a second.

#define tagPrHWXTimeout MakeWknResId(clsPreferences, 17)
#define prHWXTimeout tagPrHWXTimeout

Input Pad Style

prInputPadStyle is the resource id indicating the preferred style of handwriting pads. Reads and writes of this id use a P_U8, whose meaning is defined below.

#define tagPrInputPadStyle MakeWknResId(clsPreferences, 18)
#define prInputPadStyle tagPrInputPadStyle
#define prInputPadStyleBoxed 0 // Pad styles are boxed
#define prInputPadStyleRuled 1 // Pad styles are Ruled
#define prInputPadStyleRuledAndBoxed 2 // Pad styles are boxed-->ruled
#define prInputPadStyleSegmented 0 // Obsolete

Hold Timeout

prPenHoldTimeout is the resource id for the press hold timeout. Reads and writes of this id use a P_U16 whose meaning is 1/100’s of a second.

#define tagPrPenHoldTimeout MakeWknResId(clsPreferences, 19)
#define prPenHoldTimeout tagPrPenHoldTimeout
**Pen Cursor**

`prPenCursor` is the resource id for whether the cursor is off or on. Reads and writes of this id use a `P_U8`, whose meaning is defined below.

```c
#define tagPrPenCursor MakeWknResId(clsPreferences, 20)
#define prPenCursor #tagPrPenCursor
#define prPenCursorOff 0 // Pen cursor should be off
#define prPenCursorOn 1 // Pen cursor should be on
```

**Time Format**

`prTimeFormat` is the resource id for the preferred time format (military or civilian). Reads and writes of this id use a `P_U8`, whose meaning is defined below. This preference will affect the returned string from `PrefsTimeToString`.

```c
#define tagPrTimeFormat MakeWknResId(clsPreferences, 21)
#define prTimeFormat #tagPrTimeFormat
#define prTime12Hour 0 // Display 12 hour times
#define prTime24Hour 1 // Display 24 hour times
```

**Display Seconds**

`prTimeSeconds` is the resource id indicating if seconds should be displayed or not. Reads and writes of this id use a `P_U8`, whose meaning is defined below. This preference will affect the returned string from `PrefsTimeToString`.

```c
#define tagPrTimeSeconds MakeWknResId(clsPreferences, 22)
#define prTimeSeconds #tagPrTimeSeconds
#define prTimeSecondsDisplay 0 // Display seconds in time
#define prTimeSecondsOff 1 // Don't display seconds in time
```

**Time**

`prTime` is the resource id for the system time. Reads and writes of this ID use a `P_PREF_TIME_INFO`, containing the current time information.

```c
#define tagPrTime MakeWknResId(clsPreferences, 23)
#define prTime #tagPrTime
typedef union P(pref_time_mode) {
   OS_SET_TIME_MODE writeMode; // In: which attributes to set (for write only)
} P(pref_time_mode);
typedef struct P(pref_time_info) {
   P(pref_time_mode) mode; // In: read or write mode
   OS_DATE_TIME dateTime; // In/Out: date and time information
} P(pref_time_info), *P PREF_TIME_INFO;
```

**Primary Input**

`prPrimaryInput` is the resource id defining the primary input device. Reads and writes of this id use a `P_U8`, whose meaning is defined below.

```c
#define tagPrPrimaryInput MakeWknResId(clsPreferences, 24)
#define prPrimaryInput #tagPrPrimaryInput
#define prPrimaryInputPen 0 // Primary input is with the pen
#define prPrimaryInputKbd 1 // Primary input is with a keyboard
```
Unrecognized Character

prUnrecCharacter is the resource id used for the unrecognized character glyph. Reads and writes of this id use a P_U8, whose meaning is defined below.

```c
#define tagPrUnrecCharacter MakeWknResId(clsPreferences, 25)
#define prUnrecCharacter  tagPrUnrecCharacter
#define prUnrecCharacterQuestion  0
#define prUnrecCharacterUnder    1
```

Messages

msgNew

Creates a new preferences object.

Takes P_PREFS_NEW, returns STATUS. Category: class message.

```c
typedef struct PREFS_NEW_ONLY {
    P_CHAR     pPrefSet;  // Preference set name
} PREFS_NEW_ONLY, *P_PREFS_NEW_ONLY;
#define prefsNewFields         
    objectNewFields         
    P_PREFS_NEW_ONLY       
    prefs;
typedef struct PREFS_NEW {
    prefsNewFields
} PREFS_NEW, *P_PREFS_NEW;
```

Comments

This message should not be called by clients. Creates a preferences object. If pPrefSet is pNull, the list will start out empty. Otherwise, pPrefSet is expected to be an already installed file title in the preferences directory.

msgPrefsPreferenceChanged

Sent to observers when a preference has changed.

Takes P_PREF_CHANGED, returns STATUS. Category: observer notification.

```c
#define msgPrefsPreferenceChanged MsgNoError(MakeMsg(clsPreferences, 1))

typedef struct PREF_CHANGED {
    OBJECT manager;     // Sender of the notification (theSystemPreferences)
    RES_ID prefId;      // resId of preference that changed
} PREF_CHANGED, *P_PREF_CHANGED;
```

Comments

Sent to observers. Notifies observers that a given preference has changed. Notifies with the manager (usually theSystemPreferences, as there are no other pre-defined instances of clsPreferences), and the RES_ID of the preference that has changed.

msgPrefsLayoutSystem

Causes the system to re-layout and re-paint.

Takes NULL, returns STATUS.

```c
#define msgPrefsLayoutSystem MakeMsg(clsPreferences, 5)
```

Comments

Causes the entire system to layout. If msgPrefsWritingMany has not been called, posted to self when clsPreferences receives msgResWriteData and a new value has been written for prSystemFont, prUserFont, prOrientation, prHandPreference, or prScrollMargins. If msgPrefsWritingMany has been
called, the layout will occur when `msgPrefsWritingDone` is called. Will be sent to observers when immediately before a layout of the system occurs due to a preference change.

See Also `msgPrefsWritingMany`

### `msgPrefsWritingMany`

Indicates several preferences are to be written in succession.

Takes NULL, returns STATUS.

```c
#define msgPrefsWritingMany MakeMsg(clsPreferences, 6)
```

Comments

Causes `clsPreferences` to delay the self-posting of `msgPrefsLayoutSystem` until it receives `msgPrefsWritingDone`. Useful when writing several preference changes at once, and the client does not want the system laying out several times. If, after this message is received, a `msgResWrite` of `prSystemFont`, `prUserFont`, `prOrientation`, `prHandPreference`, or `prScrollMargins` is received, `clsPreferences` will self-post `msgPrefsLayoutSystem` when `msgPrefsWritingDone` is received. After `msgPrefsWritingDone` is received, any other `msgResWrite` of these preferences will cause an immediate layout unless this message is sent again. Will be sent to observers to allow them to be aware that several preferences are being written.

See Also `msgPrefsWritingDone`

### `msgPrefsWritingDone`

Indicates completion of writing several preferences.

Takes NULL, returns STATUS.

```c
#define msgPrefsWritingDone MakeMsg(clsPreferences, 7)
```

Comments

Causes the system to layout if necessary by self-posting `msgPrefsLayoutSystem`. You should send this message in conjunction with `msgPrefsWritingMany` to indicate that writing of successive preferences is complete. If a `msgResWrite` of `prSystemFont`, `prUserFont`, `prOrientation`, `prHandPreference`, or `prScrollMargins` with a new value has been done, layout will occur at this time. Will be sent to observers to indicate that a series of preferences writes have been completed.

See Also `msgPrefsWritingMany`

### Public Functions

#### `PrefsSysFontInfo`

Passes back the system and user font information.

Returns void.

**Arguments**

```c
typedef struct PREF_SYSTEM_FONT_INFO {
  U8 scale;
  U16 sysFontId;
  U16 userFontId;
} PREF_SYSTEM_FONT_INFO, *P_PREF_SYSTEM_FONT_INFO;
```

**Function Prototype**

```c
void EXPORTED PrefsSysFontInfo (    
  P_PREF_SYSTEM_FONT_INFO pFontInfo);
```

**Comments**

This function can be used to read all font information stored in the preferences file at one time. Equivalent functionality exists with `msgResRead`. This function is provided for convenience.
**PrefsDateToString**

Returns a pointer to the string containing a formatted date.

Returns P_CHAR.

```
#define prefsMaxDate 19
```

**Function Prototype**

```
P_CHAR EXPORTED PrefsDateToString (P_OS_DATE_TIME pTime, P_CHAR pStr);
```

**Comments**

This function will return a string containing the ASCII representation of the formatted date based on the current user-preference for date. Puts the date into the string passed in. The longest possible string is 18 characters (19 including the terminating 0) given the CURRENT formats. If additional formats are added, this may increase.

---

**PrefsTimeToString**

Returns a pointer to the string containing a formatted time.

Returns P_CHAR.

```
#define prefsMaxTime 11
```

**Function Prototype**

```
P_CHAR EXPORTED PrefsTimeToString (P_OS_DATE_TIME pTime, P_CHAR pStr);
```

**Comments**

This function will return a string containing the ASCII representation of the time based on the current user preferences for time. Puts the time into the string passed in, and returns the string pointer. The longest possible string is 10 characters (11 including the terminating 0) given the current time formats. If additional formats are added, this may increase.
This file contains definitions for input to the resource compiler.

The resource compiler is a program which runs under MS-DOS. In conjunction with your resource compiler input and the C compiler it will create a PenPoint resource file.

**NOTE:** THIS IS A MSDOS INCLUDE FILE, DO NOT CHANGE IT TO BE PENPOINT COMPATIBLE.

```c
#ifndef RESCMPLR_INCLUDED
#define RESCMPLR_INCLUDED
#ifndef RESFILE_INCLUDED
#include <resfile.h>
#endif
#endif
```

### Common #defines and typedefs

#### Types

Prototype for the client-supplied agent writing routine. If you wish to supply your own agent writing routine then write a routine of type `P_AGENT_TYPE` and supply the address to the routine in the field `pAgentWriteProc` of `RC_INPUT`. Your routine should write out (using fwrite to file) its representation of the data described by `pResInput` (and optionally also by `pAgentData`).

```c
typedef void (PASCAL * P_AGENT_WRITE) (P_UNKNOWN file, struct RC_INPUT * pResInput, P_UNKNOWN pAgentData, U32 spare1, U32 spare2) ;
```

The resource compiler uses the information supplied by `RC_INPUT` to create resources. Typically only the first four or five fields of `RC_INPUT` are used. At a minimum you should set `resId`, `pData` and `dataLen`. You do not need to set `dataLen` if you set `agent` to `resStringResAgent` or `resStringArrayResAgent` (the resource compiler will infer `dataLen` from `pData`). You should set `agent` if you do not want the default resource data agent. You should set `minSysVersion` if it has a non-zero value. You may set `objectData` to true in the rare case that an object resource is being created by the resource compiler. You should set `pAgentWriteProc` and optionally `pAgentWriteData` if you are providing your own routine to write the resource data to the resource file.

```c
typedef struct RC_INPUT {
    RES_ID resid; // the resource ID
    PUNKNOWN pData; // points to data
    dataLen; // length of data
    agent; // usually resDefaultResAgent
    minSysVersion; // min sys version for resource
    reserved;
    BOOLEAN objectData; // usually false
    P_AGENT_WRITE pAgentWriteProc; // pNull, unless supplying routine
    PUNKNOWN pAgentWriteData; // usually pNull
} RC_INPUT, *P_RC_INPUT, **PP_RC_INPUT;
```
If you use `resTaggedStringArrayResAgent` as the agent for a resource. Then the data must be a list of RC_TAGGED_STRINGs. This is converted into a linear string array and the filed using the `resStringArrayResAgent` agent.

```c
#define resTaggedStringArrayResAgent ((UID)MakeTag(clsResFile, 0xff))
typedef struct RC_TAGGED_STRING {
    TAG tag;
    P_STRING pString;
} RC_TAGGED_STRING, *P_RC_TAGGED_STRING;
```

**Public variable**

`resInput` is an exported variable that the resource compiler expects. Each element in the `resInput` array is a pointer to a structure describing the next resource. The list must be terminated with a null pointer.

```c
extern P_RC_INPUT resInput[];  // Resource compiler input
```

**Example**

Here is example input for `rescmplr` (or `rc`):

```c
// Resource ids
#define residRfANumber MakeWknResId(clsExample, 1)
#define residRfAString MakeWknResId(clsExample, 2)
#define residRfAStringArray MakeWknResId(clsExample, 3)
#define residRfATaggedStringArray MakeWknResId(clsExample, 4)

#define tagExampleErrorBogus MakeTag(clsExample, 0)
#define tagExampleErrorWrong MakeTag(clsExample, 1)
#define tagExampleErrorAgain MakeTag(clsExample, 2)

// A number.
static U16 aNumber = 1;

// A string array.
static P_CHAR errorTextData[] = {
    "This is bogus.\n",  // Define end of string array.
    "You got it wrong.\n",
    "I think you need to try again.\n",
    pNull
};

// A tagged string array.
// This is equivalent to the above string array even thought the
// elements are in a different order.
static P_RC_TAGGED_STRING errorTextTaggedData[] = {
    tagExampleErrorWrong,  // Define end of string array.
    tagExampleErrorAgain,  // "You got it wrong."
    tagExampleErrorBogus,  // "I think you need to try again."
    pNull
};

// Res compiler input for aNumber.
static RC_INPUT aNumberRes = {
    residRfANumber,
    &aNumber,
    sizeof(aNumber)
};
```
// Res compiler input for aString.
static RC_INPUT aStringRes = {
  resIdRfAString,
  "Sample string",
  0, // Size inferred by res compiler.
  resStringResAgent
};

// Res compiler input for aStringArray.
static RC_INPUT aStringArrayRes = {
  resIdRfAStringArray,
  errorTextData,
  0, // Size inferred by res compiler.
  resStringArrayResAgent
};

// Res compiler input for aTaggedStringArray.
static RC_INPUT aTaggedStringArrayRes = {
  resIdRfATaggedStringArray,
  errorTextTaggedData,
  0, // Size inferred by res compiler.
  resTaggedStringArrayResAgent
};

// Input for resource compiler.
RC_INPUT resInput [] = {
  &aNumberRes,
  &aStringRes,
  &aStringArrayRes,
  &aTaggedStringArrayRes,
  pNull
};
RESFILE.H

This file contains the API definition for clsResFile.
clsResFile inherits from clsFileHandle.
Provides resource and object filing support.
theSystemResFile is a well known instance of clsResFile.

clsResList inherits from clsList.
ResLists are lists of resource files that act like a single resource file for reading and searching (but not writing).

theProcessResList is a process well known instance of clsResList.
A resource file maintains a collection of 'resources' each identified by a 'resource ID'. A resource is filed data or a filed object. The types of data supported are: byte array, string, and array of strings. It is also possible to create an 'agent' that reads and writes other kinds of data.

A resource ID is a 32 bit TAG used as a unique (per file) key to identify and select a desired resource.

Overview

Resource files are used in three general ways: filing & unfiling objects, reading theProcessResList for configuration and customization information, and application specific data storage.

- The most common case of filing & unfiling objects is a page turn, which needs to save the state of a running process on the disk, and restore the state of another process from the disk. This is done by (un)filing the application framework, which (if everything is set up correctly) (un)files directly or indirectly all the objects that make up the state of the process.

Filling of an object is initiated with msgResWriteObject which ends up sending msgSave to the object. The save procedure uses msgStreamWrite (everything except objects) and msgResPutObject (objects) to write out its instance data. Unfiling of the object is initiated with msgResReadObject which sends msgRestore to the (newly created) object. The restore procedure uses msgStreamRead and msgResGetObject to read its instance data back in.

- theProcessResList is used for several reasons: to allow text to be stored separately from the code, to store pre-built UI objects, to allow applications to override system provided items, to provide a central set of system wide preferences, etc. To do this it normally (inside an application) contains four resource files: DOC.RES (specific to the document), APP.RES (specific to the application), current system preferences file, and PENPOINT.RES (system wide resource file). They are searched in the order listed above. There are some utility functions to access theProcessResList, see RESUTIL.H for more information on them.

- There are many other ways to use resource files, but they are application specific. If you think you have a use for resource files, it is worth checking out, but do be careful, resource files are designed and optimized for the first two uses, and do not work well for everything that it at first seems like they should.
How a Resource ID is put together

The fields in a Resource ID:

TagNum (which resource object) = 8 bits
Flags (see below) = 2 bits
Admin (as usual) = 20 or 19 bits
Scope (as usual) = 1 or 2 bits

They are laid out this way:

| Name: 0|tagNum |F| Admin+Scope |
|--------|-------|-----------|
| Size: 1|8|2| 20+1 or 19+2 |

The flags are interpreted as follows:

0 Well-Known Resource ID
1 Dynamic Resource ID
2 Well-Known List Resource ID
3 RESERVED

The Well-Knowns used here are the same ones used in other tags. This gives us three possible scopes:
global, process and local. Because resource files are not tied to a process context, there is no difference
between the global and process Well-Knowns. System and service classes should only use there own well
known. Applications can not only use the well knowns for there own classes, they can also use all local
well known values.

Well-Known Resource Ids (flag == 0) can be used to store any kind of resource.

The Dynamic Resource IDs (flag == 1) are used by the resource file in msgResPutObject to file nested
objects. It is also possible for other code to allocate them using msgResNextDynResId. They may be
used to file any kind of resource. We get 29 bits worth of Dynamic Resource IDs by combining the
tagNum, admin and scope fields.

Well-Known List Resource IDs (flag == 2) must be used with list resources to allow the Indexed
Resource IDs (see below) to work. The only list resource defined by GO is the string array, but it is
possible to define others. The tagNum field is split into two fields for List Resource IDs.

The fields in a List Resource ID:

Group of lists = 6 bits
List in group = 2 bits
Flags (always set to 0x2) = 2 bits
Admin (as usual) = 20 or 19 bits
Scope (as usual) = 1 or 2 bits

They are laid out this way:

| Name: 0|Grp |L|2| Admin+Scope |
|--------|-----|2|-----------|
| Size: 1|6|2| 20+1 or 19+2 |

| Name: 0|tagNum |F| Admin+Scope |
|--------|-------|-----------|
| Size: 1|8|2| 20+1 or 19+2 |
The Groups are allocated as follows:
00 - 1F  AVAILABLE TO DEVELOPERS
TK Table Lists
Standard Message Lists
Quick Help Lists
3F  RESERVED FOR GO

What an Indexed Resource ID is

Indexed Resource IDs are used to access list resources. They are NOT Resource IDs. Each must be converted into the List Resource ID of the desired list plus an index into the list to fetch the desired data.

The fields in an Indexed Resource ID:
TagNum (index into list)  = 8 bits
Flags (which list)       = 2 bits
Admin (as usual)        = 20 or 19 bits
Scope (as usual)        = 1 or 2 bits

They are laid out this way:

```
| Name: Delim TagNum | Delim Flags | Delim Admin+Scope |
|-------------------+-------------+-------------------|
| Size:             | Delim 8     |                  |
|                   | Delim 2     |                  |
|                   | Delim 20+1  |                  |
|                   | Delim 19+2  |                  |
```

You will note that this provides eight bits not provided by a List Resource ID (the index) and is missing eight bits needed by it (the flags and group).

The eight bits of index allow each list to contain up to 256 items. Actually they can have more, but only the first 256 can be accessed this way. Since there are four lists for each Well-Known, it is possible to access up to 1,024 items per group per Well-Known.

We provide the missing bits as follows. Since we always map to a List Resource ID we know that the flags will be set to 0x2. Which group to use is determined by which API it is used with. Thus, the passing the same Indexed Resource ID to both Quick help and a TK table will result in different data items being used.

Warnings to those going off the beaten path.

The description above gives the standard way of allocating resource IDs. While there is special support for using them this way, and some other parts of the system in fact require this usage, the resource file itself does not care. The only time it puts a special interpretation on a resource ID is for well-known-object resource IDs. They have the top (sign) bit set to one. These are automatically created by the resource file, and to avoid trouble, should never be created by anything else.

Dynamic resource IDs are based off of a 29-bit count, and gaps are not reused. Because of this it is possible to run out. While this will not happen in 'normal' use, it is possible for uses that seem reasonable. So if you use them for anything other than normal object filing, or are repeatedly filing objects, make sure you do not run into this.
When an object resource is deleted from a resource file, the other objects it filed are NOT deleted, and there is no easy way of finding them to delete them. Because of this, repeatedly filing objects will result in the file growing without bound unless you work very hard to prevent it.

Opening multiple handles on the same resource file has some limitations. It is possible to have as many read-only handles as desired, as long as there are no writable handles. If there is a writable handle, no other handles may be opened. This is due to a limitation of the current implementation. It maintains index information into the file on a per handle basis. If writing was allowed with multiple handles open, these tables would become invalid resulting in fatal errors of many kinds.

While it is possible to use a resource file as a kind of mini-database, it was not designed or optimized for such a use. So, don't be surprised if you find it is not up to the task you would like to use it for.

ResFile Debugging Flags (Shared with Penpoint kernel & fs)

ResFile flag is 'G', values are:

1-80 = Used by PenPoint Kernel (see os.h)
-800 = Used by File System (see fs.h)

1000 = Turns on debugging info for reading and writing resources.
- Turns on timing stats
- Turns on debugging info for intercepted Stream & FS messages.
- TBD

#ifndef RESFILE_INCLUDED
#define RESFILE_INCLUDED
#endif
#ifndef UUID_INCLUDED
#include <uuid.h>
#endif
#ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif
#ifndef OSHARP_INCLUDED
#include <osheap.h>
#endif
#ifndef LIST_INCLUDED
#include <list.h>
#endif
#ifndef FS_INCLUDED
#include <fs.h>
#endif

Common #defines and typedefs

These are used to define resource IDs, both well known (client-defined) and dynamic (See uuid.h for comparison). Note that the count used for the dynamic resource ID's is managed internal to the resource file, and no attempt should be made to create them elsewhere.

#define resFlagsWkn 0x0
#define resFlagsDyn 0x1
#define resFlagsLists 0x2
#define resFlagsSpare 0x3
#define resFlagWknObj (RES_ID)0x80000000)
#define MakeWknResld(wkn, i) \  
  MakeTagWithFlags(wkn, i, resFlagsWkn)
#define MakeDynResld(count) \  
  MakeTagWithFlags(((U32)(count))>>8, ((count)&0xFF), resFlagsDyn)
#define MakeListResld(wkn, grp, lst) \  
  MakeTagWithFlags(wkn, (((U32)(grp))<<2)+((lst)&0x03)), resFlagsLists)
#define MakeWknObjResld(obj) ((RES_ID)(obj) | resFlagWknObj)

Extract the pieces from resource IDs.
#define ResWknObjResld(resld) ((OBJECT)((resld) & ~resFlagWknObj))
#define ResDynldCount(resld) (WKNValue(resld)<<8 | Tag(resld))
#define ResListGroup(resld) (Tag(resld) >> 2)
#define ResListList(resld) (Tag(resld) & 0x3)

Tests on resource ID's
#define WknObjResld(resld) (!WknObjResId(resld) && TagFlags(resld) != resFlagsWkn)
#define WknItemResld(resld) (!WknObjResId(resld) && TagFlags(resld) == resFlagsWkn)
#define WknListResld(resld) (!WknObjResId(resld) && TagFlags(resld) == resFlagsLists)
#define DynResld(resld) (!WknObjResld(resld) && TagFlags(resld) == resFlagsDyn)

Constants
#define resNilResld NIL(RES_ID)

OBOLETE Resource IDs do NOT use.
#define residRfSystemVersion MakeWknResld(clsResFile, 1)
#define residRfApplicationVersion MakeWknResld(clsResFile, 2)

How to make a Indexed resource ID.
#define MakeIndexedResld(wkn, list, index) \  
  MakeTagWithFlags(wkn, index, list)

The group identifiers used to convert from Indexed resource IDs to normal resource IDs. Values from 0x00 to 0x1F are available for use by applications. Values from 0x20 to 0x3F are reserved to the system.
#define resGrpTK 0x20
#define resGrpStdMsg 0x21
#define resGrpQhelp 0x22

Predefined Resource Agents

These are used by both the resource compiler to define data resources and by msgResWriteData to dynamically write a resource.

// Don't use these definitions, use the derived values below
#define resDefaultObjAgent 3 // Use resObjectResAgent
#define resDefaultDataAgent 4 // Use resDataResAgent
#define resStringAgent 5 // Use resStringResAgent
#define resStringArrayAgent 6 // Use resStringArrayResAgent
#define MakePrivateResAgent(x) \  
  ((UID)MakeTag(clsResFile, x))

// These are the pre-defined resource types
#define resObjectResAgent objNull
#define resObjectResAgent MakePrivateResAgent(resDefaultObjAgent)
#define resDataResAgent MakePrivateResAgent(resDefaultDataAgent)
#define resStringResAgent MakePrivateResAgent(resStringAgent)
#define resStringArrayResAgent MakePrivateResAgent(resStringArrayAgent)
# Status Codes

```c
#define stsResResourceNotFound MakeStatus(clsResFile, 1)
#define stsResNotDataResource MakeStatus(clsResFile, 2)
#define stsResNotObjectResource MakeStatus(clsResFile, 3)
#define stsResBufferTooSmall MakeStatus(clsResFile, 4)
#define stsResNotFullyRead MakeStatus(clsResFile, 5)
#define stsResGetNotFromRestore MakeStatus(clsResFile, 6)
#define stsResPutNotFromSave MakeStatus(clsResFile, 7)
// removed unused MakeStatus(clsResFile, 8)
#define stsResWriteObjDynamicClass MakeStatus(clsResFile, 9)
// removed unused MakeStatus(clsResFile, 10)
#define stsResCompactInReadOrWrite MakeStatus(clsResFile, 11)
#define stsResIncorrectFileType MakeStatus(clsResFile, 12)
#define stsResFileCorrupt MakeStatus(clsResFile, 13)
#define stsResResourceTooBig MakeStatus(clsResFile, 14)
#define stsResOutOfDynResIds MakeStatus(clsResFile, 15)
```

## Types

```c
typedef OBJECT RES_FILE, *P_RES_FILE;
typedef OBJECT RES_LIST, *P_RES_LIST;
```

NOTE: That RES_ID is already defined in clsmgr.h because it is referenced by msgSave & msgRestore:

```c
typedef TAG RES_ID, *P_RES_ID;  // Resource ID
```

**Modes used in msgNew to control the creation of the resource file.**

```c
Enum16 (RES_NEW_MODE) {
    // Will the file handle be shared? Also guarantees concurrence
    resSharedResFile     = flag0,
    // Remove "deleted" fields on close
    resCompactOnClose   = flag1,
    // Compact file when ratio of deleted to non-deleted reaches compactRatio.
    resCompactAuto      = flag2,
    // Check to see that system version is new enough for resources.
    resVerifyVersions   = flag3,
    // Allow unsafe opens, internal use only.
    resUnsafeOpen       = flag4,
    // Default - No Concurrence, compact on close, verify versions.
    resNewDefault       = resCompactOnClose | resVerifyVersions
};
```

**Duplicate object checking flag for reading objects.**

```c
Enum16 (RES_READ_OBJ_MODE) {
    resReadObjectOnce   = 0,  // Should object resource be read once?
    resReadObjectMany   = 1   // Should object resource be read many times?
};
```

**Duplicate object checking flag for writing objects.**

```c
Enum16 (RES_WRITE_OBJ_MODE) {
    resWriteObjectOnce  = 0,  // Should object resource be written once?
    resWriteObjectMany  = 1   // Should object resource be written many?
};
```

**Mode used to control msgResEnumResources.**

```c
Enum16 (RES_ENUM_MODE) {
    resEnumAll          = 0,  // Enumerate all resource entries?
    resEnumByResIdClass = 1,  // Enumerate by wkn resource ID admin field?
    resEnumByObjectClass = 2, // Enumerate by object resource's class?
    resEnumByObjectUID  = 3,  // Enumerate by object resource's uid?
    resEnumByAgent      = 4,  // Enumerate by resource's agent?
    resEnumNext         = flag14, // Or in to enumerate the next item.
    resEnumDefault      = resEnumAll  // Default - all resources.
};
```
// Internal flag used to enumerate across resource lists.
#define resEnumNextFile 0x8000

// Indexed resource IDs.
typedef TAG IX_RES_ID, *P_IX_RES_ID;

Class ResFile Messages

msgNew
Creates a resource file object.

Takes P_RES_FILE_NEW, returns STATUS. Category: class message.

Arguments
typedef struct RES_FILE_NEW_ONLY {
    RES_NEW_MODE mode;
    U16 compactMinimum;
    U16 compactRatio;
    U32 spare1;
    U32 spare2;
} RES_FILE_NEW_ONLY, *P_RES_FILE_NEW_ONLY;
#define resFileNewFields
    fsNewFields
    RES_FILE_NEW_ONLY resFile;
typedef struct RES_FILE_NEW {
    resFileNewFields
} RES_FILE_NEW, *P_RES_FILE_NEW;

Return Value
stsIncompatibleVersion Filed data is incompatible with system.
stResIncorrectFileType File is not a resource file.
stResFileCorrupt Size or contents of the file are not valid.
stFSAccessDenied Incompatible with existing handles(*)

(*) Note that there can be only one open handle to a writable resource file. The file mode is automatically set to enforce this.

A resource file compacts itself at close time if the resCompactOnClose flag was set in pNew->resFile.mode.

If the resCompactAuto flag is set in pArgs->res.mode then it compacts itself when a resource is written or deleted, if the number of records is greater than compactMinimum and the number of deleted records is greater than compactRatio percent of the records in the file.

For example, a value of 10 for compactMinimum and 50 for compactRatio implies that compaction should happen whenever there are more than 10 resources in the resource file and 50% of them have been marked as deleted.

msgNewDefaults
Initializes the RES_FILE_NEW structure to default values.

Takes P_RES_FILE_NEW, returns STATUS. Category: class message.

Arguments
typedef struct RES_FILE_NEW {
    resFileNewFields
} RES_FILE_NEW, *P_RES_FILE_NEW;

Zeroes out pArgs->resFile and sets... mode = resNewDefault; compactRatio = 33; compactMinimum = 50;
**msgResFindResource**

Finds a resource in a resource file or a resource list.

Takes P_RES_FIND, returns STATUS.

```c
#define msgResFindResource MakeMsg(clsResFile, 1)
```

**Arguments**

```c
ttypedef struct RES_FIND {
  RES_ID resId; // In: Resource to find
  RES_FILE file; // Out: File location of resource
  UID agent; // Out: Agent of the resource
  U32 offset; // Out: Offset in file (Careful!)
  U16 minSysVersion; // Out: Min sys vers for the resource
  U16 reserved;
} RES_FIND, *p RES_FIND;
```

**Comments**

***This message is obsolete, you should use msgResGetInfo instead.***

This message may be used to determine if a resource exists and to get information about that resource. You must use it before writing or deleting a resource if you do not know which resource file (out of a resource list) contains the resource. This is an improved version of msgResFindResource. It gives a more useful set of values for agent (as in exactly what is in the file), and it returns the size of the resource in the file.

**Return Value**

- **stsBadParam**  resId is a nil resource ID.
- **stsResResourceNotFound**  No resource with the given resId exists.

**msgResGetInfo**

Gets information on a resource in a resource file or a resource list.

Takes P_RES_INFO, returns STATUS.

```c
#define msgResGetInfo MakeMsg(clsResFile, 17)
```

**Arguments**

```c
ttypedef struct RES_INFO {
  RES_ID resId; // In: Resource to find
  RES_FILE file; // Out: File location of resource
  UID agent; // Out: Agent of the resource
  UID objClass; // Out: Class of object (if is object)
  U32 offset; // Out: Offset in file (Careful!)
  U32 size; // Out: Size in file (Careful!)
  U16 minSysVersion; // Out: Min sys vers for the resource
  U16 reserved1;
  U32 reserved;
} RES_INFO, *p RES_INFO;
```

**Comments**

This message may be used to determine if a resource exists and to get information about that resource. You must use it before writing or deleting a resource if you do not know which resource file (out of a resource list) contains the resource (Resource lists only act upon non-destructive messages). This is an improved version of msgResFindResource. It gives a more useful set of values for agent (as in exactly what is in the file), and it returns the size of the resource in the file.

**Return Value**

- **stsBadParam**  resId is a nil resource ID.
- **stsResResourceNotFound**  No resource with the given resId exists.

**msgResReadData**

Reads resource data from a resource file or resource list.

Takes P_RES_READ_DATA, returns STATUS.

```c
#define msgResReadData MakeMsg(clsResFile, 2)
```
typedef struct RES_READ_DATA {
    RES_ID resId;
    OS_HEAP_ID heap;
    P UNKNOWN pData;
    U32 length;
    P UNKNOWN pAgentData;
    U32 sparel;
} RES_READ_DATA, *P_RES_READ_DATA;

This message requires a destination for the read data. There are two choices. You can specify a pointer and a length for the data passed back (heap = null, pData = ptr, length = xx) or you can specify a valid heap from which the resource file will allocate memory for the data (heap = heap ID, pData = doesn't matter, length = doesn't matter). Typically if the size of the data is already known and it is small and short lived, then the data is "allocated" on the stack. Otherwise, the data is allocated on behalf of a heap.

Some resources require additional data to identify the actual data to be passed back. For example, a string arrays resource requires additional information (the index into the array) to find the string to pass back. You specify an index in pAgentData (pAgentData = (P_UNKNOWN)index).

stsBadParam resId is a nil resource ID or reading a string from a string array resource and the index specified in pAgentData is out of range.

stsResResourceNotFound No resource with the given resId exists.

stsResNotDataResource The found resource was an object resource.

stsResBufferTooSmall Supplied buffer isn't big enough to hold data.

See Also

msgResWriteData To write data to resource file.

msgResReadObject To read an object from a resource file.

msgResWriteData

Writes resource data to a file.

Takes P_RES_WRITE_DATA, returns STATUS.

#define msgResWriteData MakeMsg(clsResFile, 3)

typedef struct RES_WRITE_DATA {
    RES_ID resId;
    P UNKNOWN pData;
    U32 length;
    UID agent;
    P UNKNOWN pAgentData;
    U32 sparel;
} RES_WRITE_DATA, *P_RES_WRITE_DATA;

This message writes data to the resource file. If the resource already exists it is marked as deleted and the new data is written to the end of the file.

stsBadParam resId is a nil resource ID.

stsResResourceTooBig Tried to write resource bigger than resource file can handle (16Meg).

See Also

msgResReadData To read data from resource file.

msgResUpdateData To re-write data in a resource file.

msgResWriteObject To write an object to a resource file.
**msgResUpdateData**

Updates existing data resource data.

Takes P_RES_WRITE_DATA, returns STATUS.

```c
#define msgResUpdateData MakeMsg(clsResFile, 4)
```

**Arguments**

```c
typedef struct RES_WRITE_DATA {
    RES_ID resId;       // Data to be written
    P_UNKNOWN pData;    // Optional if agent can compute size
    U32 length;        // Not used by msgResUpdateData
    UID agent;
    P_UNKNOWN pAgentData; // Agent-specific data
    U32 spare1;
} RES_WRITE_DATA, *P_RES_WRITE_DATA;
```

**Comments**

Use this message if you know that a resource already exists and is only being updated. The only advantage of this message over **msgWriteData** is that you don't have to specify the agent.

**Return Value**

- **stsBadParam** resId is a nil resource ID.
- **stsResResourceNotFound** No resource with the given resId exists.
- **stsResNotDataResource** The found resource was an object resource.
- **stsResResourceTooBig** Tried to write resource bigger than resource file can handle (16Meg).

**See Also**

- **msgResReadData** To read data from resource file.
- **msgResWriteData** To write data to a resource file.

**msgResReadObject**

Reads a resource object from a resource file or resource list.

Takes P_RES_READ_OBJECT, returns STATUS.

```c
#define msgResReadObject MakeMsg(clsResFile, 5)
```

**Arguments**

```c
typedef struct RES_READ_OBJECT {
    RES_READ_OBJ_MODE mode;    // Duplicate checking mode
    RES_ID resId;              // Object passed back in new.uid
    OBJECT_NEW objectNew;     // Only for msgResReadObjectWithFlags
    RES_SAVE_RESTORE_FLAGS sysFlags; // Only for msgResReadObjectWithFlags
    U16 appFlags;             // Only for msgResReadObjectWithFlags
    U32 spare1;
} RES_READ_OBJECT, *P_RES_READ_OBJECT;
```

**Comments**

An object must be initialized before it can be read. You must send **msgNewDefault** to clsObject.

There are two modes that can be applied to reading an object resource, **resReadObjectOnce** and **resReadObjectMany**.

Setting mode to **resReadObjectOnce**, passed back the object that is associated with the resource stored in the resource file (per open). This guarantees that all filed references to a given object refer to the same object. This is the mode to use if you are unfiling data in a **msgRestore** procedure. There are other uses of it, but they can be very tricky, so make sure you read all of the documentation and understand it thoroughly before you try to use this any place other than a **msgSave** procedure.

Setting mode to **resReadObjectMany**, passes back a new copy of the object without regard as to whether the object has already been read in before or not. This guarantees that each reader gets his own unique instance of the object. This is the mode to use if you are reading an object resource "template" (the normal case).
#define sampleResId MakeWknResId(clsXXX, 17)
readObj.resId = sampleResId;
readObj.mode = resReadObjectMany;
ObjCallRet(msgNewDefaults, clsObject, &readObj.objectNew, status);
status = ObjCallWarn(msgResReadObject, file, &readObj);
object = readObj.objectNew.uid;

## msgResWriteObject
Writes a resource object to a file.

Takes P_RES_WRITE_OBJECT, returns STATUS.

```c
#define msgResWriteObject MakeMsg(clsResFile, 6)
```

### Arguments
- typedef struct RES_WRITE_OBJECT {
  - RES_WRITE_OBJECT_MODE mode; // Duplicate checking mode
  - RES_ID resId;
  - OBJECT object; // Object to write
  - RES_SAVE_RESTORE_FLAGS sysFlags; // Only for msgResWriteObjectWithFlags
  - U16 appFlags; // Only for msgResWriteObjectWithFlags
  - U32 spare1;
} RES_WRITE_OBJECT, *P_RES_WRITE_OBJECT;

### Comments
There are two modes that can be applied to writing an object resource, resWriteObjectOnce and resWriteObjectMany.

Setting mode to resWriteObjectOnce, will only write the object to the resource file once (per open). This guarantees that all field references to a given object refer to the same object. This is the mode is used by msgResPutObject, and should be used by you if you bypass it and use msgResWriteObject directly in a msgSave procedure. There are other uses of it, but they can be very tricky, so make sure you read all of the documentation and understand it thoroughly before you try to use this any place other than a msgSave procedure.

Setting mode to resWriteObjectMany, will write a new copy of the object to the resource file whether the object has already been written before or not. This is the mode to use if you are writing an object resource "template" (the normal case).

### Return Value
- stsBadParam resId is a nil resource ID.
- stsResWriteObjDynamicClass Class of object cannot be dynamic.
- stsResResourceTooBig Tried to write resource bigger than resource file can handle (16Meg).
### msgResGetObject

Reads the filed object resource from the current file position.

Takes P_OBJECT, returns STATUS.

```
#define msgResGetObject MakeMsg(clsResFile, 8)
```

**Comments**

This should only be called by routines responding to `msgRestore`. This message is provided as a convenience. It eliminates the need for everyone to duplicate the same code and guarantees that the parallel operation (`msgResPutObject`) will work.

**Return Value**

- `stsResGetNotFromRestore`: This was sent in a context other than in response to a `msgRestore`.

This message is equivalent to this pseudo code:

```
STREAM_READ_WRITE fsRead;
RES_READ_OBJECT resRead;
STATUS status;

// Read the object's resource ID from the file.
fsRead.numBytes = SizeOf(resRead.resId);
fsRead.pBuf = &resRead.resId;
ObjCallRet(msgStreamRead, pArgs->file, &fsRead, status);

// Set up the read resource object request.
resRead.mode = resReadObjectOnce;
ObjCallRet(msgNewDefaults, clsObject, &resRead.new, status);

// Read the object if one was filed.
if (resRead.resId != resNilResId) {
    ObjCallRet(msgResReadObject, pArgs->file, &resRead, status);
}
```

### msgResPutObject

Writes the object as a filed object resource to the current file position.

Takes OBJECT, returns STATUS.

```
#define msgResPutObject MakeMsg(clsResFile, 9)
```

**Comments**

This should only be called by routines responding to `msgSave`. This message is provided as a convenience. It eliminates the need for everyone to duplicate the same code and guarantees that the parallel operation (`msgResGetObject`) is done in the correct order.

**Return Value**

- `stsResPutNotFromSave`: This was sent in a context other than in response to a `msgSave`.

This message is equivalent to this pseudo code:

```
STREAM_READ_WRITE fsWrite;
RES_WRITE_OBJECT resWrite;
STATUS status;

if (object != Nil(OBJECT)) {
    // Assign an appropriate resource ID to the object.
    if (!ObjectIsDynamic(object)) {
        resWrite.resId = MakeWknObjResId(object);
    } else {
        ObjCallRet(
            msgResNextDynResId, pArgs->file, &resWrite.resId, status
        );
    }

    // Write the object.
    resWrite.mode = resWriteObjectOnce;
    resWrite.object = object;
    ObjCallRet(msgResWriteObject, pArgs->file, &resWrite, status);
```
msgResReadObjectWithFlags

Reads a resource object, passing the supplied flags.
Takes P_RES_READ_OBJECT, returns STATUS.

```c
#define msgResReadObjectWithFlags MakeMsg(clsResFile, 15)
typedef struct RES_READ_OBJECT {
    RES_READ_OBJ_MODE mode;          // Duplicate checking mode
    RES_ID resid;                    // Object passed back in new.uid
    OBJECT_NEW objectNew;            // Only for msgResReadObjectWithFlags
    RES_SAVE_RESTORE_FLAGS sysFlags; // Only for msgResReadObjectWithFlags
    U16 appFlags;                    // Only for msgResReadObjectWithFlags
    U32 spare1;
} RES_READ_OBJECT, *P_RES_READ_OBJECT;
```

This is identical to msgResReadObject except that it copies the flag values supplied into all msgRestore calls done by this or any object reads that are done recursively from this.

The values for the sysFlags field are defined by GO and should be examined by any object that needs special behavior for any of the defined cases (currently only on copy).

The values for the appFlags field are defined by an application writer. Great care must be used with setting or testing these flags. If the flags from one application are used with a class of a second application, disaster can result. E.g. set this field to 0 unless you are very sure you know what you are doing.

- `stsBadParam` resId is a nil resource ID.
- `stsResResourceNotFound` No resource with the given resId exists.
- `stsResNotObjectResource` The found resource was a data resource.
- `stsResNotFullyRead` The msgRestore routine did not read the same amount of data as the msgSave wrote.

See Also

- msgResReadObject Normal message to read an object
- msgResWriteObjectWithFlags The matching write call.

msgResWriteObjectWithFlags

Writes a resource object, passing the supplied flags.
Takes P_RES_WRITE_OBJECT, returns STATUS.

```c
#define msgResWriteObjectWithFlags MakeMsg(clsResFile, 16)
typedef struct RES_WRITE_OBJECT {
    RES_WRITE_OBJ_MODE mode;          // Duplicate checking mode
    RES_ID resid;                    // Object to write
    OBJECT object;                   // Only for msgResWriteObjectWithFlags
    RES_SAVE_RESTORE_FLAGS sysFlags; // Only for msgResWriteObjectWithFlags
    U16 appFlags;                    // Only for msgResWriteObjectWithFlags
    U32 spare1;
} RES_WRITE_OBJECT, *P_RES_WRITE_OBJECT;
```
### msgResDeleteResource

Deletes the resource identified by RES_ID.

Takes RES_ID, returns STATUS.

```c
#define msgResDeleteResource MakeMsg(clsResFile, 10)
```

**Comments**

This marks the resource deleted in the resource file index. The space taken by the resource is reclaimed whenever the resource file is compacted. Auto compaction may happen after a resource is deleted. Note that this may NOT be called during msgSave or msgRestore. It will appear to work, but the read or write will fail.

**Return Value**

- **stsBadParam**  
  resId is a nil resource ID.
- **stsResResourceNotFound**  
  No resource with the given resId exists.

### msgResCompact

Compacts the resource file.

Takes void, returns STATUS.

```c
#define msgResCompact MakeMsg(clsResFile, 11)
```

**Comments**

This message removes all deleted entries from the file and frees any unused space that results. This can be called automatically in a couple of ways. See msgNew for an explanation of them.

**Return Value**

- **stsResCompactInReadOrWrite**  
  Can not compact during read or write. This only happens if msgCompact is sent during msgSave or msgRestore.

### msgResFlush

Flushes the resource file index.

Takes void, returns STATUS.

```c
#define msgResFlush MakeMsg(clsResFile, 12)
```
The resource file keeps track of all objects that have filed themselves in the resource file. It needs this information to implement the `resReadObjectOnce / resWriteObjectOnce` behavior. If you wish to override the `resReadObjectOnce / resWriteObjectOnce` behavior, then flush the resource file.

Clients rarely use this message. Instead, use the `resReadObjectMany / resWriteObjectMany` modes with `msgResReadObject / msgResWriteObject`.

This also sends a `msgFSFlush` to the file. If all you want to do is flush the file then use `msgFSFlush` instead of `msgResFlush`.

**msgResReadObject** To get info on read once / read many.

**msgResWriteObject** To get info on write once / write many.

---

**msgResEnumResources**

Enumerates resources in a resource file or resource list.

Takes RES_ENUM, returns STATUS.

```c
#define msgResEnumResources MakeMsg(clsResFile, 13)
```

**Arguments**

```c
typedef struct RES_ENUM {
    U16             max;  // size of pResId[] and pResFile[] arrays
    U16             count; // # to pass back in arrays
    RES_ENUM_MODE   mode;  // if count > max then memory may be allocated
    UID             match; // Out: # of valid entries in arrays
    P_RES_ID        pResId; // Enumerate based on what and first/next.
    P_RES_FILE      pResFile; // key to match on (i.e. class; agent; etc)
    // Out: ptr to array of resource IDs
    // Out: ptr to array of resource file handles
    // Note: if memory was alloc'd for previous 2
    // fields, client should heap free the memory
} RES_ENUM, *P_RES_ENUM;
```

**Comments**

This message will enumerate all resources of a given category (based on mode and match) in either a single resource file or a resource list. The max and count fields behave as all other enum messages. This passes back the resource IDs and files that contain the resources in the `pResId` and `pResFile` arrays.

Mode must always have `resEnumNext` clear the first time this is called and set subsequent times. Other mode flags selectively filter what is being enumerated.

**Return Value**

`stsBadParam` `resEnumNext` was specified first time.

Here is some pseudo-code for enumerating:

```c
#define resMaxEnums 12
STATUS status;
RES_ENUM rEnum;
RES_ID enumResIds[resMaxEnums];
RES_FILE enumResFiles[resMaxEnums];

// Enumerate only objects belonging to clsString of the resources.
rEnum.max = resMaxEnums;
rEnum.count = resMaxEnums;
rEnum.mode = resEnumByObjectClass;
rEnum.match = clsString;
rEnum.pResId = enumResIds;
rEnum.pResFile = enumResFiles;
for (status = stsOK; status == stsOK; ) {
    status = ObjectCall(msgResEnumResources, resFile, &rEnum);
    for (index = 0; index < rEnum.count; index++) {
        // Process the data, etc, etc
    }
    rEnum.mode |= resEnumNext;
}
```
msgResNextDynResId
Allocates the next available dynamic resource ID.
Takes P_RES_ID, returns STATUS.
#define msgResNextDynResId MakeMsg(clsResFile, 14)

Comments
This message may be used to allocate the next dynamic resource ID available, so that the caller can write
dynamic items without using msgResPutObject. WARNING: dynamic IDs are based on a 29 bit
count, and the values are not recycled. If you run out of available counts, this will fail.

Return Value
stsResOutOfDynResIds ran out of dynamic resIDs

ResFile Agent Message

msgResAgent
Message sent by resource file to resource agent when forwarding messages.
Takes P_RES_AGENT, returns STATUS.
#define msg_RES_AGENT, returns STATUS.
#define msgResAgent MakeMsg(clsResFile, 20)

Arguments
typedef struct RES_AGENT {
    RES_FILE file;   // File containing the resource
    U32 length;      // Length of resource entry
    MESSAGE msg;     // message passed on to agent
    P_UNKNOWN pArgs; // In-Out: message specific args
    U16 sysVersion;  // Min sys version if write
    U16 spare;
    U32 spare1;
    U32 spare2;
} RES_AGENT, *P_RES_AGENT;

Comments
Messages forwarded are msgResReadData, msgResReadObject, msgResWriteData,
For reads, current file pointer will be positioned at resource entry and length of the entry will be passed
in length field. For writes, current file pointer will be positioned where write should begin.

Class ResList Messages

msgNew
Creates a resource file (search) list object.
Takes P_RES_LIST_NEW, returns STATUS. Category: class message.

typedef struct RES_LIST_NEW ONLY {
    U16 resvd1;
    U16 resvd2;
} RES_LIST_NEW ONLY, *P_RES_LIST_NEW ONLY;
#define resListNewFields
    listNewFields
    RES_LIST_NEW ONLY resList;

typedef struct RES_LIST_NEW {
    resListNewFields
    } RES_LIST_NEW, *P_RES_LIST_NEW;
clsResList adds no additional msgNew parameters to clsList. There are no messages specific to clsResList. It adds additional behavior.

**msgResXxx**
Non-destructive resource file messages.
Takes P_RES_XXX, returns STATUS.

Comments
Resource lists accept only non-destructive resource file messages (msgResReadData, msgResReadObject, msgResReadObjectWithFlags, msgResGetObject, msgResFindResource and msgResEnumResources) and forwards the message to each resource file in the list. Resource files that are null are skipped and are not considered an error. The resource list stops forwarding the message when either all resource files in the list have been exhausted or when one of them responds with a status greater than or equal to stsOK.

Sending msgResEnumResource to a resource file list is special, because it forwards the message to all resource files in the list until the list is exhausted. Thus the enumerated data is representative of the entire resource list.

Return Value
stsRequestNotSupported  Msg was not read, find or enum.
stsListEmpty  No valid resource files in the list.

See Also
stsXXX Return values from the resource file messages that are sent to the resource list.
RESUTIL.H

This file contains the API definition for the Resource Utility procedures. The functions described in this file are contained in RESFILE.LIB.

```
#ifndef RESUTIL_INCLUDED
#define RESUTIL_INCLUDED

#ifndef RESFILE_INCLUDED
#include <resfile.h>
#endif
```

Public functions

ResUtilLoadObject

Loads an object from the ProcessResList.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED ResUtilLoadObject(
    RES_ID resId, // the resource ID of the object
    P_OBJECT pObject // Out: the object
);
```

Comments

This is a short cut to using msgResReadObject to read on object in from the ProcessResList.

ResUtilLoadString

Loads a string item from the ProcessResList.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED ResUtilLoadString(
    PP_CHAR ppString, // In/Out: the pointer to the buffer/string
    P_U32 pLength, // In/Out: the length of the buffer/string
    OS_HEAP_ID heap, // Heap to allocate from.
    RES_ID resId // resId for a string
);
```

Comments

This is a short cut to using msgResReadData to read a string in from the ProcessResList.

There are two ways of supplying space to load the string into. You can specify a pointer and a length for the data passed back (heap = null, *ppString = ptr, *pLength = xx) or you can specify a valid heap from which the resource file will allocate memory for the data (heap = heap ID, *ppString = doesn't matter, pLength = null or *pLength = doesn't matter). Typically if the size of the data is already known and it is small and short lived, then the data is "allocated" on the stack. Otherwise, the data is allocated on behalf of a heap.
ResUtilLoadListString

Loads an item from a string list in the application resource list.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED ResUtilLoadListString(
    PP_CHAR ppString,  // In/Out: the pointer to the buffer/string
    P_U32 pLength,     // In/Out: the length of the buffer/string
    OS_HEAP_ID heap,    // Heap to allocate from.
    U32 listGroup,      // The list group to select from
    IX_RES_ID listResId // Indexed resId for a string
);
```

Comments

This is a short cut to using msgResReadData to read a single string form a string array that is in theProcessResList.

Works just like ResUtilLoadString, except it uses the group and indexed resource ID to construct the resource ID of a string list and the index into it.
This class defines the Settings Notebook.

There is only one instance of the Settings Notebook in the system, on the bookshelf.

The Settings Notebook is an option book. It contains a System Preferences sheet, an Installer sheet, and a Status sheet.

The Preferences sheet contains a group of Preferences cards. These update the system preferences resource file (penpoint.res).

The Installer sheet contains one card for each installation category (apps, preferences, services, etc). Each category has an underlying install manager (see instlmgr.h). A card is automatically created when a new install manager is created, and deleted when an install manager is destroyed.

The Installer sheet allows a client to display a particular card and select an item within that card. Here's example code which activates the Settings Notebook from the Bookshelf, turns it to the Installer sheet, displays a particular card, selects an item within that card, and finally opens the Settings Notebook:

```c
#include <auxnbmgr.h>
#include <instlsht.h>

ANM_OPEN_NOTEBOOK openNotebook;
APP_METRICS am;
IUI_SELECT_ITEM selectItem;
OPTION_CARD oc;
IUI_SHOW_CARD showCard;
STATUS s;

ObjectCall(msgBusySetState, theBusyManager, (P_ARGS) true);
openNotebook.notebook = anmSettings;
openNotebook.activateOnly = true;
ObjCallRet(msgANMOpenNotebook, theAuxNotebookMgr, &openNotebook, s);
ObjSendUpdateRet(msgAppGetMetrics, openNotebook.uid, &am, SizeOf(am), s);
oc.tag = tagUIInstallerSheet;
ObjSendUpdateRet(msgOptionShowCard, am.mainWin, &oc, SizeOf(oc), s);
ObjSendUpdateRet(msgOptionGetTopCard, am.mainWin, &oc, SizeOf(oc), s);
strcpy(showCard.pCardName, "Applications");
ObjSendRet(msgIUIShowCard, oc.win, &showCard, SizeOf(showCard), s);
strcpy(selectItem.pItemName, appMgrMetrics.name);
ObjSendRet(msgIUISelectItem, oc.win, &selectItem, SizeOf(selectItem), s);
openNotebook.notebook = anmSettings;
openNotebook.activateOnly = false;
ObjCallRet(msgANMOpenNotebook, theAuxNotebookMgr, &openNotebook, s);
ObjectCall(msgBusySetState, theBusyManager, (P_ARGS) false);
```

ifndef SETTINGS_INCLUDED
#define SETTINGS_INCLUDED
ifndef APPTAG_INCLUDED
#include <apptag.h>
#endif
endif
Common #defines and typedefs

```c
#define tagSettingsPrefSheet MakeTag(clsInstallUISheet, 29)
#define tagSettingsInstallerSheet MakeTag(clsInstallUISheet, 30)
#define tagSettingsStatusSheet MakeTag(clsInstallUISheet, 31)
#define tagSettingsNBPeripheralsOnIconResId tagAppIconBitmap
#define tagSettingsNBPeripheralsOnSmallIconResId tagAppSmallIconBitmap
#define tagSettingsNBPeripheralsOffSmallIconResId \ 
  MakeTag(clsSettingsNBAppWin, 1)
#define tagSettingsNBPeripheralsOffIconResId \ 
  MakeTag(clsSettingsNBAppWin, 2)
#define tagSettingsPrefCmdBar MakeTag(clsSettingsNB, 100)
```

Error status codes

```c
#define stsSettingsValueOutOfRange MakeStatus(clsSettingsNB, 1)
#define stsSettingsFixedValueOutOfRange MakeStatus(clsSettingsNB, 2)
```
Part 12 / Installation API
This file contains the API definition for clsAppInstallMgr.

clsAppInstallMgr inherits from clsCodelInstallMgr.

Manages installation and deinstallation of applications.

There is a single instance of clsAppInstallMgr in the system; the well-known uid thelnstalledApps.

thelnstalledApps performs installation and deinstallation of applications and allows you to enumerate all of the applications that are currently installed.

An application is a directory, usually located under \penpoint\app on a given filesystem volume. The name of the directory is the name of the application. Within this directory is a .exe and zero or more .dlls that make up the application. If a application includes .dlls there must also be a .dlc file which lists all the .dlls and the .exe. The name of the .dlc file (or the name of the .exe file if there are no .dlls) must be the same as the name of the application. If a application is called MAIL, for example, its .dlc file must be named MAIL.DLC. You can use the STAMPREXE utility to give an application an extended name. Be sure to stamp the .dlc file as well.

There can also be a service.ini and app.ini file in the application's directory. These specify any additional services and applications that should be installed when this application is installed. These services and applications are deinstalled when the application is deinstalled. If one of these services or applications is already installed it is reference counted, not installed again.

This directory also contains subdirectories which hold entries in the Help notebook (HELP), stationery (STATNRy), tools (ACCESSRy), and any app-specific files that should be copied in when the app is installed (MISC). The application's resource file, app.res, is also in this directory.

The application monitor is responsible for managing the installation of these items. When an app is installed its code is loaded and app.res is copied in. The application monitor object is then created and completes the installation. You can subclass the application monitor if you need control over the installation process. See appmon.h for details.

An application is installed by sending msgIMInstall to thelnstalledApps. Applications are installed under user control from the Applications card of the Settings Notebook. \boot\penpoint\boot\app.ini specifies applications that are automatically loaded when the system cold-boots.

Each installed application has an application directory in the RAM filesystem under \penpoint\sys\app. For example, MAIL be in \penpoint\sys\app\MAIL. The application resource file and the MISC directory are copied to this directory.

Each installed application is represented by a handle, in a fashion similar to other install managers (see instlmgr.h). This handle is a directory handle onto the application's directory in the RAM filesystem.

NOTE: THE MESSAGES IN THIS CLASS ARE SENT TO THE MANAGER, NOT TO THE HANDLES.

An application can be deinstalled. Deinstallation removes all traces of an application.

An application can be deinstalled even if there are running or filed instances of that application in the machine. All running instances are shut down (saved, then terminated) when an application is removed.
The application framework will use the Placeholder (MaskApp) class if it tries to start up document with a missing application.

The following superclass messages are not understood by clsAppInstallMgr:

- msgIMGetCurrent
- msgIMSetCurrent
- msgIMSetName
- msgIMDup

The following notification messages are not sent by clsAppInstallMgr:

- msgIMNameChanged
- msgIMCurrentChanged

See Also

instlmgr.h
appmon.h

```c
#ifndef APPIMGR_INCLUDED
#define APPIMGR_INCLUDED

#ifndef CODEMGR_INCLUDED
#include <codemgr.h>
#endif
#endif
```

## Common #defines and typedefs

### msgNew

Creates a new application installation manager.

Takes P_AIM_NEW, returns STATUS. Category: class message.

**Arguments**

typedef struct AIM_NEW {
    installMgrNewFields
} AIM_NEW, *P_AIM_NEW;

**Comments**

There is only one instance of this class, theInstalledApps, in the system. Clients should never send msgNew.

### msgAIMGetMaskClass

Passes back the mask class.

Takes P_CLASS, returns STATUS.

```c
#define msgAIMGetMaskClass MakeMsg(clsAppInstallMgr, 6)
```

**Comments**

The mask application class is used by the application framework when it tries to start up a document with an unavailable application.

### msgAIMSetMaskClass

Sets the mask class.

Takes CLASS, returns STATUS.

```c
#define msgAIMSetMaskClass MakeMsg(clsAppInstallMgr, 7)
```
The mask application class is used by the application framework when it tries to start up a document with an unavailable application.

This message can be sent at any time; however, the new mask class will only be used for subsequent switches.
This file contains the class definition and methods for clsAuxNotebookMgr. clsAuxNotebookMgr inherits from clsObject. Manages the system notebooks and documents on the bookshelf.

There is a single instance of clsAuxNotebookMgr in the system; the well-known uid theAuxNotebookMgr. The auxiliary notebook manager creates the following items on the bookshelf:
The Help NotebookSettings NotebookAccessories PalletteStationery NotebookKeyboard InstanceConnections Notebook InstanceInbox Notebook Outbox Notebook

It provides access to those items that are guaranteed to always be on the bookshelf:
The Help NotebookSettings NotebookAccessories PalletteStationery NotebookInbox Notebook Outbox Notebook

It allows documents and sections to be created in the Notebooks it manages, and copies documents into the Notebooks. It also provides several Stationery-specific functions.

theAuxNotebookMgr is usually not used by applications, other than to activate and open one of system items on the bookshelf.

The document/section creation and copy facilities are used by application installation.

ifndef AUXNBGR_INCLUDED
#define AUXNBGR_INCLUDED
ifndef GEO_INCLUDED
#include <geo.h>
#endif
ifndef FS_INCLUDED
#include <fs.h>
#endif

Common #defines and typedefs

Which bookshelf item? Used in most messages to theAuxNotebookMgr. Also used as part of the definition of the well-known uids for these items.

typedef enum AM_AUX_NOTEBOOK {
    amnReserved = 0,  // Never use this value! See amnAttrWhichAuxNB below.
    amnSettingsNotebook = 1,  // Settings Notebook.
    amnHelpNotebook = 3,  // Help Notebook.
    amnStationeryNotebook = 4,  // Stationery Notebook.
    amnInboxNotebook = 5,  // Inbox.
    amnOutboxNotebook = 6,  // Outbox.
    amnAccessories = 7,  // Accessories Pallette.
} AM_AUX_NOTEBOOK, *P_AM_AUX_NOTEBOOK;

Exist behavior for creating sections and docs.
typedef enum ANM_EXIST_BEHAVIOR {
    anmExistGenError,
    anmExistDoNothing,
    anmExistTruncate,
    anmExistGenUnique
} ANM_EXIST_BEHAVIOR, *P_ANM_EXIST_BEHAVIOR;

Should a section and/or a notebook entry be added to the stationery menu?

typedef struct STAT_MENU_STYLE {
    U16 section : 2, // Add a section entry.
    notebook : 2, // Add a notebook entry.
    unused1 : 12; // reserved
} STAT_MENU_STYLE, *P_STAT_MENU_STYLE;

Filesystem Attributes

Should a given piece of stationery be on the stationery menu?

#define anmAttrStationeryMenu FSMakeFix32Attr(clsAuxNotebookMgr, 1)
typedef enum ANM_ATTR_STATIONERY_MENU {
    anmNotOnMenu = 0, // Same as no attribute.
    anmOnMenu = 1
} ANM_ATTR_STATIONERY_MENU;

Should a stationery or tool document not be loaded at install time? This attribute is on the the document on the external filesystem.

#define anmAttrNoLoad FSMakeFix32Attr(clsAuxNotebookMgr, 2)
typedef enum ANM_ATTR_NO_LOAD {
    anmLoad = 0, // Same as no attribute.
    anmNoLoad = 1
} ANM_ATTR_NO_LOAD;

Id tag; used to designate stationery or accessory documents.

#define anmAttrId FSMakeFix32Attr(clsAuxNotebookMgr, 3)
Attribute used to tell the difference between an auxiliary notebooks and a data notebooks. Backup programs take note. Never backup an auxiliary notebook!

#define anmAttrAuxNB FSMakeFix32Attr(clsAuxNotebookMgr, 4)
typedef enum ANM_ATTR_AUX_NB {
    anmDataNB = 0, // Same as no attribute.
    anmAuxNB = 1
} ANM_ATTR_AUX_NB;

Attribute used by clsNBToC to perform special behavior for each auxnb. This attribute is stamped on the auxnb's TOC at initialization time. The attribute values are specified in the ANM_AUX_NOTEBOOK enum. Note: ANM_AUX_NOTEBOOK must never have a 0 value; 0 indicates no anmAttrWhichAuxNB attribute.

#define anmAttrWhichAuxNB FSMakeFix32Attr(clsAuxNotebookMgr, 5)
Used to get auto-expand behavior of stationery sections.

#define anmAttrExpandStationerySection FSMakeFix32Attr(clsAuxNotebookMgr, 6)
**Messages**

`msgNew`
Creates a new auxiliary notebook manager.

Takes `P_ANM_NEW`, returns `STATUS`. Category: class message.

```c
#define auxNotebookMgrNewFields
  \ 
  objectNewFields
```

**Arguments**

typedef struct ANM_NEW {
  auxNotebookMgrNewFields
} ANM_NEW, *P_ANM_NEW;

**Comments**

Note: this is done once and only once in the init routine of this dll to create theAuxNotebookMgr. This message must never be called by anyone else!

`msgANMCreateDoc`
Create a document in one of the auxiliary notebooks.

Takes `P_ANM_CREATE_DOC`, returns `STATUS`.

```c
#define msgANMCreateDoc MakeMsg(clsAuxNotebookMgr, 1)
```

**Arguments**

typedef struct ANM_CREATE_DOC {
  ANM_AUX_NOTEBOOK notebook; // Which auxiliary notebook?
  CLASS docClass; // Document class.
  P_STRING pPath; // Path to create doc in, relative to
                  // base of the aux notebook. pNull
                  // says to create at top level.
  P_STRING pName; // Name of doc.
  U32 sequence; // Sequence number to create in front of.
  P_STRING pBookmarkLabel; // pNull for no bookmark.
  ANM_EXIST_BEHAVIOR exist; // What to do if the doc exists/doesn't
                             // exist. Note: doc might exist due to
                             // warm boot.
  BOOLEAN putInMenu; // If type is stationery, should the doc
                     // initially be in the stationery menu?
  P_FS_FLAT_LOCATOR pDestPath; // Out: Location of created doc.
                                // if pDestPath is pNull then nothing is
                                // returned.
  U32 id; // Id to tag everything with. 0 is no tag.
} ANM_CREATE_DOC, *P_ANM_CREATE_DOC;

`msgANMCreateSect`
Create a section in one of the auxiliary notebooks.

Takes `P_ANM_CREATE_SECT`, returns `STATUS`.

```c
#define msgANMCreateSect MakeMsg(clsAuxNotebookMgr, 2)
```

**Arguments**

typedef struct ANM_CREATE_SECT {
  ANM_AUX_NOTEBOOK notebook; // Which auxiliary notebook?
  CLASS sectClass; // Section class.
  P_STRING pPath; // Path to create section in, relative to
                   // base of the aux notebook. pNull
                   // says to create at top level.
  P_STRING pName; // Name of section.
  U32 sequence; // Sequence number to create in front of.
  P_STRING pBookmarkLabel; // pNull for no bookmark.
  ANM_EXIST_BEHAVIOR exist; // What to do if the sect exists/doesn’t
                     // exist. Note: sect might exist due to
msgANMMoveInDoc

Move a document into an auxiliary notebook.

Takes P_ANM_MOVE_COPY_DOC, returns STATUS.

#define msgANMMoveInDoc MakeMsg(clsAuxNotebookMgr, 3)

typedef struct ANM_MOVE_COPY_DOC {
  ANM_AUX_NOTEBOOK notebook; // Which auxiliary notebook?
  FS LOCATOR source; // Source document.
  P_STRING pPath; // Path to move/copy doc to, relative to
                  // base of the aux notebook. pNull
                  // says to create at top level.
  CLASS defaultClass; // Class to use if source isn’t stamped.
  U32 sequence; // Sequence number to move/copy in front
                  // of.
  P_STRING pBookmarkLabel; // pNull for no bookmark.
  ANM_EXIST_BEHAVIOR exist; // What to do if the doc exists/doesn’t
                             // exist. Note: doc might exist due to
                             // warm boot.
  BOOLEAN forceInMenu; // If this is stationery, override
                       // any local attribute and put it in
                       // the stationery menu.
  P_FS_FLAT_LOCATOR pDestPath; // Out: Location of created section.
                              // if pDestPath is pNull then nothing is
                              // returned.
  U32 id; // Id to tag everything with. 0 is no tag.
} ANM_CREATE_SECT, *P_ANM_CREATE_SECT;

msgANMCopyInDoc

Copy a document into an auxiliary notebook.

Takes P_ANM_MOVE_COPY_DOC, returns STATUS.

#define msgANMCopyInDoc MakeMsg(clsAuxNotebookMgr, 4)

typedef struct ANM_MOVE_COPY_DOC {
  ANM_AUX_NOTEBOOK notebook; // Which auxiliary notebook?
  FS LOCATOR source; // Source document.
  P_STRING pPath; // Path to move/copy doc to, relative to
                  // base of the aux notebook. pNull
                  // says to create at top level.
  CLASS defaultClass; // Class to use if source isn’t stamped.
  U32 sequence; // Sequence number to move/copy in front
                  // of.
  P_STRING pBookmarkLabel; // pNull for no bookmark.
  ANM_EXIST_BEHAVIOR exist; // What to do if the doc exists/doesn’t
                             // exist. Note: doc might exist due to
                             // warm boot.
  BOOLEAN forceInMenu; // If this is stationery, override
                       // any local attribute and put it in
                       // the stationery menu.
  P_FS_FLAT_LOCATOR pDestPath; // Out: Location of destination doc.
                              // if pDestPath is pNull then nothing is
                              // returned.
  U32 id; // Id to tag everything with. 0 is no tag.
} ANM_MOVE_COPY_DOC, *P_ANM_MOVE_COPY_DOC;
**msgANMDelete**

Delete a section or document in one of the auxiliary notebooks.

Takes \texttt{P\_ANM\_DELETE}, returns \texttt{STATUS}.

```c
#define msgANMDelete MakeMsg(clsAuxNotebookMgr, 7)
```

**Arguments**

typedef struct ANM\_DELETE {
    ANM\_AUX\_NOTEBOOK notebook; // Which auxiliary notebook?
    P\_STRING pPath; // Path of item to delete.
} ANM\_DELETE, *P\_ANM\_DELETE;

**msgANMDeleteAll**

Delete all the nodes that are identified by 'id'.

Takes \texttt{P\_ANM\_DELETE\_ALL}, returns \texttt{STATUS}.

```c
#define msgANMDeleteAll MakeMsg(clsAuxNotebookMgr, 8)
```

**Arguments**

typedef struct ANM\_DELETE\_ALL {
    ANM\_AUX\_NOTEBOOK notebook; // Which auxiliary notebook?
    U32 id; // Id.
} ANM\_DELETE\_ALL, *P\_ANM\_DELETE\_ALL;

**Comments**

If a node's id attribute or its app class is 'id' then delete it.

**msgANMGetNotebookPath**

Returns the base path of one of the auxiliary notebooks.

Takes \texttt{P\_ANM\_GET\_NOTEBOOK\_PATH}, returns \texttt{STATUS}.

```c
#define msgANMGetNotebookPath MakeMsg(clsAuxNotebookMgr, 9)
```

**Arguments**

typedef struct ANM\_GET\_NOTEBOOK\_PATH {
    ANM\_AUX\_NOTEBOOK notebook; // Which auxiliary notebook?
    P\_FS\_FLAT\_LOCATOR pLocator; // Out: base location of notebook.
} ANM\_GET\_NOTEBOOK\_PATH, *P\_ANM\_GET\_NOTEBOOK\_PATH;

**Comments**

Note: This will return a path to the table of contents of the notebook. See \texttt{msgANMGetNotebookUUID} if you want the actual notebook itself.

**msgANMGetNotebookUUID**

Returns the uuid of one of the auxiliary notebooks.

Takes \texttt{P\_ANM\_GET\_NOTEBOOK\_UUID}, returns \texttt{STATUS}.

```c
#define msgANMGetNotebookUUID MakeMsg(clsAuxNotebookMgr, 10)
```

**Arguments**

typedef struct ANM\_GET\_NOTEBOOK\_UUID {
    ANM\_AUX\_NOTEBOOK notebook; // Which auxiliary notebook?
    UUID uuid; // Out: uuid of auxiliary notebook.
} ANM\_GET\_NOTEBOOK\_UUID, *P\_ANM\_GET\_NOTEBOOK\_UUID;

**Comments**

Note: This is the UUID of the actual notebook. Use \texttt{msgANMGetNotebookPath} to get to the table of contents of the notebook.
msgANMOpenNotebook
Activate and optionally open an auxiliary notebook.
Takes P_ANM_OPEN_NOTEBOOK, returns STATUS.

```c
#define msgANMOpenNotebook MakeMsg(clsAuxNotebookMgr, 11)
```

**Arguments**

typedef struct ANM_OPEN_NOTEBOOK {
    ANM_AUX_NOTEBOOK notebook;  // Which notebook.
    BOOLEAN activateOnly;       // Only activate; don’t open
    OBJECT uid;                 // Out: uid of activated or
                                 // opened auxnb.
} ANM_OPEN_NOTEBOOK, *P_ANM_OPEN_NOTEBOOK;

**Private**

msgANMPopUpStationeryMenu
Pop up the stationery menu at the specified location.
Takes P_ANM_POP_UP_MENU, returns STATUS.

```c
#define msgANMPopUpStationeryMenu MakeMsg(clsAuxNotebookMgr, 5)
```

**Arguments**

typedef struct ANM_POP_UP_MENU {
    XY32 hotSpot;          // Where to pop up menu. Coords are
                            // relative to destObj.
    OBJECT destObj;       // Object to create stationery in front
                            // of.
    STAT_MENU_STYLE style; // Menu style.
} ANM_POP_UP_MENU, *P_ANM_POP_UP_MENU;

**Comments**
If the user hits one of the menu items create a stationery document in the destination object at the hotSpot.

msgANMGetStationeryMenu
Passes back the stationery menu.
Takes P_ANM_GET_MENU, returns STATUS.

```c
#define msgANMGetStationeryMenu MakeMsg(clsAuxNotebookMgr, 6)
```

**Arguments**

typedef struct ANM_GET_MENU {
    XY32 hotSpot;          // Where to pop up menu. Coords are
                            // relative to destObj.
    OBJECT destObj;       // Object to create stationery in front
                            // of.
    STAT_MENU_STYLE style; // Menu style.
    OBJECT menu;          // Out: Stationery menu.
} ANM_GET_MENU, *P_ANM_GET_MENU;

**Comments**
This message allows the app framework to add the stationery menu to an existing menu bar. When the stationery menu is invoked, stationery is created in destObj at the hotSpot.

msgANMAddToStationeryMenu
Add a stationery notebook doc to the stationery menu.
Takes P_ANM_MENU_ADD_REMOVE, returns STATUS.

```c
#define msgANMAddToStationeryMenu MakeMsg(clsAuxNotebookMgr, 12)
```
typedef struct ANM_MENU_ADD_REMOVE {
    UUID document; // Dir Index of document to remove.
} ANM_MENU_ADD_REMOVE, *P_ANM_MENU_ADD_REMOVE;

msgANMRemoveFromStationeryMenu
Remove a document from the stationery menu.
Takes P_ANM_MENU_ADD_REMOVE, returns STATUS.
#define msgANMRemoveFromStationeryMenu MakeMsg(clsAuxNotebookMgr, 13)

typedef struct ANM_MENU_ADD_REMOVE {
    UUID document; // Dir Index of document to remove.
} ANM_MENU_ADD_REMOVE, *P_ANM_MENU_ADD_REMOVE;

msgANMStationeryMenuNameChanged
Informs the stationery menu that one of its documents has changed name.
Takes P_ANM_MENU_NAME_CHANGED, returns STATUS.
#define msgANMStationeryMenuNameChanged MakeMsg(clsAuxNotebookMgr, 17)

typedef struct ANM_MENU_NAME_CHANGED {
    UUID document; // Dir Index of document whose name
                   // changed.
} ANM_MENU_NAME_CHANGED, *P_ANM_MENU_NAME_CHANGED;

Obsolete
#define anmAttrPermanent FSMakeFix32Attr(clsAuxNotebookMgr, 0)
typedef enum ANM_ATTR_PERMANENT {
    anmNotPermanent = 0, // Same as no attribute.
    anm Permanent = 1
} ANM_ATTR_PERMANENT;
// Next available message number: 18
CODEMGR.H

This file contains the API definition for clsCodeInstallMgr.
clsCodeInstallMgr inherits from clsInstallMgr.
Manages installation and deinstallation of code: applications and services.
clsAppInstallMgr and clsServiceInstallMgr inherit from this class.
The following superclass messages are not understood by clsCodeInstallMgr:
• msgIMGetCurrent
• msgIMSetCurrent
• msgIMSetName
• msgIMDup
The following notification messages are not sent by clsCodeInstallMgr:
• msgIMNameChanged
• msgIMCurrentChanged

instlmgr.h
#ifndef CODEMGR_INCLUDED
#define CODEMGR_INCLUDED
#ifndef INSTLMGR_INCLUDED
#include <instlmgr.h>
#endif

▶ Common #defines and typedefs

▶ Status Codes
An application or service's name can be a max of nameBufLength - 4 chars.
#define stsCIMNameTooLong MakeStatus(clsCodeInstallMgr, 0)

▶ Filesystem Attribute Definitions
Note: Most clients do not deal with attributes directly.
Application or service class
#define cimAttrClass FSMakeFix32Attr(clsCodeInstallMgr, 0)
Application or service program handle
#define cimAttrProgHandle FSMakeFix32Attr(clsCodeInstallMgr, 1)
Application or service program well-known name
#define cimAttrProgramName FSMakeStrAttr(clsCodeInstallMgr, 2)
Should this app or service be seen in the installer? This determines whether the user can configure and
deinstall it.

```c
#define cimAttrDeinstallable FSMMakeFix32Attr(clsCodeInstallMgr, 4)
typedef enum CIM_ATTR_DEINSTALLABLE {
    cimDeinstallable = 0, // Same as no attribute
    cimNotDeinstallable = 1
} CIM_ATTR_DEINSTALLABLE;
```

Dependent application list

```c
#define cimAttrAppList FSMMakeVarAttr(clsCodeInstallMgr, 6)
```

Dependent services list

```c
#define cimAttrServiceList FSMMakeVarAttr(clsCodeInstallMgr, 7)
```

Common data structure used by msgCIMTerminateVetoed and msgCIMGetTerminateStatus.

```c
typedef struct CIM_TERMINATE_VETOED {
    IM_HANDLE handle;
    OBJECT vetoer; // Object that vetoed the terminate.
    STATUS status; // Veto status.
} CIM_TERMINATE_VETOED, *P_CIM_TERMINATE_VETOED;
```

### Messages

**msgCIMGetClassList**

Passes back a list of the classes of the installed applications or services.

Takes P_LIST, returns STATUS.

```c
#define msgCIMGetClassList MakeMsg(clsCodeInstallMgr, 1)
```

**msgCIMGetClass**

Given a handle, passes back the class.

Takes P_CIM_GET_CLASS, returns STATUS.

```c
#define msgCIMGetClass MakeMsg(clsCodeInstallMgr, 2)
```

**msgCIMFindClass**

Returns the handle which references the specified class.

Takes P_CIM_FIND_CLASS, returns STATUS.

```c
#define msgCIMFindClass MakeMsg(clsCodeInstallMgr, 3)
```

**Return Value**

```
stsNoMatch No handle for this class was found.
```
**msg CIMFindProgram**
Finds a item's handle, given its program name.
Takes P_CIM_FIND_PROGRAM, returns STATUS.

```c
#define msgCIMFindProgram MakeMsg(clsCodeInstallMgr, 22)
```

**Arguments**
- `typedef struct CIM_FIND_PROGRAM {
  P_STRING pName; // Program name to search for
  IM_HANDLE handle; // Out: Resulting handle
} CIM_FIND_PROGRAM, *P_CIM_FIND_PROGRAM;
```

**Return Value**
- `stsNoMatch` Item not found.

**msg CIMLoad**
Installs code for the item specified.
Takes P_CIM_LOAD, returns STATUS. Category: descendant responsibility.

```c
#define msgCIMLoad MakeMsg(clsCodeInstallMgr, 4)
```

**Arguments**
- `typedef struct CIM_LOAD {
  IM_HANDLE handle; // Handle of item to load.
} CIM_LOAD, *P_CIM_LOAD;
```

**Comments**
This message is sent to subclasses to do the actual work of installing the item. The working directory is set to the source. `pArgs->handle` references the deactivated item to load.

**msg CIMTerminateOK**
Is this item willing to be terminated?
Takes P_CIM_TERMINATE_OK, returns STATUS. Category: descendant responsibility.

```c
#define msgCIMTerminateOK MakeMsg(clsCodeInstallMgr, 5)
```

**Arguments**
- `typedef struct CIM_TERMINATE_OK {
  IM_HANDLE handle; // Item to ask.
  OBJECT vetoer; // Out: Object which vetoed the terminate.
} CIM_TERMINATE_OK, *P_CIM_TERMINATE_OK;
```

**msg CIMTerminate**
Unconditionally terminate this item.
Takes P_CIM_TERMINATE, returns STATUS. Category: descendant responsibility.

```c
#define msgCIMTerminate MakeMsg(clsCodeInstallMgr, 6)
```

**Arguments**
- `typedef struct CIM_TERMINATE {
  IM_HANDLE handle;
} CIM_TERMINATE, *P_CIM_TERMINATE;
```

**msg CIMTerminateVetoed**
Somebody vetoed the termination sequence.
Takes P_CIM_TERMINATE, returns STATUS. Category: descendant responsibility.

```c
#define msgCIMTerminateVetoed MakeMsg(clsCodeInstallMgr, 7)
```

**Arguments**
- `typedef struct CIM_TERMINATE {
  IM_HANDLE handle;
} CIM_TERMINATE, *P_CIM_TERMINATE;
```
msgCIMGetTerminateStatus

Gets termination status of last item deinstalled.

Takes P_CIM_TERMINATE_VETOED, returns STATUS.

#define msgCIMGetTerminateStatus MakeMsg(clsCodeInstallMgr, 8)

typedef struct CIM_TERMINATE_VETOED {
   IM_HANDLE handle;
   OBJECT vetoer; // Object that vetoed the terminate.
   STATUS status; // Veto status.
} CIM_TERMINATE_VETOED, *P_CIM_TERMINATE_VETOED;

If there was an error then pArgs->vetoer is the object which caused the error; an application instance in
the case of applications and a service instance in the case of services. pArgs->status is the termination
status.
DYNTABLE.H

This file contains the API definition for clsDynamicTableMgr.

clsDynamicTableMgr inherits from clsObject.

It allows a tk table to track the comings and goings of installable items.

Overview

tkTables (see tktable.h) are typically used to display static tables. However, there are times when clients wish to build a tkTable that views a dynamic structure, such as the installed fonts or the currently connected filesystem volumes. clsDynamicTableMgr allows a tk table to be dynamically updated as one of these things changes. Specifically, clsDynamicTableMgr supports viewing the contents of an install manager (see instlmgr.h) and filesystem volumes (see fs.h).

When the dynamic table manager is first created it generates a tkTable entry for each item in the dynamic structure. The label of the tkTable entry is set to the name of the item. The tkTable entry is tagged with the uid of the Install Manager handle or the uid of a volume's root directory handle.

If the specified Install Manager is the InstalledFonts and the entry class inherits from clsButton then the short font id is also stored in the entry's data field.

clsDynamicTKTableMgr also supports an optional write-in field that is added to the end of the tk table.

#ifndef DYNTABLE_INCLUDED
#define DYNTABLE_INCLUDED
#endif

#ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif

#ifndef FONT_INSTALL_INCLUDED
#include <fontmgr.h>
#endif

Common #defines and typedefs

Object property tag for entries managed by this class.

#define propDTEntry MakeTag(clsDynamicTableMgr, 1)

Tag on the fill-in field button, if style.addFilllnField is true.

#define tagDTFillInField MakeTag(clsDynamicTableMgr, 2)

Activated/Deactivated display styles

#define dtNoShowDeactivated 0 // Don't show any deactivated items.
#define dtShowDeactivated 1 // Show deactivated items same as normal items.
#define dtShowDeactivatedAsInactive 2 // Show deactivated with bsLookInactive.

typedef struct DYN_TABLE_STYLE {
    U16 showDeactivated : 2, // How to deal with deactivated elements.
    autoDestroy : 1, // Destroy self when tkTable is freed.
    ignoreRamVolume : 1, // Don't show the RAM filesystem volume.
    putFontIdInData : 1, // Put short font id in entry's data field.
    addFillInField : 1, // Add a blank write-in field. This is a text field inside of a button.
    unused : 10;
    U16 spare1;
} DYN_TABLE_STYLE, *P_DYN_TABLE_STYLE;
typedef struct DYN_TABLE_NEW_ONLY {
    DYN_TABLE_STYLE style;
    OBJECT installMgr; // Install Mgr, ie. theInstalledFonts.
                         // can also be theFileSystem.
    OBJECT tkTable;    // Table to manage. Must be updated
                         // after msgRestore via
                         // msgDynTableSetTable.
    CLASS entryClass;  // Class of tktable entries.
    P_UNKNOWN pNewArgs; // msgNewDefaulted newArgs for
                         // entryClass.
    SIZEOF newArgsSize; // Size of newArgs.
    FIM_PRUNE_CONTROL pruneControl; // Prune control if theInstalledFonts.
    US _spare[24];
} DYN_TABLE_NEW_ONLY, *P_DYN_TABLE_NEW_ONLY;
#define dynTableNewFields
    objectNewFields
        DYN_TABLE_NEW_ONLY dynTable;
} DYN_TABLE_NEW, *P_DYN_TABLE_NEW;

Messages

msgNew
Creates a new dynamic table manager.
Takes P_DYN_TABLE_NEW, returns STATUS. Category: class message.

**Arguments**

typedef struct DYN_TABLE_NEW {
    dynTableNewFields
} DYN_TABLE_NEW, *P_DYN_TABLE_NEW;

msgNewDefaults
Initializes the DYN_TABLE_NEW structure to default values.
Takes P_DYN_TABLE_NEW, returns STATUS. Category: class message.

**Arguments**

typedef struct DYN_TABLE_NEW {
    dynTableNewFields
} DYN_TABLE_NEW, *P_DYN_TABLE_NEW;

**Comments**
Sets

dynTable.style.showDeactivated = noShowDeactivated;
dynTable.style.autoDestroy = true;
dynTable.style.ignoreRamVolume = true;
dynTable.style.putFontIdInData = true;
dynTable.style.addFillInField = false;

msgDynTableGetTable
Gets the tkTable we are associated with.
Takes P_OBJECT, returns STATUS.

#define msgDynTableGetTable
    MakeMsg(clsDynamicTableMgr, 1)
**msgDynTableSetTable**

Sets our tkTable.

Takes OBJECT, returns STATUS.

`#define msgDynTableSetTable MakeMsg(clsDynamicTableMgr, 2)`

Comments

This must be done whenever this object is restored. It is the client's responsibility to relink the tkTable with the dynamic table manager.

**msgDynTableFindButton**

Finds a button in the table which has the specified label.

Takes P_DYN_TABLE_FIND_BUTTON, returns STATUS.

`#define msgDynTableFindButton MakeMsg(clsDynamicTableMgr, 3)`

Arguments

typedef struct DYN_TABLE_FIND_BUTTON {
  P_STRING pName; // Label name of field to find.
  OBJECT button; // Out: Found button.
} DYN_TABLE_FIND_BUTTON, *P_DYN_TABLE_FIND_BUTTON;

Return Value

stsNoMatch Label not found.

**msgDynTableSetFillInField**

Sets the fill-in field to a text string.

Takes P_STRING, returns STATUS.

`#define msgDynTableSetFillInField MakeMsg(clsDynamicTableMgr, 4)`

Return Value

stsBadParam There is no fill-in field in the table.
FONTMGR.H

This file contains the API definition for clsFontInstallMgr.

clsFontInstallMgr inherits from clsInstallMgr.

It performs font installation and maintenance.

There is a single instance of clsFontInstallMgr in the system; the well-known uid theInstalledFonts.

The font manager maintains the installed and deinstalled fonts on the system. The font manager differs from a generic install manager in the area of font identification and the system font.

A font is a structured file. The system comes with several pre-defined font files that are loaded at cold boot time.

Font files typically reside in the \penpoint\font directory on a given filesystem volume. This is not a requirement, however.

Fonts are identified in four ways:

• a font file handle
• a short font ID
• a string font ID
• the name of a font file

Font file handles are open file handles on to the font files. Much of the install manager interface uses these handles. A short font ID is a pre-defined, 16 bit value that identifies a specific font. It is a compact, specific reference for a particular font. The window system API uses short font IDs. A string font ID is a 4 character string version of a short font ID. The font file name is the user-visible name for the font. Given a handle, you can get the font file name by sending msgIMGetName. Given a short font ID, you can get the font file name by sending msgFIMGetNameFromId.

NOTE: THE MESSAGES IN THIS CLASS ARE SENT TO THE MANAGER, NOT TO THE HANDLES.

A list of all the font handles in the system is available via superclass message msgIMGetList. A pruned list of the fonts that is appropriate for end-user display is available via msgFIMGetInstalledIDList.

The following messages are not understood by clsFontInstallMgr:

• msgIMGetCurrent
• msgIMSetCurrent
• msgIMDup

The following notification messages are not sent by clsFontInstallMgr:

• msgIMCurrentChanged

See Also

instlmgr.h

 ifndef FONTMGR_INCLUDED
 define FONTMGR_INCLUDED
 ifndef INSTLMGR_INCLUDED
 define INSTLMGR_INCLUDED
 include <instlmgr.h>
 endif
Common #defines and typedefs

Filesystem attribute definitions

Note: Most clients do not deal with attributes directly.

Font ID

```c
#define fimAttrId

Font ID definitions

typedef U16 FIM_SHORT_ID, *P_FIM_SHORT_ID;
typedef struct FIM_LONG_ID {
    U8 pId[5];
} FIM_LONG_ID, *P_FIM_LONG_ID;

FIM_GET_SET_ID is used by msgFIMGetId and msgFIMSetId.

typedef struct FIM_GET_SET_ID {
    IM_HANDLE handle; // Font handle to get IDs on.
    FIM_SHORT_ID id;   // Out: short version of ID.
    FIM_LONG_ID longId; // Out: long ID version.
} FIM_GET_SET_ID, *P_FIM_GET_SET_ID;
```

Messages

**msgNew**

Creates a new font install manager.

Takes P_FIM_NEW, returns STATUS. Category: class message.

**Arguments**

typedef struct FIM_NEW {
    installMgrNewFields
} FIM_NEW, *P_FIM_NEW;

**Comments**

There is only one instance of this class, the InstalledFonts, in the system. Clients should never send msgNew.

**msgNewDefaults**

Initializes the FIM_NEW structure to default values.

Takes P_FIM_NEW, returns STATUS. Category: class message.

**Message**

typedef struct FIM_NEW {
    installMgrNewFields
} FIM_NEW, *P_FIM_NEW;

**Comments**

Sets

installMgr.fileMode |= fsReadOnly | fsSystemFile;

**msgFIMGetId**

Gets the short and long font IDs, given a handle.

Takes P_FIM_GET_SET_ID, returns STATUS.

```c
#define msgFIMGetId

MakeMsg(clsFontInstallMgr, 3)
```
### msgFIMSetId

Set the font file's ID.

Takes P_FIM_GET_SET_ID, returns STATUS.

```c
define msgFIMSetId
    MakeMsg(clsFontInstallMgr, 4)
```

**Arguments**

typedef struct FIM_GET_SET_ID {
    IM_HANDLE handle;       // Font handle to get IDs on.
    FIM_SHORT_ID id;        // Out: short version of ID.
    FIM_LONG_ID longId;     // Out: long ID version.
} FIM_GET_SET_ID, *P_FIM_GET_SET_ID;

**Comments**

If the short version of the ID is 0 then the long version of the ID is used.

Note: A font ID is not normally changed. This message is here to allow a tool that edits font IDs to be written.

### msgFIMFindId

Finds a font handle given a short ID.

Takes P_FIM_FIND_ID, returns STATUS.

```c
define msgFIMFindId
    MakeMsg(clsFontInstallMgr, 5)
```

**Arguments**

typedef struct FIM_FIND_ID {
    FIM_SHORT_ID id;        // ID, short form
    IM_HANDLE handle;       // Out: resulting handle
} FIM_FIND_ID, *P_FIM_FIND_ID;

**Return Value**

stsNoMatch  font handle not found.

### msgFIMGetNameFromId

Passes back font name given an short ID.

Takes P_FIM_GET_NAME_FROM_ID, returns STATUS.

```c
define msgFIMGetNameFromId
    MakeMsg(clsFontInstallMgr, 6)
```

**Arguments**

typedef struct FIM_GET_NAME_FROM_ID {
    FIM_SHORT_ID id;        // ID, short form
    P_STRING pName;         // Out: name, max size is nameBufLength
} FIM_GET_NAME_FROM_ID, *P_FIM_GET_NAME_FROM_ID;

**Return Value**

stsNoMatch  short ID not found.

**See Also**

msgIMGetName  Gets the name given a handle.

### msgFIMGetInstalledIdList

Passes back a list of the short IDs of all installed fonts.

Takes P_FIM_GET_INSTALLED_ID_LIST, returns STATUS.

```c
define msgFIMGetInstalledIdList
    MakeMsg(clsFontInstallMgr, 7)
```
typedef enum FIM_PRUNE_CONTROL {
    fimNoPruning = 0,
    fimPruneDupFamilies = flag1,
    fimPruneSymbolFonts = flag2
} FIM_PRUNE_CONTROL, *P_FIM_PRUNE_CONTROL;

typedef struct FIM_GET_INSTALLED_ID_LIST {
    FIM_PRUNE_CONTROL prune;     // What sort of pruning should be done
    OBJECT list;                 // Out: list
} FIM_GET_INSTALLED_ID_LIST, *P_FIM_GET_INSTALLED_ID_LIST;

This list is pruned so that it is useable as a user pick list. For example, if both Helvetica and Helvetica Bold are in the system, only Helvetica is on this list.

THE CALLER MUST DESTROY THE LIST OBJECT WHEN IT IS FINISHED USING IT.

See Also

msgIMGetList  Gets a list of all handles.
This file contains the API definition for dsHWXProtoInstallMgr.

dsHWXProtoInstallMgr inherits from dsInstallMgr.

It performs handwriting prototype installation and maintenance.

There is a single instance of dsHWXProtoInstallMgr in the system; the well-known uid theInstalledHWXProtos.

The hwxproto manager maintains the installed and deinstalled handwriting prototype sets on the system, and their relation to the installable handwriting translation engines, which are kept on theHWXEngines service manager. The hwxproto manager differs from a generic install manager in the area of hwx engine identification and its tie-in with theHWXEngines service manager.

A handwriting prototype set is a directory which contains engine-specific information. Each installed engine on the system must have at least one hwxproto set in theInstalledHWXProtos in order for it to be used.

See Also

#include <instlmgr.h>

Common #defines and typedefs

Status Codes

The hwx engine for this prototype set is not available
#define stsHIMEngineUnavailable MakeStatus(clsHWXProtoInstallMgr, 0)

Can’t change current hwx prototype; hwx engine is in use with it
#define stsHIMCurrentEngineInUse MakeStatus(clsHWXProtoInstallMgr, 1)

No training for this handwriting set.
#define stsHIMNoTraining MakeStatus(clsHWXProtoInstallMgr, 2)

No practice for this handwriting set.
#define stsHIMNoPractice MakeStatus(clsHWXProtoInstallMgr, 2)

Filesystem attribute definitions

HWX Engine name
#define himAttrEngineName FSMakeStrAttr(clsHWXProtoInstallMgr, 0)
Is the engine for this hwxproto available?

```c
#define himAttrEngineAvailable FSMakeFix32Attr(clsHWXProtoInstallMgr, 1)
typedef enum HIM_ATTR_ENGINE_AVAILABLE {
    himEngineAvailable = 0,  // Same as no attribute
    himEngineUnavailable = 1
} HIM_ATTR_ENGINE_AVAILABLE;
```

HWX Training window class. This is stamped on the HWX Engine Service class directory.

```c
#define himAttrTrainingWinClass FSMakeFix32Attr(clsHWXProtoInstallMgr, 3)
```

HWX Practice window class. This is stamped on the HWX Engine Service's class directory.

```c
#define himAttrPracticeWinClass FSMakeFix32Attr(clsHWXProtoInstallMgr, 4)
```

Gesture Training window class. This is stamped on the Gesture Engine Service's class directory.

```c
#define himAttrGestTrainingWinClass FSMakeFix32Attr(clsHWXProtoInstallMgr, 5)
```

Gesture Practice window class. This is stamped on the Gesture Engine Service's class directory.

```c
#define himAttrGestPracticeWinClass FSMakeFix32Attr(clsHWXProtoInstallMgr, 6)
```

### Popup Training and Practice tags

```c
#define msgHIMPopupTraining MakeMsg(clsHWXProtoInstallMgr, 100)
#define msgHIMPopupPractice MakeMsg(clsHWXProtoInstallMgr, 101)
#define msgHIMPopupGestureTraining MakeMsg(clsHWXProtoInstallMgr, 102)
#define msgHIMPopupGesturePractice MakeMsg(clsHWXProtoInstallMgr, 103)
#define tagHIMPopupTraining MakeTag(clsHWXProtoInstallMgr, 1)
#define tagHIMPopupPractice MakeTag(clsHWXProtoInstallMgr, 2)
#define tagHIMPopupGestureTraining MakeTag(clsHWXProtoInstallMgr, 3)
#define tagHIMPopupGesturePractice MakeTag(clsHWXProtoInstallMgr, 4)
#define hlpHIMTrainingButton MakeTag(clsHWXProtoInstallMgr, 100)
#define hlpHIMPracticeButton MakeTag(clsHWXProtoInstallMgr, 101)
#define hlpHIMGestureTrainingButton MakeTag(clsHWXProtoInstallMgr, 102)
#define hlpHIMGesturePracticeButton MakeTag(clsHWXProtoInstallMgr, 103)
```

### Messages

#### msgNew

Creates a new handwriting prototype install manager.

Takes `P_HIM_NEW`, returns `STATUS`. Category: class message.

**Arguments**

```c
typedef struct HIM_NEW {
    HIM_NEW_FIELDS
} HIM_NEW, P_HIM_NEW;
```

**Comments**

There is only one instance of this class, `theInstalledHWXProtos`, in the system. Clients should never send `msgNew`.

#### msgHIMGetEngine

Gets the name and availability of the engine associated with this hwxproto.

Takes `P_HIM_GET_SET_ENGINE`, returns `STATUS`.

```c
#define msgHIMGetEngine MakeMsg(clsHWXProtoInstallMgr, 1)
```
**Arguments**

typedef struct HIM_GET_SET_ENGINE {
    IM_HANDLE handle;  // hwxproto handle to get engine name of.
    P_STRING pEngineName; // Out: Name. Must have at least
        // nameBufLength bytes allocated.
    BOOLEAN available;  // Out: Is the engine available?
} HIM_GET_SET_ENGINE, *P_HIM_GET_SET_ENGINE;

**Comments**

Engine names can be up to nameLength characters long.

---

**msgHIMSetEngine**

Set the hwxproto's engine name.

Takes P_HIM_GET_SET_ENGINE, returns STATUS.

```c
#define msgHIMSetEngine MakeMsg(clsHWXProtoInstallMgr, 2)
```

**msgHIMAvailabilityChanged**

An hwx proto's engine availability has changed.

Takes P_HIM_AVAILABILITY_NOTIFY, returns STATUS. Category: observer notification.

```c
#define msgHIMAvailabilityChanged MakeMsg(clsHWXProtoInstallMgr, 20)
```

**Arguments**

typedef struct HIM_AVAILABILITY_NOTIFY {
    OBJECT manager;  // manager that sent notification
    IM_HANDLE handle;  // handle that changed
    BOOLEAN available;  // new engine availability state
} HIM_AVAILABILITY_NOTIFY, *P_HIM_AVAILABILITY_NOTIFY;
This file contains the API definition for clsIniFileHandler.

clsIniFileHandler inherits from clsObject.

Reads and processes a .ini file.

.ini files are used to ask the system to install multiple applications, services, or any installable entity. A .ini file is an ASCII file that contains the path of each item to be installed on a separate line. Examples of .ini files include app.ini (applications) and service.ini (services).

To process a .ini file, simply create an instance of clsIniFileHandler. The newArgs specify the location of the .ini file. The .ini file will be completely processed as part of the msgNew. Free the ini file handler immediately after creating it.

```c
#ifndef INIFILE_INCLUDED
#define INIFILE_INCLUDED
#ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif
#ifndef INSTLMGR_INCLUDED
#include <instlmgr.h>
#endif
```

**Messages**

**msgNew**

Creates a new ini file processor.

Takes P_INI_FILE_NEW, returns STATUS. Category: class message.

Arguments

typedef struct INI_FILE_STYLE {
    U16 deleteFileWhenDone : 1,  // Delete file after processing it.
    returnInstallErrors : 1,    // aborts the install and returns error
    style;
    spare : 11;                 // unused (reserved)
} INI_FILE_STYLE, *P_INI_FILE_STYLE;

typedef struct INI_FILE_NEW_ONLY {
    INI_FILE_STYLE style;
    IM_INSTALL_EXIST exist;    // What to do if the item already
    // exists.
    FS_LOCATOR locator;       // .ini file location.
    OBJECT manager;           // Install manager to send
    // msgIMInstalls to.
    FS_ATTR_LABEL listAttrLabel; // List attr; 0 if not needed.
    OBJECT listHandle;        // FS handle for list attr; objNull
    // if not needed.
    OBJECT relDir;            // Relative dir for ini file paths.
    spare[8];                // unused (reserved)
} INI_FILE_NEW_ONLY, *P_INI_FILE_NEW_ONLY;

#define iniFileNewFields
#define objectNewFields
#define INI_FILE_NEW_ONLY iniFile;
typedef struct INI_FILE_NEW {
    iniFileNewFields
} INI_FILE_NEW, *P_INI_FILE_NEW;

This message will return after the entire file has been processed. The file is processed by sending msgIMInstall to the specified install manager for each path in the .ini file.

pArgs->iniFile.listAttrLabel and pArgs->iniFile.listHandle are passed through to msgIMInstall. See instlmgr.h for details on msgIMInstall.

msgNewDefaults

Initializes the INI_FILE_NEW structure to default values.

Takes P_INI_FILE_NEW, returns STATUS. Category: class message.

typedef struct INI_FILE_NEW {
    iniFileNewFields
} INI_FILE_NEW, *P_INI_FILE_NEW;

Sets

iniFile.style.returnInstallErrors = true;
iniFile.style.deleteFileWhenDone = false;
iniFile.listAttrLabel = 0;
iniFile.listHandle = objNull;
iniFile.exist = imExistReactivate;
INSTALL.H

This file contains definitions for IMProgramInstall and IMModuleLoad. The functions described in this file are contained in INSTALL.LIB.

APPLICATION DEVELOPERS MUST USE THESE FUNCTIONS INSTEAD OF OSPProgramInstall AND OSMODULELoad.

OSProgramInstall and OSMODULELoad do not dispatch messages, because they are Ring 0 routines. This will cause the system to lock up if the code being loaded needs to send messages to the process that installed it, as all applications and services do.

 ifndef INSTALL_INCLUDED
 define INSTALL_INCLUDED

IMProgramInstall
Low-level .exe installation routine.

Returns STATUS.

Function Prototype

 STATUS EXPORTED IMProgramInstall(
 P_STRING pPath, // WorkingDir relative path of 
 P_STRING pWorkingDir, // WorkingDir relative path of where 
 P_OS_PROG_HANDLE pProgHandle, // Out: program handle 
 P_STRING pBadName, // Out: if error, dll/exe that was bad 
 P_STRING pBadRef // Out: If error, reference that was bad 
);

IMModuleLoad
Low-level .dll installation routine.

Returns STATUS.

Function Prototype

 STATUS EXPORTED IMModuleLoad(
 P_STRING pPath, // WorkingDir relative path of 
 P_STRING pWorkingDir, // WorkingDir relative path of where 
 P_OS_PROG_HANDLE pProgHandle, // Out: program handle 
 P_STRING pBadName, // Out: if error, dll that was bad 
 P_STRING pBadRef // Out: If error, reference that was bad 
);


This file contains the class definition and methods for clsInstallMgr.

clsInstallMgr inherits from clsObject.

Provides the basic facilities for installing items.

NOTE: THE MESSAGES IN THIS CLASS ARE SENT TO THE INSTALL MANAGER, NOT TO THE HANDLES.

clsInstallMgr provides almost everything needed to manage installable items. An installable item is anything that can be installed and deinstalled on a Penpoint machine, such as fonts, applications, services, handwriting prototype sets, etc. You create an instance of clsInstallMgr for each category of installable item. Penpoint creates well-known install managers for the following categories at cold boot time:

- theInstalledHWXProtos: Handwriting prototype sets
- theInstalledPrefs: Preference sets
- theInstalledPDicts: Personal dictionaries

In addition there are several well-known install managers that are created from subclasses of clsInstallMgr:

- theInstalledApps: Applications (appimgr.h)
- theInstalledServices: Services (servimgr.h)
- theInstalledFonts: Fonts (fontmgr.h)

clsInstallMgr makes use of the filesystem to keep a database of the installed items. Each item is represented by a file or directory handle. This is a big win for items which *are* files or directories; the InstallMgr's handle is a handle onto the actual item. There is an extra level of indirection for items which are not files. The item's ID (whatever that means for a particular type of item) is stored as an attribute of the handle. An item's name is the name of that item's filesystem node. This means that items on a given install manager must have unique names.

An install manager has a base directory in which it keeps its items' filesystem nodes. The createInitial style bit determines whether the install manager creates an initial set of item handles from whatever is in this directory when the install manager is first created.

clsInstallMgr provides an API for installing new items and deinstalling existing items. An item is installed from a location on an external filesystem.

An item can be deinstalled, which removes all traces of the item from the system.

The install manager maintains a bit which specifies if an item has changed. It is the client's responsibility to maintain this bit by sending msgIMSetModified when it modifies an item. The install manager will remember the time and date that the item was modified.

Install managers also maintains a "current" item, and provide an API for getting and setting the current item. This is used by theInstalledHWXProtos, theInstalledPrefs and theInstalledPDicts to specify which handwriting prototype set, preferences, or personal dictionary the system is actively using. A
current item is optional; some install managers (theInstalledApps, theInstalledServices) do not make use of a current item.

An item can be marked as being "in use". This means that the item cannot be deinstalled. The current item is considered to be in use.

Each install manager can have a verifier object, which it queries whenever installation takes place. The verifier object makes sure that the item being installed is valid for this install manager.

An InstallMgr sends notification to its observers whenever an item is installed, deinstalled, the current item changed, etc. Subclasses of clsInstallMgr can turn notification generation on and off with msgIMSetNotify. Notification is on by default.

A subset of the notification messages are also sent to any observers of an item's handle. This allows clients who are only interested in a particular item to monitor just that item. The messages sent are:

- msgIMNameChanged
- msgIMInUseChanged
- msgIMModifiedChanged
- msgIMDeinstalled
- msgIMCurrentChanged (sent to both old and new current handles)

Clients access installable managers via an ObjectCall interface. clsInstallMgr can accommodate simultaneous access by multiple clients if the "shared" style bit is set true (the default). This causes it to semaphore all of its operations. This semaphore is available to subclasses via msgIMGetSema, and should be used to protect all subclass messages if multiple clients will be accommodated. clsInstallMgr also sets objCapCall on by default.

There is a well-known, shared list object (see list.h) that is a list of all the install managers in the system. This object is called theInstallManagers. You can observe this list and get notification when an install manager is added and removed. See msgListNotifyAddition and msgListNotifyDeletion.

clsFontInstallMgr, clsAppInstallMgr, and clsServiceMgr inherit from clsInstallMgr. See fontmgr.h, appimgr.h and servmgr.h for these classes.

#ifndef INSTLMGR_INCLUDED
#define INSTLMGR_INCLUDED
#endif

#ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif

#ifndef FS_INCLUDED
#include <fs.h>
#endif

#ifndef LIST_INCLUDED
#include <list.h>
#endif

#ifndef TKTABLE_INCLUDED
#include <tktable.h>
#endif

#ifndef OPTION_INCLUDED
#include <option.h>
#endif
Common #defines and typedefs

Handle type
typedef OBJECT IM_HANDLE, *P_IM_HANDLE;

Warning Codes
Some install manager request has been user cancelled
#define stsIMRequestCancelled MakeWarning(clsInstallMgr, 0)

Quick Help Tags
#define appQHInstallMgr MakeTag(clsInstallUISheet, 32)
#define svcQHInstallMgr MakeTag(clsInstallUISheet, 33)
#define hwxQHInstallMgr MakeTag(clsInstallUISheet, 34)
#define gestQHInstallMgr MakeTag(clsInstallUISheet, 35)
#define dictQHInstallMgr MakeTag(clsInstallUISheet, 36)
#define fontsQHInstallMgr MakeTag(clsInstallUISheet, 37)
#define userPfleQHInstallMgr MakeTag(clsInstallUISheet, 38)

Status Codes
The item is current, so cannot be removed.
#define stsIMCurrent MakeStatus(clsInstallMgr, 1)
An item to be installed failed verification.
#define stsIMInvalidItem MakeStatus(clsInstallMgr, 2)
A new name cannot be created for this item.
#define stsIMUniqueNameFailed MakeStatus(clsInstallMgr, 3)
The item is in use, so cannot be removed.
#define stsIMInUse MakeStatus(clsInstallMgr, 6)
The item to be installed is already installed.
#define stsIMAlreadyInstalled MakeStatus(clsInstallMgr, 8)
An invalid handle was passed in.
#define stsIMBadHandle MakeStatus(clsInstallMgr, 20)

File System Attribute Definitions
Note: Most clients do not deal with attributes directly.
Node's home on an external volume. Absolute path.
This attribute is used only during installation.
#define imAttrHome FSMakeStrAttr(clsInstallMgr, 0)
Is this node the current node? Use IM_ATTR_CURRENT values.
#define imAttrCurrent FSMakeFix32Attr(clsInstallMgr, 2)
typedef enum IM_ATTR_CURRENT {
    imNotCurrent = 0,  // Same as no attribute
    imCurrent = 1
} IM_ATTR_CURRENT;
Is this node in use? Use IM_ATTR_INUSE values.

```c
#define irnAttrInUse    FSMakeFix32Attr(clsInstallMgr, 3)
typedef enum IM_ATTR_INUSE {
    imNotInUse = 0,    // Same as no attribute
    imInUse = 1
} IM_ATTR_INUSE;
```

Has this node been modified? Use IM_ATTR_MODIFIED values.

```c
#define irnAttrModified FSMakeFix32Attr(clsInstallMgr, 4)
typedef enum IM_ATTR_MODIFIED {
    imNotModified = 0,   // Same as no attribute
    imModified = 1
} IM_ATTR_MODIFIED;
```

Ref count. When an item is installed the installer can choose to maintain a reference count if the item is already installed.

```c
#define irnAttrRefCount FSMakeFix32Attr(clsInstallMgr, 5)
```

Is this item on some other item's dependency list? Use IM_ATTR_DEPENDENT values.

```c
#define irnAttrDependent FSMakeFix32Attr(clsInstallMgr, 7)
typedef enum IM_ATTR_DEPENDENT {
    imNotDependent = 0,   // Same as no attribute
    imDependent = 1
} IM_ATTR_DEPENDENT;
```

Is this item a system inviolate item? Use IM_ATTR_SYSTEM values.

```c
#define irnAttrSystem   FSMakeFix32Attr(clsInstallMgr, 8)
typedef enum IM_ATTR_SYSTEM {
    imNotSystem = 0,    // Same as no attribute
    imSystemInviolate,
    imSystemNotRenameable = flag0,
} IM_ATTR_SYSTEM;
```

Version string

```c
#define irnAttrVersion FSMakeStrAttr(clsAppInstallMgr, 3)
```

---

**Debug Flags**

```c
#define installDebugFlag 'I'
```

---

**Messages**

**msgNew**

Creates a new install manager.

Takes P_IM_NEW, returns STATUS. Category: class message.

**Arguments**

```c
typedef struct IM_STYLE {
    UI16 shared : 1,    // Provide concurrency protection.
    createInitial : 1,  // Create initial list of handles from
                        // contents of base directory.
    autoSetCurrent : 1, // Choose any item as the initial current
                        // setting if no one has current attr set.
    copyOnInstall : 1,  // Copy nodes to manager's dir or create
                        // handles directly on install locator.
    addToGlobalList : 1, // Add this installmgr to the InstallManagers.
    createIcon : 1,     // Create an icon for this install manager.
    privatel : 1,       // Always set this to false.
    duplicatable : 1,   // Items in this installmgr can be duplicated.
} IM_STYLE;
```
usesVersions : 1, // Items in this instlmgr have versions.
reserved : 7;
U16 sizeCol : 1, // Show size column in Settings NB card.
hwxTypeCol : 1, // Show hwx engine type column.
svcTypeCol : 1, // Show service type column.
modifiedCol : 1, // Show modified column.
currentColorCol : 1, // Show current column.
inUseCol : 1, // Show inUse column.
reserved1 : 10;
U32 helpId;       // Help tag for instlmgr's Settings NB card.
U16 spare1;
U16 spare2;
} IM_STYLE, *P_IM_STYLE;

typedef struct IM_NEW_ONLY {
  IM_STYLE style;
  FS_DIR_NEW_MODE dirMode;       // Default mode for dir handles.
  FS_FILE_NEW_MODE fileMode;     // Default mode for file handles.
  FS_LOCATOR locator;           // Base directory. Instlmgr will
  // create it if it doesn't exist.
  P_STRING p SingularName;       // Singular name of installer. Must be
  // <= nameLength in size.
  P_STRING pName;                // Plural name of installer. Must be
  // <= nameLength in size.
  P_STRING pInstallPath;         // Base path for installable items,
  // (i.e. \penpoint\app).
  OBJECT verifier;               // Verifier object. Can be null.
  OS_HEAP_ID heap;              // Instlmgr heap. Must be global.
  // Can be osInvalidHeapId; instlmgr
  // will use global heap of the task
  // that this object is created in.
  P_TK_TABLE_ENTRY pSettingsMenu; // Additional controls for this
  // instlmgr's Settings NB card.
  U32 settingsMenuSize;         // Size (in bytes) of pSettingsMenu.
  U32 unused1;
  U32 unused2;
  U32 unused3;
  U32 unused4;
} IM_NEW_ONLY, *P_IM_NEW_ONLY;
#define installMgrNewFields
  objectNewFields
  IM_NEW_ONLY installMgr;

typedef struct IM_NEW {
  installMgrNewFields
} IM_NEW, *P_IM_NEW;

Comments
The locator field specifies the directory where the managed items live. If this directory does not exist it
will be created.

msgNewDefaults
Initializes the IM_NEW structure to default values.
Takes P_IM_NEW, returns STATUS. Category: class message.

Comment
Clients do not normally change the defaults.
Zeroes out `installMgr` and sets

```c
object.cap | objCapCall;
installMgr.style.shared = true;
installMgr.style.createInitial = true;
installMgr.style.updateOK = true;
installMgr.style.copyOnInstall = true;
installMgr.style.addToGlobalList = true;
installMgr.style.private1 = false;
installMgr.style.duplicatable = false;
installMgr.style.createIcon = true;
installMgr.style.duplicatable = false;
installMgr.style.sizeCol = true;
installMgr.style.usesVersions = false;
installMgr.style.sizeCol = true;
installMgr.dirMode = fsUnchangeable;
installMgr.fileMode = fsSharedMemoryMap;
installMgr.pInstallPath = pNull;
installMgr.verifier = objNull;
installMgr.heap = osInvalidHeapId;
installMgr.pSettingsMenu = objNull;
installMgr.settingsMenuSize = 0;
```

---

**msgDestroy**

Frees the install manager.

Takes `OBJ_KEY`, returns `STATUS`.

**Comments**

Note: This message does not destroy the install manager’s directory, nor any files/directories in that directory.

---

**msgDump**

Prints out the items in the install manager and their state.

Takes `OBJ_KEY`, returns `STATUS`.

---

**msgIMGetStyle**

Passes back the current style settings.

Takes `P_1M_STYLE`, returns `STATUS`.

```c
#define msgIMGetStyle MakeMsg(clsInstallMgr, 1)
typedef struct IM_STYLE {
    U16 shared : 1, // Provide concurrency protection.
    createInitial : 1, // Create initial list of handles from
        // contents of base directory.
    autoSetCurrent : 1, // Choose any item as the initial current
        // setting if no one has current attr set.
    copyOnInstall : 1, // Copy nodes to manager’s dir or create
        // handles directly on Install locator.
    addToGlobalList : 1, // Add this instlmgr to theInstallManagers.
    createIcon : 1, // Create an icon for this install manager.
    private1 : 1, // Always set this to false.
    duplicatable : 1, // Items in this installmgr can be duplicated.
    usesVersions : 1, // Items in this installmgr have versions.
    reserved : 7;
    U16 sizeCol : 1, // Show size column in Settings NB card.
    hwxTypeCol : 1, // Show hwx engine type column.
    svcTypeCol : 1, // Show service type column.
    modifiedCol : 1, // Show modified column.
    currentCol : 1, // Show current column.
} IM_STYLE;
```
### msgIMSetStyle

Sets the current style.

Takes `P_IM_STYLE`, returns STATUS.

```
define msgIMSetStyle   MakeMsg(clsInstallMgr, 2)
```  

#### Arguments

<table>
<thead>
<tr>
<th>Message</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>U16</td>
<td>shared</td>
</tr>
<tr>
<td></td>
<td>: 1</td>
</tr>
<tr>
<td>U16</td>
<td>createInitial</td>
</tr>
<tr>
<td></td>
<td>: 1</td>
</tr>
<tr>
<td>U16</td>
<td>autoSetCurrent</td>
</tr>
<tr>
<td></td>
<td>: 1</td>
</tr>
<tr>
<td>U16</td>
<td>copyOnInstall</td>
</tr>
<tr>
<td></td>
<td>: 1</td>
</tr>
<tr>
<td>U16</td>
<td>addToGlobalList</td>
</tr>
<tr>
<td></td>
<td>: 1</td>
</tr>
<tr>
<td>U16</td>
<td>createIcon</td>
</tr>
<tr>
<td></td>
<td>: 1</td>
</tr>
<tr>
<td>U16</td>
<td>private1</td>
</tr>
<tr>
<td></td>
<td>: 1</td>
</tr>
<tr>
<td>U16</td>
<td>duplicatable</td>
</tr>
<tr>
<td></td>
<td>: 1</td>
</tr>
<tr>
<td>U16</td>
<td>usesVersions</td>
</tr>
<tr>
<td></td>
<td>: 1</td>
</tr>
<tr>
<td>U16</td>
<td>sizeCol</td>
</tr>
<tr>
<td></td>
<td>: 1</td>
</tr>
<tr>
<td>U16</td>
<td>hwxTypeCol</td>
</tr>
<tr>
<td></td>
<td>: 1</td>
</tr>
<tr>
<td>U16</td>
<td>svcTypeCol</td>
</tr>
<tr>
<td></td>
<td>: 1</td>
</tr>
<tr>
<td>U16</td>
<td>modifiedCol</td>
</tr>
<tr>
<td></td>
<td>: 1</td>
</tr>
<tr>
<td>U16</td>
<td>currentCol</td>
</tr>
<tr>
<td></td>
<td>: 1</td>
</tr>
<tr>
<td>U32</td>
<td>inUseCol</td>
</tr>
<tr>
<td></td>
<td>: 1</td>
</tr>
<tr>
<td>U32</td>
<td>helpId</td>
</tr>
<tr>
<td></td>
<td>: 10</td>
</tr>
<tr>
<td>U16</td>
<td>spare1</td>
</tr>
<tr>
<td>U16</td>
<td>spare2</td>
</tr>
</tbody>
</table>

### msgIMGetInstallerName

Passes back the install manager's name.

Takes `P_STRING`, returns STATUS.

```
define msgIMGetInstallerName MakeMsg(clsInstallMgr, 3)
```  

#### Comments

- `pArgs` must point to a `nameBufLength` buffer.
- The install manager's name was set at `msgNew` time in `installMgr->pName`.

### msgIMGetInstaller SingularName

Passes back the install manager's singular name.

Takes `P_STRING`, returns STATUS.

```
define msgIMGetInstaller SingularName MakeMsg(clsInstallMgr, 51)
```  

#### Comments

- `pArgs` must point to a `nameBufLength` buffer.
- The install manager's name was set at `msgNew` time in `installMgr->pName`.
msgIMGetCurrent

Passes back the current item's handle.
Takes P_IM_HANDLE, returns STATUS.

#define msgIMGetCurrent MakeMsg(clsInstal1Mgr, 4)

Comments
Passes back objNull if there is no current handle.

msgIMSetCurrent

Sets the current item.
Takes IM_HANDLE, returns STATUS.

#define msgIMSetCurrent MakeMsg(clsInstal1Mgr, 5)

The argument is the handle to be made current. It can be objNull to indicate that no handle is the current one.
If the handle specified in the argument is already current then nothing is done (no observer message is generated).
Causes the install manager to notify observers with msgIMCurrentChanged.

msgIMSetInUse

Changes an item's in use setting.
Takes P_IM_SET_INUSE, returns STATUS.

#define msgIMSetInUse MakeMsg(clsInstal1Mgr, 6)

typedef struct IM_SET_INUSE {
  IM_HANDLE handle; // Handle of item to set inUse on.
  BOOLEAN inUse; // InUse value.
} IM_SET_INUSE, *P_IM_SET_INUSE;

Arguments
Setting inUse to true means that the item cannot be deinstalled.
Use msgIMGetState to query the value of this field.
Causes the install manager to notify observers with msgIMInUseChanged.

msgIMSetModified

Changes an item's modified setting.
Takes P_IM_SET_MODIFIED, returns STATUS.

#define msgIMSetModified MakeMsg(clsInstal1Mgr, 7)

typedef struct IM_SET_MODIFIED {
  IM_HANDLE handle; // Handle of item to set modified on.
  BOOLEAN modified; // Modified value.
} IM_SET_MODIFIED, *P_IM_SET_MODIFIED;

Arguments
Use msgIMGetState to query the value of this field.
Causes the install manager to notify observers with msgIMModifiedChanged.
msgIMGetName

Get the name of a item.

Takes P_IM_GET_SET_NAME, returns STATUS.

```c
#define msgIMGetName       MakeMsg(clsInstallMgr, 8)
```

**Arguments**

typedef struct IM_GET_SET_NAME {
   IM_HANDLE handle;  // Handle of item to get/set name on.
   P_STRING pName;    // In: (Set) Out: (Get) name. This
                     // pointer must reference a nameBufLength
                     // size buffer.
} IM_GET_SET_NAME, *P_IM_GET_SET_NAME;

msgIMSetName

Sets the name of a item.

Takes P_IM_GET_SET_NAME, returns STATUS.

```c
#define msgIMSetName       MakeMsg(clsInstallMgr, 9)
```

**Arguments**

typedef struct IM_GET_SET_NAME {
   IM_HANDLE handle;  // Handle of item to get/set name on.
   P_STRING pName;    // In: (Set) Out: (Get) name. This
                     // pointer must reference a nameBufLength
                     // size buffer.
} IM_GET_SET_NAME, *P_IM_GET_SET_NAME;

**Comments**
The name must be a legitimate file name and unique among all the items on this install manager. Causes the install manager to notify observers with msgIMNameChanged.

**Return Value**

stsFSNodeExists  An item with this name already exists.

msgIMGetVersion

Get the version string for this item.

Takes P_IM_GET_VERSION, returns STATUS.

```c
#define msgIMGetVersion    MakeMsg(clsInstallMgr, 37)
```

**Arguments**

typedef struct IM_GET_VERSION {
   IM_HANDLE handle;  // Handle of item to get version of.
   P_STRING pVersion; // Out: Version string. Pointer must
                        // reference a nameBufLength
                        // size buffer.
} IM_GET_VERSION, *P_IM_GET_VERSION;

**Comments**
Not all install managers have a version string. pVersion is set to pNull if there is no version.

msgIMGetList

Passes back a list of all the items on this install manager.

Takes P_LIST, returns STATUS.

```c
#define msgIMGetList       MakeMsg(clsInstallMgr, 14)
```

**Comments**
The memory for the list object is allocated out of the caller's local process heap.

CAUTION: Caller must destroy the list object when it is finished using it.
**msgIMGetState**

Gets the state of an item.

Takes P.IM_GET_STATE, returns STATUS.

```c
#define msgIMGetState

typedef struct IM_GET_STATE {
    IM_HANDLE handle;   // Handle of item to get state on.
    BOOLEAN current;    // Out: Is it the current item?
    BOOLEAN reserved;   // Reserved.
    BOOLEAN modified;   // Out: Is it modified?
    BOOLEAN inUse;      // Out: Is it in use?
} IM_GET_STATE, *P_IM_GET_STATE;
```

**msgIMGetSize**

Returns the size of an item.

Takes P.IM_GET_SIZE, returns STATUS.

```c
#define msgIMGetSize

typedef struct IM_GET_SIZE {
    IM_HANDLE handle;   // Handle of item to get size of.
    U32 size;           // Out: size.
} IM_GET_SIZE, *P_IM_GET_SIZE;
```

**msgIMInstall**

Installs a new item.

Takes P.IM_INSTALL, returns STATUS.

```c
#define msgIMInstall

typedef enum IM_INSTALL_EXIST {
    imExistUpdate = 0, // Copy new over existing.
    imExistReactivate = 1, // Deactivate existing, then activate new.
    imExistGenError = 2, // Return stsIMAlreadyInstalled.
    imExistGenUnique = 3, // Generate a unique name for the new item.
    imExistIncRefCount = 4 // Just increment ref count of existing item.
} IM_INSTALL_EXIST, *P_IM_INSTALL_EXIST;

typedef struct IM_INSTALL {
    FS_LOCATOR locator;   // Location of item on external filesystem.
    IM_INSTALL_EXIST exist; // What to do if item already exists.
    FS_ATTR_LABEL listAttrLabel; // Attr list to add install handle to.
    OBJECT listHandle;  // filesystem handle to put attr on.
    IM_HANDLE handle;   // Out: Handle of installed item.
} IM_INSTALL, *P_IM_INSTALL;
```

**Comments**

The install manager derives the item's name from the filesystem location specified in `pArgs->locator`. `pArgs->exist` controls what happens if an item of the same name as the item to be installed already exists. `pArgs->listAttrLabel` and `pArgs->listHandle` are used to specify an attr list to which the install handle is added. This is used to keep track of sub-apps and sub-services. Set these arguments to 0 if this should not be done.

Causes the install manager to notify observers with `msgIMInstalled`. The install manager also sends `msgIMModifiedChanged` if the modified states changed due to the install.
Return Value

stsIMInvalid  Item to be installed does not pass verification.

stsIMAlreadyInstalled  Item already installed and pArgs->exist == imExistGenError.

stsBadParam  pArgs->exist is set to an invalid value.

msgIMDeinstall

Deinstalls an item.
Takes P_IM_DEINSTALL, returns STATUS.

#define msgIMDeinstall  MakeMsg(clsInstallMgr, 19)

typedef struct IM_DEINSTALL {
    IM_HANDLE handle;  // Item to delete.
} IM_DEINSTALL, *P_IM_DEINSTALL;

All traces of the item are removed, including the item's handle.

Return Value

stsIMInUse  Item is in use; cannot be deinstalled.

msgIMDup

Creates a new item that is a duplicate of an existing one.
Takes P_IM_DUP, returns STATUS.

#define msgIMDup  MakeMsg(clsInstallMgr, 23)

typedef struct IM_DUP {
    IM_HANDLE handle;  // item to duplicate.
    P_STRING pName;  // new name. If pNull then a unique name
                      // is generated.
    IM_HANDLE newHandle;  // Out: Handle to the new item.
} IM_DUP, *P_IM_DUP;

Causes the install manager to notify observers with msgIMInstalled.

Return Value

stsIMAlreadyInstalled  An item with pArgs->name already exists.

msgIMFind

Finds a item's handle, given its name.
Takes P_IM_FIND, returns STATUS.

#define msgIMFind  MakeMsg(clsInstallMgr, 24)

typedef struct IM_FIND {
    P_STRING pName;  // Resource name to search for
    IM_HANDLE handle;  // Out: Resulting handle
} IM_FIND, *P_IM_FIND;

Return Value

stsNoMatch  Item not found.

msgIMGetSema

Gets the concurrency protection semaphore.
Takes P_OS_FAST_SEMA, returns STATUS.

#define msgIMGetSema  MakeMsg(clsInstallMgr, 25)

This message is for subclasses that need to do concurrency protection to their messages. Subclasses should get this semaphore and acquire and release it at the beginning and end of their messages. Subclasses should use this semaphore instead of creating one of their own in order to avoid race conditions.
msgIMGetDir
Passes back a directory handle on the install manager's directory.
Takes P_OBJECT, returns STATUS.
#define msgIMGetDir MakeMsg(clsInstallMgr, 26)
Comments
This dir handle is owned by the install manager; clients must not destroy it!

msgIMGetInstallPath
Passes back the install base path.
Takes P_STRING, returns STATUS.
#define msgIMGetInstallPath MakeMsg(clsInstallMgr, 27)
Comments
The install base path is an absolute path to the install manager's directory.
pArgs must point to an fsPathBufLength sized buffer.

msgIMGetVerifier
Passes back the current verifier object.
Takes P_OBJECT, returns STATUS.
#define msgIMGetVerifier MakeMsg(clsInstallMgr, 33)
Comments
This object is sent msgIMVerify whenever an item is attempted to be installed. The verifier should return stsOK if the item is valid, stsFailed if it isn't.

msgIMSetVerifier
Sets the current verifier object.
Takes OBJECT, returns STATUS.
#define msgIMSetVerifier MakeMsg(clsInstallMgr, 34)
Comments
This object is sent msgIMVerify whenever an item is attempted to be installed. The verifier should return stsOK if the item is valid, stsFailed if it isn't.

msgIMVerify
Verify the validity of an item that is being installed.
Takes OBJECT, returns STATUS.
#define msgIMVerify MakeMsg(clsInstallMgr, 35)
Comments
This message is sent to an install manager's verifier object whenever an installation is attempted.
pArgs specifies the node being installed. It is either a file handle or a dir handle. The verifier object should determine if the item to be installed is valid, and return stsOK if so, stsFailed if not.

msgIMExists
Verify the existance of an item that is being installed.
Takes P_IM_EXISTS, returns STATUS.
#define msgIMExists MakeMsg(clsInstallMgr, 61)
typedef struct IM_EXISTS {  
    OBJECT source;      // In: {File|Dir} handle of item to be installed.  
    IM_HANDLE handle;   // Out: Handle of item if found.  
} IM_EXISTS, *P_IM_EXISTS;

This message is self sent whenever an installation is attempted.

pArgs specifies the node being installed. It is either a file handle or a dir handle. The handler should
determine if the item to be installed already exists. Returns stsOK if the item is found; stsFailed
otherwise.

#### UI Messages

**msgIMUUIDeinstall**
Installs a new item with a user interface.

Takes P_IM_UI_INSTALL, returns STATUS.

```c
#define msgIMUUIDeinstall MakeMsg(clsInstallMgr, 58)
```

**Arguments**

typedef struct IM_UI_INSTALL {  
    FS_LOCATOR locator;  // Location of item on external filesystem.  
    IM_HANDLE handle;  // Out: Handle of installed item.  
} IM_UI_INSTALL, *P_IM_UI_INSTALL;

**Comments**
Performs msgIMInstall, but lets the user decide exist behavior. Pops up a progress note which allows the
user to cancel the install. Informs the user of successful or unsuccessful completion.

**Return Value**
Returns msgIMInstall statuses.

**msgIMUIDup**
Duplicates and item with a UI.

Takes P_IM_UI_DUP, returns STATUS.

```c
#define msgIMUIDup MakeMsg(clsInstallMgr, 39)
```

**Arguments**

typedef struct IM_UI_DUP {  
    IM_HANDLE handle;  // item to duplicate.  
    P_STRING pName;  // new name. If pNull then a unique name  
                     // is generated.  
    IM_HANDLE newHandle;  // Out: Handle to the new item.  
} IM_UI_DUP, *P_IM_UI_DUP;

**Return Value**
Returns msgIMDup statuses.
Notification Messages

**msgIMNameChanged**
The name of a item has changed.

Takes P_IM_NOTIFY, returns STATUS. Category: observer notification.

```c
#define msgIMNameChanged MakeMsg(clsInstallMgr, 40)
```

**Arguments**
- `typedef struct IM_NOTIFY {
  OBJECT manager; // manager that sent notification.
  IM_HANDLE handle; // handle that changed.
  U8 reserved[40];
} IM_NOTIFY, *P_IM_NOTIFY;`

**msgIMCurrentChanged**
The current item has changed.

Takes P_IM_CURRENT_NOTIFY, returns STATUS. Category: observer notification.

```c
#define msgIMCurrentChanged MakeMsg(clsInstallMgr, 42)
```

**Arguments**
- `typedef struct IM_CURRENT_NOTIFY {
  OBJECT manager; // manager that sent notification
  IM_HANDLE newHandle; // the new current handle
  IM_HANDLE oldHandle; // the previous current handle
  U8 reserved[40];
} IM_CURRENT_NOTIFY, *P_IM_CURRENT_NOTIFY;`

**msgIMInUseChanged**
An item's inUse attribute has changed.

Takes P_IM_INUSE_NOTIFY, returns STATUS. Category: observer notification.

```c
#define msgIMInUseChanged MakeMsg(clsInstallMgr, 43)
```

**Arguments**
- `typedef struct IM_INUSE_NOTIFY {
  OBJECT manager; // manager that sent notification
  IM_HANDLE handle; // handle that changed
  BOOLEAN inUse; // new inUse state
  U8 reserved[40];
} IM_INUSE_NOTIFY, *P_IM_INUSE_NOTIFY;`

**msgIMModifiedChanged**
An item's modified attribute has changed.

Takes P_IM_MODIFIED_NOTIFY, returns STATUS. Category: observer notification.

```c
#define msgIMModifiedChanged MakeMsg(clsInstallMgr, 44)
```

**Arguments**
- `typedef struct IM_MODIFIED_NOTIFY {
  OBJECT manager; // manager that sent notification
  IM_HANDLE handle; // handle that changed
  BOOLEAN modified; // new modified state
  U8 reserved[40];
} IM_MODIFIED_NOTIFY, *P_IM_MODIFIED_NOTIFY;`
**msgIMInstalled**

A new item was installed.

Takes P_IM_NOTIFY, returns STATUS. Category: observer notification.

```c
#define msgIMInstalled            MakeMsg(clsInstallMgr, 45)
```

```c
typedef struct IM_NOTIFY {
    OBJECT manager;          // manager that sent notification.
    IM_HANDLE handle;        // handle that changed.
    US_ reserved[40];
} IM_NOTIFY, *P_IM_NOTIFY;
```

**msgIMDeinstalled**

An item has been deinstalled.

Takes P_IM_DEINSTALL_NOTIFY, returns STATUS. Category: observer notification.

```c
#define msgIMDeinstalled        MakeMsg(clsInstallMgr, 46)
```

```c
typedef struct IM_DEINSTALL_NOTIFY {
    OBJECT manager;          // manager that sent notification.
    IM_HANDLE handle;        // handle of item that was deinstalled.
    US_ pName[nameBufLength]; // item name.
    US_ pVersion[nameBufLength]; // item version.
    US_ reserved[40];
} IM_DEINSTALL_NOTIFY, *P_IM_DEINSTALL_NOTIFY;
```

Comments

Since the handle is no longer valid when this message is received, the **pArgs** includes all information about the item.

**Private**

**msgIMDeactivate**

Deactivate an item.

Takes P_IM_DEACTIVATE, returns STATUS.

```c
#define msgIMDeactivate          MakeMsg(clsInstallMgr, 20)
```

```c
typedef struct IM_DEACTIVATE {
    IM_HANDLE handle;         // item to deactivate.
} IM_DEACTIVATE, *P_IM_DEACTIVATE;
```

Comments

This removes everything but an empty filesystem node with attributes which represents the item. The item's handle and attributes remain intact.

Returns

**stsRequestNotSupported**  

style.copyOnInstall is false. Install mgs of this style don't support deactivation.

**msgIMAActivate**

Activate an item by copying it in from disk.

Takes P_IM_ACTIVATE, returns STATUS.

```c
#define msgIMAActivate            MakeMsg(clsInstallMgr, 21)
```
typedef struct IM_ACTIVATE {
    IM_HANDLE handle;      // Item to activate.
} IM_ACTIVATE, *P_IM_ACTIVATE;

The install manager also sends msgIMModifiedChanged if the modified state changed due to the activate.

stsIMAalreadyActive  Item is already active.
stsIMInvalidItem     There is nothing valid out on disk.

msgAppMgrGetMetrics
Returns generic icon for this installer.
Takes P_APP_MGR_METRICS, returns STATUS.

Install managers understand this message so they can present an icon for use by the disk manager. Install managers look for their icons in the system resource file.
Only the iconBitmap, smallIconBitmap, and name fields of pArgs are filled in.

msgIMAddCards
Asks the install manager to add option cards for the specified item.
Takes P_IM_ADD_CARDS, returns STATUS.

#define msgIMAddCards
MakeMsg(clsInstallMgr, 56)

typedef struct IM_ADD_CARDS {
    IM_HANDLE handle;      // Item to add cards for. Can be objNull.
    OPTION_TAG optionTag;  // msgOptionAddCards argument.
} IM_ADD_CARDS, *P_IM_ADD_CARDS;

The handle argument specifies the currently selected item. It may be objNull if there is no selection.
This message is a superset of msgOptionAddCards. The optionTag argument is exactly the same as that for msgOptionAddCards.

msgIMSetNotify
Turns notification generation on or off.
Takes BOOLEAN, returns STATUS.
#define msgIMSetNotify
MakeMsg(clsInstallMgr, 28)

msgIMGetNotify
Returns notification generation state.
Takes P_BOOLEAN, returns STATUS.
#define msgIMGetNotify
MakeMsg(clsInstallMgr, 29)

msgIMRemoveHandle
Removes and frees a handle from our internal list.
Takes OBJECT, returns STATUS.
#define msgIMRemoveHandle
MakeMsg(clsInstallMgr, 30)
msgIMRenameUninstalledItem
Renames an item on disk.
Takes P_IM_RENAME_UNINSTALLED, returns STATUS.

#define msgIMRenameUninstalledItem MakeMsg(clsInstallMgr, 53)

typedef struct IM_RENAME_UNINSTALLED {
    FS_LOCATOR locator;  // Location of item to rename. Must not
    P_STRING pOldName;   // be an absolute path!
    P_STRING pNewName;   // Old name.
} IM_RENAME_UNINSTALLED, *P_IM_RENAME_UNINSTALLED;

msgIMGetSettingsMenu
Sets a pointer to the tkTable entries for the Settings NB menu.
Takes PP_TK_TABLE_ENTRY, returns STATUS.

#define msgIMGetSettingsMenu MakeMsg(clsInstallMgr, 54)
pArgs must be the address of a P_TK_TABLE_ENTRY pointer.

msgIMGetItemImage
Gets the icons for a given item.
Takes P_IM_GET_ITEM_ICON, returns STATUS.

#define msgIMGetItemImage MakeMsg(clsInstallMgr, 57)

typedef struct IM_GET_ITEM_ICON {
    IM_HANDLE handle;     // Handle of item.
    OBJECT iconBitmap;    // Out: Icon bitmap.
    TAG iconTag;          // Out: Icon’s tag in resfile.
    BOOLEAN iconInSystemRes;  // Out: Is this icon in system
                              // resource file?
    OBJECT smallIconBitmap; // Out: Small icon bitmap.
    TAG smallIconTag;      // Out: Icon’s tag in resfile.
    BOOLEAN smallIconInSystemRes; // Out: Is this icon in system
                              // resource file?
    U32 reserved;
} IM_GET_ITEM_ICON, *P_IM_GET_ITEM_ICON;}
This file contains the API definition for clsInstallUISheet.

clsSettingsNB inherits from clsOption.

This class defines the Installer sheet in the Settings Notebook.

The Installer sheet contains one card for each installation category (apps, preferences, services, etc). Each category has an underlying install manager (see instlmgr.h). A card is automatically created when a new install manager is created, and deleted when an install manager is destroyed.

The Installer sheet allows a client to display a particular card and select an item within that card. Here’s example code which activates the Settings Notebook from the Bookshelf, turns it to the Installer sheet, displays a particular card, selects an item within that card, and finally opens the Settings Notebook:

```c
#include <auxnbmgr.h>
#include <instlsht.h>

ANM OPEN NOTEBOOK openNotebook;
APP METRICS am;
IUI_SELECT_ITEM selectItem;
OPTION CARD oc;
IUI_SHOW_CARD showCard;
STATUS s;

ObjectCall(msgBusySetState, theBusyManager, (P_ARGS) true);
openNotebook.notebook = anmSettings;
openNotebook.activateOnly = true;
ObjCallRet(msgANMOpenNotebook, theAuxNotebookMgr, &openNotebook, s);
ObjSendUpdateRet(msgAppGetMetrics, openNotebook.uid, &am, SizeOf(am), s);
oc.tag = tagSettingsInstallerSheet;
ObjSendUpdateRet(msgOptionShowCard, am.mainWin, &oc, SizeOf(oc), s);
ObjSendUpdateRet(msgOptionGetTopCard, am.mainWin, &oc, SizeOf(oc), s);
strcpy(showCard.pCardName, "Applications");
ObjSendRet(msgIUIShowCard, oc.win, &showCard, SizeOf(showCard), s);
strcpy(selectItem.pItemName, appMgrMetrics.name);
ObjSendRet(msgIUISelectItem, oc.win, &selectItem, SizeOf(selectItem), s);
openNotebook.notebook = anmSettings;
openNotebook.activateOnly = false;
ObjCallRet(msgANMOpenNotebook, theAuxNotebookMgr, &openNotebook, s);
ObjectCall(msgBusySetState, theBusyManager, (P_ARGS) false);
```
**Messages**

**msgIUIShowCard**
Show the specified Installer category card.
Takes P_IUI_SHOW_CARD, returns STATUS.

```c
#define msgIUIShowCard
MakeMsg(clsInstallUISheet, 1)
```

**Arguments**

typedef struct IUI_SHOW_CARD {
  CHAR pCardName[nameBufLength]; // Card Name. These names
  // correspond to installmgr
  // names; ie. Applications,
  // Services, Fonts. See
  // instlmgr.h.
  TAG itemTag; // If name is of zero length
  // use the tag
} IUI_SHOW_CARD, * P_IUI_SHOW_CARD;

**Return Value**

stsFailed The specified card was not found.

**msgIUISelectItem**
Set the selection to an item on the current card.
Takes P_IUI_SELECT_ITEM, returns STATUS.

```c
#define msgIUISelectItem
MakeMsg(clsInstallUISheet, 2)
```

**Arguments**

typedef struct IUI_SELECT_ITEM {
  CHAR pItemName[nameBufLength]; // Name of item to select.
  TAG itemTag; // If name is of zero length
  // use the tag
} IUI_SELECT_ITEM, * P_IUI_SELECT_ITEM;

**Return Value**

stsFailed The specified item was not found.

**msgIUIGetSelectionUID**
Gets the UID of the selection on the current card.
Takes P_UID, returns STATUS.

```c
#define msgIUIGetSelectionUID
MakeMsg(clsInstallUISheet, 5)
```

**Return Value**

stsFailed There is no selection.

**msgIUIGetSelectionName**
Gets the name of the selection on the current card.
Takes P_CHAR, returns STATUS.

```c
#define msgIUIGetSelectionName
MakeMsg(clsInstallUISheet, 6)
```

**Return Value**

stsFailed There is no selection.

**msgIUIGetMetrics**
Get installUI metrics.
Takes P_IUI_METRICS, returns STATUS.

```c
#define msgIUIGetMetrics
MakeMsg(clsInstallUISheet, 3)
```
typedef struct IUI_METRICS {
    OBJECT currentCard;          // Card displayed.
    CHAR pCurrentCardName[nameBufLength]; // Name of displayed card.
    TAG currentCardTag;          // Tag of card.
    CHAR spare[24];              // spare[24];
} IUI_METRICS, * P_IUI_METRICS;
This file contains the API definition for clsPDictProtoInstallMgr.

clsPDictProtoInstallMgr inherits from clsInstallMgr.

It performs personal dictionary installation and maintenance.

See Also

instlmgr.h

```c
#ifndef PDICTMGR_INCLUDED
#define PDICTMGR_INCLUDED
#ifndef INSTMGR_INCLUDED
#include <instlmgr.h>
#endif
#endif
```

### Common #defines and typedefs

#### Popup Editor messages and tags

```c
#define msgPIMPopUpEditor MakeMsg(clsPDictInstallMgr, 100)
#define tagPIMPopUpEditor MakeTag(clsPDictInstallMgr, 1)
#define hlpPIMEditorButton MakeTag(clsPDictInstallMgr, 100)
```

### Messages

#### msgNew

Creates a new personal dictionary install manager.

Takes P_PIM_NEW, returns STATUS. Category: class message.

**Arguments**

```c
typedef struct PIM_NEW {
    installMgrNewFields
} PIM_NEW, *P_PIM_NEW;
```

**Comments**

There is only one instance of this class, theInstalledPDicts, in the system. Clients should never send msgNew.
This file contains the API definition for clsServiceInstallMgr.

clsServiceInstallMgr inherits from clsCodeInstallMgr.

Manages installation and deinstallation of services.

There is a single instance of clsServiceInstallMgr in the system; the well-known uid theInstalledServices.

theInstalledServices performs installation and deinstallation of services, allows you to enumerate all of
the services that are currently installed, and find out their classes.

See service.h for the messages that a service implementor needs. See servmgr.h for the messages that a
service client uses to find and open a particular service.

Services provide non-application functionality under PenPoint; typically some form of background
server or device driver. Examples of services include: device drivers, inbox/outbox transfer agents such as
fax and e-mail, network protocol stacks, and databases.

A service is a directory, usually located under \penpoint\service on a given filesystem volume. The name
of the directory is the name of the service. Within this directory are one or more .dlls that make up the
service.

If a service includes more than one .dll there must also be a .dlc file which lists all the .dlls. The name of
the .dlc file (or the name of the .dll file if there is only one .dll) must be the same as the name of the
service. If a service is called MAIL, for example, its .dlc file must be named MAIL.DLC. You can use the
STAMPEXE utility to give a service an extended name. Be sure to stamp the .dlc file as well.

A service can contain an init.dll. This .dll will be loaded, run, and unloaded during service loading. This
can be used to set up or modify the service's resource file programmatically. A handle to the service's
resource file is available to init.dll via msgSvcGetClassMetrics.

When a service is installed, a service directory is created in the RAM filesystem. All of the state for that
service lives in this directory.

A service can have an optional MISC directory. This is very similar to an application's MISC directory.
MISC is used to store static data files that are common to all service instances. The MISC directory will
be copied into the service directory when the service is installed. You can get to the MISC directory
from a service instance by getting class metrics, then specifying a path of "MISC" relative to the service's
directory.

A service can have a resource file, called service.res. This is similar to an application's app.res file. The
resource file is automatically copied to the service directory in RAM when the service is installed, and a
resource file handle is opened on it and stored in the service class metrics. This resource file should
contain the service's UI components and quick-help resources. Each service's resource file handle is
added to the well-known resList theServiceResList. Quick-help searches theServiceResList as part of its
normal operation. Note that theServiceResList is not callable; you must ObjectSend to it.
There is an optional INST directory in a service directory, which contains saved service instance state nodes. Pre-configured service instances will be created from the nodes in this directory when the service is loaded (see service.h for details).

There can also be a service.ini and app.ini file in the service directory. These specify any additional services and applications that should be installed when this service is installed. These services and applications are deinstalled when the service is deinstalled. If one of these services or applications is already installed it is reference counted, not installed again.

A service is installed by sending msgIMInstall to theInstalledServices. Services are installed under user control from the Services card of the Settings Notebook, or via the pop-up quick installer (see qckinstall.h). \boot\penpoint\boot\service.ini specifies services that are automatically loaded when the system cold-boots.

Each installed service has a service directory in the RAM filesystem, under \penpoint\sys\service. For example, service MAIL would have \penpoint\sys\service\mail. The instance state nodes for the service are kept in a directory called INST, under this directory. If the service has preconfigured instances then they are copied to the INST directory when the service is first installed.

Each installed service is represented by a handle, in a fashion similar to other install managers (see instlmgr.h). This handle is a directory handle onto the service's directory in the RAM filesystem.

NOTE: THE MESSAGES IN THIS CLASS ARE SENT TO THE MANAGER, NOT TO THE HANDLES.

A service can be deinstalled. Deinstallation removes all traces of a service and decrements the reference count for any dependent services or applications. All service instances are removed from their service managers and freed when a service is deinstalled.

Deinstallation only occurs if the main service and all dependent applications and services agree to deinstall. A service or application can veto the deinstallation if it chooses. The default behavior for services is to veto if any service instance is open (in use).

The following superclass messages are not understood by clsServiceInstallMgr:

- msgIMGetCurrent
- msgIMSetCurrent
- msgIMSetName
- msgIMDup

The following notification messages are not sent by clsServiceInstallMgr:

- msgIMNameChanged
- msgIMCurrentChanged

NOTE: Each service must contain one and only one service class. Don't try and define more than one service class in a single service.
# Common #defines and typedefs

## Well-known filenames

These are the files created by clsServiceInstallMgr in a service's directory.

```c
#define svcResFileName     "service.res"
```

## Messages

### msgNew

Creates a new service installation manager.

Takes P_SIM_NEW, returns STATUS. Category: class message.

**Arguments**

typedef struct SIM_NEW {
  installMgrNewFields
} SIM_NEW, *P_SIM_NEW;

**Comments**

There is only one instance of this class, theInstalledServices, in the system. Clients should never send msgNew.

### msgSIMGetMetrics

Gets the specified service class's metrics.

Takes P_SIM_GET_METRICS, returns STATUS.

```c
#define msgSIMGetMetrics MakeMsg(clsServiceInstallMgr, 1)
```

**Arguments**

typedef struct SIM_GET_METRICS {
  IM_HANDLE  handle;  // Handle of service class to get metrics on.
  SVC_CLASS_METRICS metrics;  // Out: metrics.
} SIM_GET_METRICS, *P_SIM_GET_METRICS;

**Comments**

See service.h for SVC_CLASS_METRICS.
This file contains the API definition for clsSystem. clsSystem inherits from clsObject. Provides information about the system.

There is a single instance of clsSystem, theSystem. You send all clsSystem messages to theSystem. theSystem manages PenPoint booting. If you need to know when PenPoint booting reaches a certain stage or is complete then you can observe theSystem and receive msgBootStateChanged. You can also send msgSysGetBootState to find out what stage booting is currently at.

PenPoint Booting Sequence

<table>
<thead>
<tr>
<th>Cold Boot</th>
<th>Warm Boot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>Kernel</td>
</tr>
<tr>
<td>System Dlls Loaded (boot.dlc)</td>
<td>System Dll Upgrade</td>
</tr>
<tr>
<td>System Apps Installed (sysapp.ini)</td>
<td>System Dlls reinitialized</td>
</tr>
<tr>
<td>Initial App Installed</td>
<td>Instance 0’s/DLLMain()s rerun</td>
</tr>
<tr>
<td>Bookshelf Created</td>
<td>App Upgrade</td>
</tr>
<tr>
<td>Services Installed (service.ini)</td>
<td>Services Upgrade</td>
</tr>
<tr>
<td>Apps Installed (app.ini)</td>
<td>Run Initial App</td>
</tr>
<tr>
<td>Run Initial App</td>
<td>Boot Complete</td>
</tr>
<tr>
<td>Boot Complete</td>
<td></td>
</tr>
</tbody>
</table>

This header file defines constants for all the interesting PenPoint filesystem locations that you might be tempted to hard-code. Use these defines instead; for example, to set a string to the location where PenPoint applications live, use:

```c
strcpy(pFoo, sysBaseDir "\" sysInstallableAppDir);
```

PenPoint defines "live" areas for documents on volumes. The live area is where the volume’s bookshelf is. Use msgSysGetLiveRoot to access the live area on a volume.

```c
#ifndef SYSTEM_INCLUDED
#define SYSTEM_INCLUDED

#ifndef APPDIR_INCLUDED
#include <appdir.h>
#endif

#ifndef APPMGR_INCLUDED
#include <appmgr.h>
#endif

#ifndef UUID_INCLUDED
#include <uuid.h>
#endif

#ifndef SYSTEM_INCLUDED
#define SYSTEM_INCLUDED

#define sysInstallableAppDir "\"

#ifndef APPDIR_INCLUDED
#include <appdir.h>
#endif

#ifndef APPMGR_INCLUDED
#include <appmgr.h>
#endif

#ifndef UUID_INCLUDED
#include <uuid.h>
#endif
```
## System Debugging Flags

System debug flag is 'B', values are:

1 = Enable active doc cache tracing  
2 = Install items from the selected Volume at warm boot  
4 = Go into debugger when stdmsg functions are called  
8 = Enable serial port option sheet testing  
800 = Enable showing of the RAM (the selected Volume) Volume

## Common #defines and typedefs

penpoint.res is invalid. This is checked during cold and warm boot.

#define STS_SYS_INVALID_SYSTEM_RES_FILE MakeStatus(clsSystem, 1)

Penpoint base directory.

#define SYS_BASE_DIR "PENPOINT"

Filesystem locations off the base Penpoint directory.

#define SYS_INSTALLABLE_FONT_DIR "FONT"
#define SYS_INSTALLABLE_PREF_DIR "PREFS"
#define SYS_INSTALLABLE_HWX_PROT_DIR "HWXPROT"
#define SYS_INSTALLABLE_GESTURE_DIR "GESTURE"
#define SYS_INSTALLABLE_P_DICT_DIR "PDICTIONARY"
#define SYS_INSTALLABLE_APP_DIR "APP"
#define SYS_INSTALLABLE_SERVICE_DIR "SERVICE"
#define SYS_BOOT_DIR "BOOT"
#define SYS_QUICK_INSTALL "QINSTALL"
#define SYS_RUNTIME_ROOT_DIR "SYS"

Filesystem locations off the runtime root.

#define SYS_SYS_APP_FILE "SYSAPP.INI"
#define SYS_APP_FILE "APP.INI"
#define SYS_SYS_SERVICE_FILE "SYSSERV.INI"
#define SYS_SERVICE_FILE "SERVICE.INI"
#define SYS_COPY_FILE "SYSCOPY.INI"
#define SYS_RES_FILE "PENPOINT.RES"
#define SYS_MIL_RES_FILE "MIL.RES"
#define SYS_LIVE_ROOT "Bookshelf"
#define SYS_LOADER_DIR "LOADER"

Default initial app (in penpoint\boot\app).

#define SYS_DEFAULT_INITIAL_APP "Bookshelf"

Boot type.

typedef enum SYS_BOOT_TYPE {
    sysWarmBoot = 1,
    sysColdBoot = 2
} SYS_BOOT_TYPE, *P_SYS_BOOT_TYPE;

Boot progress.

typedef enum SYS_BOOT_PROGRESS {
    sysKernelComplete = 1,
    sysSystemDllsComplete = 2,
    sysSystemAppsInstalled = 3,
    sysInitialAppInstalled = 4,
    sysBookshelfItemsCreated = 5,
    sysServicesInstalled = 6,
    sysAppsInstalled = 7,
    sysInitialAppRunning = 8,
    sysBootComplete = 9
} SYS_BOOT_PROGRESS, *P_SYS_BOOT_PROGRESS;
Boot state.

typedef struct SYS_BOOT_STATE {
    BOOLEAN    booted; // Has booting totally completed?
    SYS_BOOT_PROGRESS progress; // Where are we in the boot cycle?
    SYS_BOOT_TYPE type; // Boot type; warm or cold.
    CLASS initialAppClass; // Class of the initial app.
} SYS_BOOT_STATE, *P_SYS_BOOT_STATE;

Messages

msgNew

Used by PenPoint to create well-known uid theSystem.

Takes P_SYS_NEW, returns STATUS. Category: class message.

Arguments
typedef struct SYS_NEW_ONLY {
    U32 unused1;
    U32 unused2;
    U32 unused3;
    U32 unused4;
} SYS_NEW_ONLY, *P_SYS_NEW_ONLY;

#define systemNewFields
    objectNewFields
    SYS_NEW_ONLY system;

typedef struct SYS_NEW {
    systemNewFields
} SYS_NEW, *P_SYS_NEW;

Comments

This message should never be called by anybody else.

msgSysGetBootState

What stage of booting is the system in?

Takes P_SYS_BOOT_STATE, returns STATUS.

#define msgSysGetBootState
    MakeMsg(clsSystem, 1)

typedef struct SYS_BOOT_STATE {
    BOOLEAN    booted; // Has booting totally completed?
    SYS_BOOT_PROGRESS progress; // Where are we in the boot cycle?
    SYS_BOOT_TYPE type; // Boot type; warm or cold.
    CLASS initialAppClass; // Class of the initial app.
} SYS_BOOT_STATE, *P_SYS_BOOT_STATE;

Comments

This message allows callers to determine the current state of system booting.

See Also

msgSysBootStateChanged Observer message sent at each stage.

msgSysGetRuntimeRoot

Returns a dir handle onto the root of the Penpoint runtime area.

Takes P_OBJECT, returns STATUS.

#define msgSysGetRuntimeRoot
    MakeMsg(clsSystem, 2)

Comments

Penpoint maintains all of its runtime information in one area of the filesystem on the "selected" volume (theSelectedVolume). This message returns a directory handle onto the root of this area.

NOTE: Caller must free the handle when finished.
msgSysGetLiveRoot

Returns an appDir handle onto the root of a volume's live document area.

Takes P_SYS_GET_LIVE_ROOT, returns STATUS.

```
#define msgSysGetLiveRoot MakeMsg(clsSystem, 3)
```

Arguments
typedef struct SYS_GET_LIVE_ROOT {
  OBJECT volHandle;  // Handle onto volume in question.
  OBJECT liveRoot;  // Out: appDir handle to live root on
                     // the volume.
} SYS_GET_LIVE_ROOT, *P_SYS_GET_LIVE_ROOT;

Comments
Live Penpoint documents (those that can be activated) are stored within the live area of a volume. This message returns the root of the live area for a given volume.

pArgs->volHandle is a filesystem handle onto the volume in question. This handle can be on any location of the volume. You can also use the root directory handle for a volume. Use theSelectedVolume if you want to get the live area within the filesystem that Penpoint stores its on-machine documents in.

NOTE: Caller must free the pArgs->liveHandle when finished.

Return Value
stsFSNodeNotFound  No live root on this volume.

msgSysIsHandleLive

Determines if a filesystem handle is within the live document area.

Takes P_SYS_IS_HANDLE_LIVE, returns STATUS.

```
#define msgSysIsHandleLive MakeMsg(clsSystem, 4)
```

Arguments
typedef struct SYS_IS_HANDLE_LIVE {
  OBJECT handle;  // Handle onto the node in question.
  BOOLEAN live;  // Out: Is it in the live area?
} SYS_IS_HANDLE_LIVE, *P_SYS_IS_HANDLE_LIVE;

Comments
Penpoint maintains live documents within a particular point in the directory hierarchy of each volume. This message determines whether a filesystem handle is within the live area of its volume.

Return Value
stsFSNodeNotFound  No live root on the handle's volume.

msgSysCreateLiveRoot

Create a new live root on a volume.

Takes P_SYS_CREATE_LIVE_ROOT, returns STATUS.

```
#define msgSysCreateLiveRoot MakeMsg(clsSystem, 5)
```

Arguments
typedef struct SYS_CREATE_LIVE_ROOT {
  OBJECT volHandle;  // Handle onto volume in question.
  CLASS rootClass;  // Class of app which should run on the
                    // live root directory.
} SYS_CREATE_LIVE_ROOT, *P_SYS_CREATE_LIVE_ROOT;

Comments
Penpoint maintains live documents within a particular point in the directory hierarchy of each volume. This message creates a new live root on a volume if one doesn't already exist. If the live root already exists it creates an instance of the app over whatever is there currently. Use msgSysGetLiveRoot if you want to check for an existing live root.
**msgSysGetVersion**

Returns the system version number.

Takes P_U16, returns STATUS.

```c
#define msgSysGetVersion MakeMsg(clsSystem, 6)
```

Comments

This message allows callers to determine the current PenPoint system version number.

**msgSysGetSecurityObject**

Gets the current security object.

Takes P_OBJECT, returns STATUS.

```c
#define msgSysGetSecurityObject MakeMsg(clsSystem, 31)
```

Comments

Returns objNull if there is no current security object.

**msgSysSetSecurityObject**

Sets the current security object.

Takes P_SYS_SET_SECURITY_OBJECT, returns STATUS.

```c
#define msgSysSetSecurityObject MakeMsg(clsSystem, 32)
```

Arguments

```c
typedef struct SYS_SET_SECURITY_OBJECT {
  OBJECT securityObject; // New security object.
  OBJ_KEY oldKey; // Object key for old security object.
} SYS_SET_SECURITY_OBJECT, *P_SYS_SET_SECURITY_OBJECT;
```

Comments

If a security object already exists then it is destroyed, using the key specified in the arguments. If it refuses to be destroyed then the new security object will not be set.

The security object will be sent `msgSysPowerOn` and `msgSysPowerOff` when the power goes on and off. At shutdown, `msgSysPowerOff` is sent to the security object after `msgSysPowerOff` is sent to power button observers and after `msgAppSave` is sent to applications. At power up, `msgSysPowerOn` is sent to the security object before `msgSysPowerOn` is sent to power button observers.

`msgSysPowerOn` and `msgSysPowerOff` are sent when the machine is suspended/resumed, or shutdown and swap-booted. However, these messages are not sent when a warm-boot occurs. A warm-boot destroys all processes and objects. The service or application that owns the security object will be restarted in the warm-boot case. Security objects must handle the warm-boot case. For example, if the security object is created by the app monitor, the app monitor will receive `msgApplnit` when the application is first installed and `msgRestore` on all warm-boots.

At power down, anything painted on the screen by the security object will not appear immediately, but will appear on the screen when it is restored at power on time. If the security object wishes to display a window on top of all other windows, it should observe `theSystem` for `msgBootStateChanged` to determine when booting is complete.

At power on, the security object may choose to veto the powering on of the system by sending `msgPMSetPowerState` to `thePowerMgr` to turn off power.

Return Value

```c
stsProtectionViolation old security object refused to be destroyed.
```
**msgSysGetCorrectiveServiceLevel**

Gets the corrective service level.

Takes P_STRING, returns STATUS.

```c
#define msgSysGetCorrectiveServiceLevel MakeMsg(clsSystem, 33)
```

Comments

The corrective service level is a string of up to `maxNameLength` characters.

---

**msgSysSetCorrectiveServiceLevel**

Sets the corrective service level.

Takes P_STRING, returns STATUS.

```c
#define msgSysSetCorrectiveServiceLevel MakeMsg(clsSystem, 34)
```

Comments

The corrective service level is a string of up to `maxNameLength` characters.

---

**Notification Messages**

**msgSysBootStateChanged**

The system has reached another stage of booting.

Takes P_SYS_BOOT_STATE, returns STATUS. Category: observer notification.

```c
#define msgSysBootStateChanged MakeMsg(clsSystem, 10)
```

Message Arguments

```c
typedef struct SYS_BOOT_STATE {
    BOOLEAN  booted;  // Has booting totally completed?
    SYS_BOOT_PROGRESS progress;  // Where are we in the boot cycle?
    SYS_BOOT_TYPE type;  // Boot type; warm or cold.
    CLASS initialAppClass;  // Class of the initial app.
} SYS_BOOT_STATE, *P_SYS_BOOT_STATE;
```

Comments

This message is sent to all observers of the System whenever another stage of booting is attained. If you are just interested in whether the system has completed booting or not, look at the `pArgs->booted` boolean.
Part 13 / Writing PenPoint Services
This file contains the API definition for clsHWXEngineService.

clsHWXEngineService inherits from clsService.

Provides default behavior for handwriting engine services.

```c
#ifndef HWXSERV_INCLUDED
#define HWXSERV_INCLUDED
#ifndef SERVICE_INCLUDED
#include <service.h>
#endif
#endif
```

## Messages

### msgNew

Creates a new service object.

Takes P_HWX_SVC_NEW, returns STATUS. Category: class message.

```c
typedef struct HWX_SVC_NEW_ONLY {
    U32 unused1;
    U32 unused2;
    U32 unused3;
    U32 unused4;
} HWX_SVC_NEW_ONLY, *P_HWX_SVC_NEW_ONLY;
#define hwxServiceNewFields 
    serviceNewFields \ 
    HWX_SVC_NEW_ONLY hwxService;

typedef struct HWX_SVC_NEW {
    hwxServiceNewFields 
} HWX_SVC_NEW, *P_HWX_SVC_NEW;
```

### msgHWXSvcCurrentChanged

The current handwriting prototype set has changed.

Takes P_HWX_SVC_CURRENT_CHANGED, returns STATUS.

```c
#define msgHWXSvcCurrentChanged MakeMsg(clsHWXEngineService, 1)

typedef struct HWX_SVC_CURRENT_CHANGED {
    OBJECT newHandle;
    OBJECT oldHandle;
} HWX_SVC_CURRENT_CHANGED, *P_HWX_SVC_CURRENT_CHANGED;
```

The user has switched to or from a handwriting prototype set that uses this engine. See hwxmgr.h and instlmgr.h for details on handwriting prototype set management.

pArgs->newHandle and pArgs->oldHandle provide the handles of the new and old prototype sets. objNull means that the new/former prototype set used some other engine.
This file contains the API definition for clsMILService. The functions described in this file are contained in milserv.lib.

clsMILService inherits from clsService.

Provides default behavior for MIL services.

MIL services are PenPoint device drivers. They represent a MIL device, which represents a piece of hardware. A MIL service sits between a MIL device and the rest of PenPoint.

A MIL service is typically composed of a Ring 0 part, which interfaces to the MIL, and a Ring 3 part, which interfaces to the rest of PenPoint.

MIL service instances are created automatically by PenPoint. Never send msgNew to a MIL Service class yourself! Each MIL device contains a deviceId, which is the class of the MIL service that should be created for it. PenPoint scans the MIL at power-up time and whenever a MIL service installed, and creates one MIL service for each unit of each device.

The MIL service writer can find out the logical id of the device it represents by self-sending msgMILSvcGetDevice.

A MIL service can install a MIL extension if necessary. The new MIL device is installed into the MIL when the MIL service is installed, and removed from the MIL when the MIL service is deinstalled. Use the InstallMILDevice() function in your DLLMain() to do this.

You must also let the service framework know about a service by sending msgSvcClassInitService to your service class in DLLMain(). Here's an example:

```c
STATUS EXPORTED DLLMain(void)
{
    SVC_INIT_SERVICE initService;
    STATUS s;
    // Initialize classes.
    StsRet(ClsMILServiceInit(), s);
    // Include if it is necessary to install MIL extensions.
    InstallMILDevice(&deviceInfo);
    // Initialize service. This creates MIL service instances.
    memset(initService.spare, 0, sizeof(initService.spare));
    initService.autoCreate = true;
    initService.serviceType = 0;
    initService.initServiceFlags = 0;
    ObjCallRet(msgSvcClassInitService, clsTestService, &initService, s);
    return stsOK;
} // DLLMain
```

See project MILSVC for a template for creating MIL services.

```c
#define MIL_SERVICE_INCLUDED
#ifndef MIL_SERVICE_INCLUDED
#define MIL_SERVICE_INCLUDED
#include <go.h>
#include
#endif
```
```c
#ifndef SERVICE_INCLUDED
#include <service.h>
#endif

#ifndef MIL_INCLUDED
#include <mil.h>
#endif

---

**Common #defines and typedefs**

Did this service install MIL devices?

```c
#define svcMILAttrInstalledDevice FSMakeFix32Attr(clsMILService, 1)
```

The MIL device that this mil service is associated with.

```c
typedef struct MIL_SVC_DEVICE {
    TAG unitResourceTag; // resource tag into mil.res
    UID conflictGroup; // conflict group mil svc is on
    U16 logicalId; // mil device logical id to use
    U16 unit; // mil device unit number to use
    U8 reserved[12];
} MIL_SVC_DEVICE, *P_MIL_SVC_DEVICE;
```

---

**Functions**

### InstallMILDevice

Install a MIL device.

Returns STATUS.

**Function Prototype**

```c
STATUS EXPORTED InstallMILDevice(
    P_MIL_DEVICE_INFO pDeviceInfo, // Installable MIL device info.
    U32 reserved1, // Set this to 0
    U32 reserved2); // Set this to 0
```

**Comments**

This routine should used to install one or more MIL devices. These devices will be automatically deinstalled when the MIL service is deinstalled.

This routine *must* be called in the service’s DLLMain(), after the classes are created but before msgSvcClassInitService is sent.

---

**Class Messages**

### msgNew

Creates a new MIL service object.

Takes P_MIL_SVC_NEW, returns STATUS. Category: class message.

**Arguments**

```c
typedef struct MIL_SVC_NEW_ONLY {
    MIL_SVC_DEVICE device;
    U32 unused1;
    U32 unused2;
    U32 unused3;
    U32 unused4;
} MIL_SVC_NEW_ONLY, *P_MIL_SVC_NEW_ONLY;
```

# define milServiceNewFields
    serviceNewFields \
    MIL_SVC_NEW_ONLY milSvc;
```
typedef struct MIL_SVC_NEW {
    milServiceNewFields
} MIL_SVC_NEW, *P_MIL_SVC_NEW;

This message should never be sent by clients. PenPoint automatically creates all MIL service instances by scanning the MIL.

**msgNewDefaults**

Initializes the MIL_SVC_NEW structure to default values.

Takes P_MIL_SVC_NEW, returns STATUS. Category: class message.

```c
typedef struct MIL_SVC_NEW {
    milServiceNewFields
} MIL_SVC_NEW, *P_MIL_SVC_NEW;
```

Sets

```c
pArgs->svc.style.exclusiveOpen = true;
```

```c
pArgs->svc.style.checkOwner = true;
```

Note pArgs->svc.style.connectStyle will be set automatically to reflect underlying MIL device's auto-detection facilities. It will be set to svcAutoDetect if milDevFlagDetachable is true,

```c
svcNoAutoDetect if milDevFlagDetachable is false.
```

Note pArgs->milSvc.device will be set automatically from the MIL.

**msgSvcSetConnected**

Sets connection state of self.

Takes P_SVC_GET_SET_CONNECTED, returns STATUS.

'P_SVC_GET_SET_CONNECTED' structure is defined in service.h.

This message is self-sent whenever a MIL service thinks that it's connection state has changed. This message should be sent even when a mil service isn't sure if it is connected (due to possible interference from other mil services in its conflict group).

If the mil service isn't in a conflict group then the message is sent to ancestor. If it is in a conflict group then the following will occur:

```c
if (pArgs->connected == true) {
```

1. **msgCGPollConnected** is sent to the conflict group manager.

2. The conflict group manager sends **msgMILSvcAreYouConnected** to all services in the conflict group (including the one that self-sent **msgSvcSetConnected**).

3. The conflict group manager decides which service really should be connected and sends **msgMILSvcConnectionStateResolved** to all services. This tells which service (if any) has been chosen to be the connected one. MIL services should restart their connection detection logic if nobody is currently connected.

4. Default behavior for **msgMILSvcConnectionStateResolved** is to send **msgSvcSetConnected** to ancestor if a change of state is indicated. MIL services must *always* send **msgMILSvcConnectionStateResolved** to ancestor.
586  
PENPOINT API REFERENCE  
Part 13 / Writing PenPoint Services

```c
} else {

1. msgSvcSetConnected is sent to ancestor.
2. msgCGInformDisconnected is sent to the conflict group manager.
3. The conflict group manager sends msgMILSvcConnectionStateResolved to all mil services except the
   mil service that sent the msgSvcSetConnected message. MIL services should restart their connection
detection logic.
}
```

See Also  
msgSMConnectedChanged (servmgr.h)

### clsMILService Functionality Available to Subclasses

#### msgMILSvcGetDevice

Returns MIL device associated with this service.

Takes `P_MIL_SVC_DEVICE`, returns STATUS.

```c
#define msgMILSvcGetDevice
MakeMsg(clsMILService, 1)
```

**Message Arguments**

- `TAG` unitResourceTag;  // resource tag into mil.res
- `UID` conflictGroup;  // conflict group mil svc is on
- `U16` logicalId;  // mil device logical id to use
- `U16` unit;  // mil device unit number to use
- `U8` reserved[12];

```c
typedef struct MIL_SVC_DEVICE {
  TAG unitResourceTag;
  UID conflictGroup;
  U16 logicalId;
  U16 unit;
  U8 reserved[12];
} MIL_SVC_DEVICE, *P_MIL_SVC_DEVICE;
```

#### msgMILSvcSetDevice

Sets MIL device associated with this service.

Takes `P_MIL_SVC_DEVICE`, returns STATUS.

```c
#define msgMILSvcSetDevice
MakeMsg(clsMILService, 2)
```

**Message Arguments**

- `TAG` unitResourceTag;  // resource tag into mil.res
- `UID` conflictGroup;  // conflict group mil svc is on
- `U16` logicalId;  // mil device logical id to use
- `U16` unit;  // mil device unit number to use
- `U8` reserved[12];

```c
typedef struct MIL_SVC_DEVICE {
  TAG unitResourceTag;
  UID conflictGroup;
  U16 logicalId;
  U16 unit;
  U8 reserved[12];
} MIL_SVC_DEVICE, *P_MIL_SVC_DEVICE;
```

**Comments**

Note: This message is almost never used. Usually a MIL service is associated with the device that is set at
`msgNew` time, and never changed. This message is included for completeness and very special
circumstances.

#### msgMILSvcInstalledMILDevice

Is this MIL service targeting an installed MIL device?

Takes `pNull`, returns STATUS.

```c
#define msgMILSvcInstalledMILDevice
MakeMsg(clsMILService, 3)
```

**Comments**

Returns `stsOK` if it is, `stsFailed` if it is not.
msgMILSvcAddToConflictManager
Add this service instance to a conflict group manager.
Takes P_MIL_SVC_ADD_TO_CONFLICT_MANAGER, returns STATUS.

#define msgMILSvcAddToConflictManager MakeMsg(clsMILService, 8)

typedef struct MIL_SVC_ADD_TO_CONFLICT_MANAGER {
    OBJECT manager;
} MIL_SVC_ADD_TO_CONFLICT_MANAGER, *P_MIL_SVC_ADD_TO_CONFLICT_MANAGER;

Comments
This message is used to add a MIL service to a conflict group manager.

---

Descendant Responsibility Messages

msgMILSvcPowerOff
The power is about to be turned off.
Takes pNull, returns STATUS.

#define msgMILSvcPowerOff MakeMsg(clsMILService, 4)

Comments
This message is sent after all other power off messages are sent. MIL services must *not* observe the power button to get power notification.
MIL services should save any hardware-specific state that must be restored when the power is applied.

msgMILSvcPowerOn
The power has just come on.
Takes pNull, returns STATUS.

#define msgMILSvcPowerOn MakeMsg(clsMILService, 5)

Comments
This message is sent before all other power on messages are sent. MIL services must *not* observe the power button to get power notification.
MIL services should restore any hardware-specific state that was saved when the power was disconnected.

msgMILSvcAreYouConnected
Do you think you are connected?
Takes P_MIL_SVC_ARE_YOU_CONNECTED, returns STATUS.

#define msgMILSvcAreYouConnected MakeMsg(clsMILService, 6)

Arguments
Enum16 (MIL_SVC_ARE_YOU_CONNECTED) {
    msYes = 0,
    msMaybe = 1,
    msNo = 2
};

Comments
This message is sent to all members of a conflict group whenever any service thinks it has become connected. It allows all members of the conflict group to participate in deciding who is really connected.
Default superclass behavior is to return msMaybe.
msgMILSvcConnectionStateResolved

Tells a MIL service who was chosen to be connected.
Takes UI6, returns STATUS.

```c
#define msgMILSvcConnectionStateResolved MakeMsg(clsMILService, 7)
```

**Comments**
The `pArgs` is the logical id of the service that was chosen to be connected. It is set to `maxUI6` if nobody is connected.
Default superclass behavior is to send `msgSvcSetConnected` to ancestor if a change of state is indicated.
MIL services must always send `msgMILSvcConnectionStateResolved` to ancestor.

msgMILSvcStartConnectionProcessing

It is ok to start connection processing.
Takes `pNull`, returns STATUS.

```c
#define msgMILSvcStartConnectionProcessing \
    MsgNoError(MakeMsg(clsMILService, 9))
```

**Comments**
This message is sent after booting is complete. MIL services should not start their connection processing until they receive this message.
This file contains the API definition for clsMILConflictGroupMgr.

clsMILConflictGroupMgr inherits from clsServiceMgr.

Provides definition of conflict group managers.

A conflict group manager is automatically created for each conflict group in the MIL when one or more MIL service instances are created for the MIL devices which are part of that conflict group. The uid of the conflict group manager is that of the conflict group itself. In other words, if there is a conflict group identified with the tag theMILConflictGroup4, then the conflict group manager will have a well-known uid of MILConflictGroup4.

A conflict group manager is very much like a service manager. All of the MIL service instances that represent devices in the conflict group are on the conflict group manager. Each service instance is also made an observer of the conflict group manager.

The conflict group manager keeps track of which MIL service owns the conflict group. The owning service is the only one that is permitted to actually use one of the devices in the conflict group.

```c
#ifndef SERVCONF_INCLUDED
#define SERVCONF_INCLUDED
#endif

#include <servmgr.h>
#endif
```

### Messages

**msgNew**

Creates a new conflict group manager.

Takes P_SM_NEW, returns STATUS. Category: class message.

**Comments**

This message should *never* be called by clients. Conflict group managers are automatically created. The new args must always be the same as for a service manager.

```c
define msgNew
MakeMsg(clsMILConflictGroupMgr, 1)
```

**msgCGGetOwner**

Gets the current owner of the conflict group.

Takes P.CG_GET_OWNER, returns STATUS.

```c
define msgCGGetOwner
typedef struct CG_GET_OWNER {
    OBJECT owner;
    U8 reserved[16];
} CG_GET_OWNER, *P.CG_GET_OWNER;
```

**Arguments**

If no one owns the conflict group, 'objNull' will be returned in the owner field.
msgCGSetOwner

Sets a new conflict group owner.

 Takes P_CG_SET_OWNER, returns STATUS.

```c
#define msgCGSetOwner MakeMsg(clsMILConflictGroupMgr, 2)
```

**Arguments**

```c
typedef struct CG_SET_OWNER {
    OBJECT owner; // New owner.
} CG_SET_OWNER, *P_CG_SET_OWNER;
```

**Comments**

"owner" can be objNull to specify that this conflict group has no owner.

Old and new owners will receive service messages which allow them to veto the ownership change and informs them that the change has taken effect. The message sequence is as follows:

1. msgSvcOwnerAquireRequested is sent to the new owner. `pArgs->ownedService` is set to the conflict group. The new owner can veto the owner change by returning a status of anything other than `stsOK` or `stsNotUnderstood`. **msgCGSetOwner** returns with the abort status.

2. msgSvcOwnerReleaseRequested is sent to the old owner. `pArgs->ownedService` is set to the conflict group. The old owner can can veto the owner change by returning a status of anything other than `stsOK` or `stsNotUnderstood`. **msgCGSetOwner** returns with the abort status.

3. msgSvcOwnerReleased is sent to the old owner.

4. msgSvcOwnerAquired is sent to the new owner.

5. msgCGOwnerChanged is sent to all observers of this conflict group manager, including all of the service instances on this manager.

**Return Value**

- **stsBadObject** New owner is not an object.
- **stsBadAncestor** New owner has invalid ancestor.

**See Also**

service.h, for definition of msgSvc... messages.

msgCGPollConnected

Polls all the services in the conflict group to see who is connected.

 Takes pNull, returns STATUS.

```c
#define msgCGPollConnected MakeMsg(clsMILConflictGroupMgr, 3)
```

**Comments**

A conflict group manager receives this message when any service within the conflict group thinks it might be connected. The conflict group manager sends **msgMILSvcAreYouConnected** to each service. It then sends **msgSvcConnectionStateResolved** to each service, choosing one of the services as the connected one.

msgCGInformDisconnected

Tells all the services in the conflict group that a disconnect happened.

 Takes pNull, returns STATUS.

```c
#define msgCGInformDisconnected MakeMsg(clsMILConflictGroupMgr, 4)
```

**Comments**

A conflict group manager receives this message when the connected service within the conflict group decides it is disconnected. The conflict group manager sends **msgSvcConnectionStateResolved** to each service, specifying that nobody is connected.
**Notification Messages**

**msgCGOwnerChanged**

A conflict group's owner has changed.

Takes `P_CG_OWNER_NOTIFY`, returns `STATUS`. Category: observer notification.

```
#define msgCGOwnerChanged
MakeMsg(clsMILConflictGroupMgr, 10)
```

**Arguments**

typedef struct CG_OWNER_NOTIFY {
    OBJECT conflictGroup; // conflict group whose owner changed.
    OBJECT oldOwner;      // old owner.
    OBJECT owner;         // new owner.
} CG_OWNER_NOTIFY, *P_CG_OWNER_NOTIFY;

**Tags**

```
#define tagConflictChoice
MakeTag(clsMILConflictGroupMgr, 1)
```
This file contains the API definition for clsService.

clsService inherits from clsStream.

Provides default behavior for services.

Introduction

All non-application functionality under Penpoint is expressed as a service. If what you want to do does not fit the application model (documents created via Stationery or Accessories, subclass of clsApp, etc) then it should be a service. Some examples of services are: device drivers, inbox/outbox transfer agents such as fax and e-mail, network protocol stacks, and device drivers.

Service instances are automatically organized onto service managers. A service manager represents a category of service, such as Printers or Serial Devices. All of the service instances in a given category are can be used interchangeably; that is, they all support the API that is required to be in that category.

Clients access service instances via service managers. See servmgr.h for details.

Each service instance has a text name, which is how it is uniquely identified. Clients use this name to identify a service instance on a service manager. A service instance's name is specified at msgNew time. Names must be unique for all services on the same service manager, and all services of the same class.

There are two exclusivity models for services: services that require exclusive access by a single client, and services that allow multiple clients simultaneous access. Services provides default behavior for arbitrating ownership of exclusive access services.

Multiple access services can either be shared (each client gets back the uid of the service when they open the service) or multi-user (each client gets back a different object when they open the service).

Service instances can optionally maintain state. By default each service instance has a node in the filesystem. clsService will automatically recreate service instances from their state files when PenPoint is rebooted. Also, service instances can be saved and restored from external disks by moving their state nodes on and off the machine.

A service instance can have an optional "target". A target is some other service instance. If a service has a target, the service superclass takes care of remembering what the target points at. Typically, data flows from one service instance to next, going down the target chain. Control information, such as when a physical device is becomes connected, flows up the target chain.

A service is implemented as an installable DLL. Service instances are either created in the DLLMain() of the service DLL, created dynamically after the service has been installed, or created from pre-configured instance state nodes when the service is first installed. See servmgr.h for a description of how services are installed and deinstalled, and how a service is organized on disk.
Writing A Basic Service

A minimal service that does not save state or use a target must handle just one superclass message: msgNewDefaults. There are four fields which need to be filled in:

\[\begin{align*}
\text{pArgs->svc.style.exclusiveOpen} & \quad \text{Is this an exclusive access service?} \\
\text{svc.style.openClass} & \quad \text{Is this a multi-user service?} \\
\text{svc.pManagerList} & \quad \text{List of service managers to add to.} \\
\text{svc.numManagers} & \quad \text{Number of managers on the list.}
\end{align*}\]

Project BASICSVC is a template for a minimal service. Use it as a guide.

Writing a Service That Saves State

clsService maintains an open handle on a service's state node. By default the state node type is a file and the open handle is an instance of clsFileHandle. Both of these things can be overridden in your msgNewDefaults handler.

Services must decide for themselves when they need to update their state node. They should always maintain enough state to be able to survive a reboot. There is no explicit Save/Restore messages for services; A SERVICE MUST UPDATE ITS STATE NODE WHENEVER ITS STATE CHANGES.

When its time to save state, self-send msgSvcGetHandle to get your state node handle. Self-send msgSvcSetModified when you complete updating state. These are the only messages that you will need to use for this type of service.

Service instances will be automatically recreated when a warm boot occurs. The msgNew arguments to clsService include the locator of the state node. Service instances must check to see if this node is non-empty then a warm boot is happening, and the service must recreate itself from the state node.

State nodes can be copied out to disk, then reloaded the next time the service class is installed, or reloaded one at time. clsService will automatically create a service instance for each state node at this time using the same mechanism as warm boot recovery. There is no difference between warm boot recovery and creation from a pre-configured state node copied in at installation time, as far as the service is concerned.

Writing a Service That Has A Target

Services can also bind and open other services. In fact, this is such a common situation that clsService provides lots of support for this. Each service can have a target, which refers to some other service. When the service is first created the default behavior is to attempt to bind to the target. clsService will automatically open the target when the service is opened if the autoOpen style bit is true.

A service becomes a client of its target. All client observer notifications and ownership messages from a service's target are sent to the service.

A service's target is usually set at msgNew time, and can be changed anytime after with msgSvcSetTarget. msgSvcGetTarget gets a service instance's target.

Typically a service will open its target when a client opens it, using msgSvcOpenTarget. msgSvcCloseTarget should be used to close the target.

Services also support the notion of being connected. Most hardware services can detect whether their hardware is connected or disconnected. Each service has a state bit which says whether it is connected or not. When the hardware changes connection state the service sends msgSVCSetConnected to itself, which notifies everyone who is bound to that service.
Non-hardware services automatically change their connection state when their targets change connection state. Thus, connection state propagates up from the hardware to all services that are bound to that hardware.

A hardware service for a device that cannot auto-detect connection is always in the connected state. Project TESTSVC provides a template for a service that deals with a target.

Advanced Features

Services that can provide both global and service instance option cards. A global option sheet sets configuration information for the entire service. It is invoked when the user calls for options of a service from the Service card of the Installer. Services can add additional cards to the global option sheet.

Service instance option sheets allow the user to set the configuration of particular instance. For example, the serial service provides a card which allows the user to set baud rate, parity, etc. Services should update their state node when the user applies a change to the option sheet. There is no default service instance option card.

Services should respond to the standard option sheet protocol (msgOptionAddCards, msgOptionRefreshCard, etc) if they wish to provide option cards. See option.h for details. The option sheet messages are either sent as class messages for global options or normal instance messages for instance options.

A service's configuration information can also be queried and set programmatically via msgSvcGetMetrics and msgSvcSetMetrics. A service must be able to respond to these messages at any time, and should update its state node when its metrics are changed. The Get/SetMetrics messages are generic; they allow a client to save and restore metrics independently of the size or contents of the metrics. This allows a client to have absolutely no knowledge of the internals of a service. The client can ask the user to set configuration options, then save and restore these configuration options via the generic Get/Set messages.

Service instances can have icons associated with them in the same fashion as documents. Create icons using tagAppIconBitmap and tagAppSmallIconBitmap and put them in the service resource file. This is done in the same manner as applications.

Services and Tasking

A service, just like any other object, is owned by some task. However, all services must be callable from outside the owning task (objCapCall is always true for service instances). Service authors must take this into account. Services must either use explicitly-created global heaps or instance data; never store data in a local heap or the shared process heap.

If the service is not exclusive access or multi-user, anyone who has the service open can call the service at anytime, even while someone else is in the middle of another call. Use semaphores to protect access where appropriate.

You must also make sure that the a service's owning task will remain active for the real lifetime of the service. For instance, if a service is created via some transient user interface task such as a document or a tool, then the service instance will become invalid when that tool is shut down.

An alternative to keeping the creating task around for the lifetime of the service instance is to use msgObjectNew to create the service instance under another task. A very good task to create instances under is the main task of the service. The service resource file handle is available for use with msgObjectNew. Use msgSvcGetClassMetrics to get this handle (metrics.resFile). Send msgObjectNew
to this handle. Note that msgObjectNew must be sent, not called. Remember, any pointers in the msgObjectNew pArgs must be in global memory.

### Recovering From Unexpected Client Termination

Service instances automatically detect if a client terminates unexpectedly; that is, if a client terminates while it is bound to the service instance or owns it. msgSvcClientDestroyedEarly is sent to the service instance when this condition is detected. Subclasses that maintain per-client state can handle this message and perform cleanup. By default the service is closed and unbound from the terminating object.

### Sample DLLMain Routine

You must let the service framework know about a service by sending msgSvcClassInitService to your service class. Here’s an example:

```c
STATUS EXPORTED DLLMain(void)
{
    SVC_INIT_SERVICE initService;
    STATUS s;
    StsRet(ClsTestServiceInit(), s);
    memset(initService.spare, 0, sizeof(initService.spare));
    initService.autoCreate = true;
    initService.serviceType = 0;
    initService.initServiceFlags = 0;
    ObjCallRet(msgSvcClassInitService, clsTestService, &initService, s);
    return stsOK;
}
```

```c
#ifndef SERVICE INCLUDED
#define SERVICE INCLUDED
#endif

#ifndef STREAM INCLUDED
#include <stream.h>
#endif
#ifndef FS INCLUDED
#include <fs.h>
#endif
```

### Common #defines and typedefs

### Service Status Codes

An exclusive-open service is already open by someone else (msgSvcOpenRequested), or a service’s target is already open (msgSvcOpenTarget).

```c
#define stsSvcAlreadyOpen MakeStatus(clsService, 1)
```

A service tried to open its target but the target manager field is null.

```c
#define stsSvcNoTarget MakeStatus(clsService, 2)
```

A service tried to open its target but the target service doesn’t exist or the target’s service manager hasn’t shown up yet.

```c
#define stsSvcTargetNotBound MakeStatus(clsService, 3)
```

An autoMsgPassing service tried to pass a message to its target, but the target was not open.

```c
#define stsSvcTargetNotOpen MakeStatus(clsService, 4)
```
An attempt was made to change ownership, `queryLock`, or deinstall an open service.

```c
#define stsSvcInUse MakeStatus(clsService, 5)
```

Someone who wasn't the owner of a `checkOwner` service tried to open it.

```c
#define stsSvcNotOwner MakeStatus(clsService, 6)
```

Someone tried to open or `queryLock` a service that is `queryLocked`.

```c
#define stsSvcLocked MakeStatus(clsService, 7)
```

Problem following the target chain during `msgSvcAutoDetectingHardware`.

```c
#define stsSvcValidConnectStyleNotFound MakeStatus(clsService, 8)
```

A deinstallation is in process. No new clients can be accepted.

```c
#define stsSvcDeinstallInProcess MakeStatus(clsService, 10)
```

A service of this name already exists and refuses to terminate.

```c
#define stsSvcAlreadyExists MakeStatus(clsService, 11)
```

A service was created with style `waitForTarget` set to false and the target wasn't found at `msgNew` or `msgSvcSetTarget` time.

```c
#define stsSvcTargetNotFound MakeStatus(clsService, 12)
```

### Target

A target references another service.

```c
typedef struct SVC_TARGET {
    OBJECT manager;
    U8 pName[nameBufLength];
    U8 spare[12];
} SVC_TARGET, *P_SVC_TARGET;
```

### Service Class Metrics

Passed back by `msgSvcGetClassMetrics`. Also used in `clsServiceInstallMgr` (servimgr.h) and `clsServiceMgr` (servmgr.h).

```c
typedef struct SVC_CLASS_METRICS {
    CLASS serviceClass; // The class of this service.
    U8 pClassName[nameBufLength]; // Service class name.
    U32 type; // See svctypes.h.
    U8 pTypeName[nameBufLength]; // Service type name.
    OS_PROG_HANDLE progHandle; // Service dll program handle.
    U32 initServiceFlags; // As specified in
    // msgSvcClassInitService.
    OBJECT resFile; // Handle to service res file.
    // Can be objNull if not
    // full environment and
    // service.res is empty.
    OBJECT serviceDir; // Dir handle to service global
    // directory.
    OBJECT privateServiceMgr; // Private service mgr, if the
    // svcCreatePrivateServiceMgr
    // flag is set.
    U32 reserved1;
    U32 reserved2;
    U32 reserved3;
    U32 reserved4;
    U32 reserved5;
} SVC_CLASS_METRICS, *P_SVC_CLASS_METRICS;
```
### Auxiliary Messages

See servmisc.h for less commonly used (but important!) service messages

```c
#ifndef SERVMISC_INCLUDED
#include <servmisc.h>
#endif
```

### Creation Messages

#### msgSvcClassInitService

Initializes the service class.

Takes P_SVC_INIT_SERVICE, returns STATUS. Category: class message.

```c
#define msgSvcClassInitService MakeMsg(clsService, 56)
```

**Comments**

You must send this message to the service class immediately after it has been created.

#### initServiceFlags

Don’t show this service in the installer. User can’t configure or deinstall the service if this flag is set

```c
#define svcNoShow ((U32) flag0)
```

Automatically pop up the global service option card the first time this service is installed.

```c
#define svcPopupOptions ((U32) flag1)
```

Don’t copy in the state files from the INST directory when the service is installed.

```c
#define svcNoLoadInstances ((U32) flag2)
```

Create a private service manager for instances of this class. All instances of this class will automatically be added to the private service manager. See SVC_CLASS_METRICS for uid of the private service manager.

```c
#define svcCreatePrivateServiceMgr ((U32) flag3)
```

Generate a complete process environment in the DLLMain() process. Right now this means creating the ProcessResList. Also, a service resource file handle will be created even if the service resource file is empty. Note that a complete process environment takes up significant memory. Only turn this on if you need it.

```c
#define svcFullEnvironment ((U32) flag4)
```

#### msgNew

Creates a new service object.

Takes P_SVC_NEW, returns STATUS. Category: class message.

**Comments**

Callers send msgNew to create a new service instance. The instance will add itself to one or more service managers. Clients should access the service instance via the service manager API after msgNew.
Superclass behavior includes associating the service with its node in the filesystem, adding it to the specified service managers, and attempting to bind to a target service. If style.waitForTarget is false and the target isn't found then stsSvcTargetNotFound is returned.

The following parameters are usually set by the caller of msgNew:

- pServiceName
- target

The following parameters are usually set by the subclass of clsService in msgNewDefaults (after the ancestor call):

- style (including openClass)
- pManagerList
- numManagers

If a subclass wants to change the handleClass, fnNew, or fnNewExtra parameters it must also execute the following in its msgNewDefaults method, after sending msgNewDefaults to ancestor:

```c
pNew->svc.handleClass = myFSHandleClass;
ObjCallOK(msgNewDefaults, pNew->svc.handleClass, &(pNew->svc.fsNew), s);
```

Most services will not need to do this.

If a service with the same name as the new service already exists on any relevant service manager, the old service will be destroyed and the new service will replace it. However, if any of the old services veto the termination then the new service will not be created and an error status (stsSvcAlreadyExists) is returned.

- stsNoMatch: Target not found and style.waitForTarget is false.
- stsSvcAlreadyExists: Service of this name already exists and can't be terminated.
- stsBadParam: Illegal target type.

```c
#define svcAutoDetect 0 // Can auto-detect hardware connect/disconnect.
#define svcNoAutoDetect 1 // Can't do hardware auto-detect.
#define svcFollowTarget 2 // Connect state follows target's connect state.
typedef struct SVC_STYLE {
    U16 waitForTarget : 1, // OK if target doesn't exist; wait for it
                         // to show up.
    exclusiveOpen : 1, // Allow only one open or QueryLock at a time.
    reserved1 : 1, // Reserved.
    autoOwnTarget : 1, // Set this service to be the owner of its
                       // target when it receives
                       // msgSvcChangeOwnerRequested.
    autoOpen : 1, // Open/close our target when we are
                  // opened/closed.
    autoMsgPass : 1, // Forward all messages that are not
                     // clsObject, clsService or clsOption
                     // messages to target.
    checkOwner : 1, // Only allow the owner to open us;
                    // return stsNotOwner if opener is wrong.
    autoOption : 1, // Forward all option sheet messages to
                    // target. If the target is exclusive open
                    // and checkOwner, then only forward if
                    // target is owned by this service instance.
    connectStyle : 2, // Connect detect abilities.
    reserved2 : 6; // Reserved.
}```
CLASS openClass; // Class used to create object returned from
// msgSMOpen. Can be objNull to return the
// service instance object itself.

U16 spare1;
U16 spare2;
} SVC_STYLE, *P_SVC_STYLE;
typedef struct SVC_NEW_ONLY {
  SVC_TARGET target; // Initial target. target.manager
  // can be objNull for no target.
  P_STRING pServiceName; // Name of instance.
  SVC_STYLE style; // Overall style.
  CLASS handleClass; // Class of service’s node handle.
  FS_NEW fsNew; // NewArgs for handle, filled in
  // at msgNewDefault time.
  U32 fsNewExtra[25]; // Extra fsNew space.
  P_UID pManagerList; // List of service managers that
  // self should be added to.

  U16 numManagers; // Number of uids in manager list.
  U32 unused1;
  U32 unused2;
  U32 unused3;
  U32 unused4;
} SVC_NEW_ONLY, *P_SVC_NEW_ONLY;
#define serviceNewFields \ 
  serviceNewFields \ 
  SVC_NEW_ONLY svc;
typedef struct SVC_NEW {
  serviceNewFields
} SVC_NEW, *P_SVC_NEW;

msgNewDefaults

Initializes the SVC_NEW structure to default values.

Takes P_SVC_NEW, returns STATUS. Category: class message.

Sets

object.cap |= objCapCall; // Client must not override this in msgNew
svc.target.manager = objNull;
strcpy(pNew->svc.target.pName, "");
svc.pServiceName = pNull;
svc.style.waitForTarget = true;
svc.style.exclusiveOpen = false;
svc.style.autoOwnTarget = true;
svc.style.autoOpen = false;
svc.style.autoMsgPass = false;
svc.style.checkOwner = false;
svc.style.autoOption = false;
svc.style.connectStyle = svcFollowTarget;
svc.style.openClass = objNull;
svc.handleClass = clsFileHandle;
ObjCallOK(msgNewDefaults, pNew->svc.handleClass, \&svc.fsNew, s);
svc.fsNew.fs.exist = fsExistOpen | fsNoExistCreate;
svc.pManagerList = pNull;
svc.numManagers = 0;

State File Messages

msgSvcGetHandle
Returns a handle to the service's state node.
Takes P_OBJECT, returns STATUS.
#define msgSvcGetHandle
MakeMsg(clsService, 12)

Comments
Every service instance has an open handle to its state node. Use this message when you want to update the contents of your state node.
NOTE: This handle must NOT be freed, closed, or changed.

msgSvcGetModified
Gets the modified state of this service.
Takes P_SVC_GET_MODIFIED, returns STATUS.
#define msgSvcGetModified
MakeMsg(clsService, 36)

Arguments
typedef struct SVC_GET_SET_MODIFIED {
    BOOLEAN modified; // modified state
} SVC_GET_SET_MODIFIED, *P_SVC_GET_SET_MODIFIED;

msgSvcSetModified
Sets modified state of self.
Takes P_SVC_GET_SET_MODIFIED, returns STATUS.
#define msgSvcSetModified
MakeMsg(clsService, 20)

Message
Arguments
typedef struct SVC_GET_SET_MODIFIED {
    BOOLEAN modified; // modified state
} SVC_GET_SET_MODIFIED, *P_SVC_GET_SET_MODIFIED;

Comments
Service subclasses must send this message with pArgs->modified set to true whenever they change their state file.
Propogates msgIMModifiedChanged to everyone who has bound to this service and is an observer of all service managers that this service is on.

See Also
msgIMModifiedChanged (instmgr.h)
**msgSvcOpenTarget**

Attain access to the target service for data transfer.

Takes P_SVC_OPEN_CLOSE_TARGET, returns STATUS.

```c
#define msgSvcOpenTarget MakeMsg(clsService, 13)
```

**Arguments**

```c
typedef struct SVC_OPEN_CLOSE_TARGET {
P_ARGS  pArgs;  // Open or close parameters.
} SVC_OPEN_CLOSE_TARGET, *P_SVC_OPEN_CLOSE_TARGET;
```

**Backwards compatibility**

```c
typedef SVC_OPEN_CLOSE_TARGET SVC_OPEN_TARGET, *P_SVC_OPEN_TARGET;
```

This call should be made when the service is ready to actually transfer data to its target. It will cause `msgSMOpen` to be sent to the target's service manager. The target service instance can refuse the subsequent `msgSvcOpenRequested` request if it wants. The target service should be kept open for the minimum time possible.

This message is sent automatically if `newArgs.style.autoOpen` is true. Note that `pArgs` is set to `pNull` in this case.

**Return Value**

- `stsFailed` target.type is not svcTypeService.
- `stsSvcNoTarget` target.manager is null.
- `stsSvcNotBound` service is still waiting to bind to its target.
- `stsSvcAlreadyOpen` target is already open.

**See Also**

- `msgSMOpen` (servmgr.h)

**msgSvcCloseTarget**

Give up data transfer access to the target service.

Takes P_SVC_OPEN_CLOSE_TARGET, returns STATUS.

```c
#define msgSvcCloseTarget MakeMsg(clsService, 14)
```

**Message**

```c
typedef struct SVC_OPEN_CLOSE_TARGET {
P_ARGS  pArgs;  // Open or close parameters.
} SVC_OPEN_CLOSE_TARGET, *P_SVC_OPEN_CLOSE_TARGET;
```

This will cause `msgSMClose` to be sent to the target's service manager, resulting in `msgSVCCloseRequested` being sent to the target.

This message is sent automatically if `newArgs.style.autoOpen` is true. Note that `pArgs` is set to `pNull` in this case.

**Return Value**

- `stsFailed` target.type is not svcTypeService.

**See Also**

- `msgSMClose` (servmgr.h)
**msgSvcGetTarget**

Returns current target.

Takes `P_SVC_GET_TARGET`, returns `STATUS`.

```c
#define msgSvcGetTarget
MakeMsg(clsService, 15)
```

**Arguments**

typedef struct SVC_GET_TARGET {
  SVC_TARGET target; // Out: target
  OBJECT targetHandle; // Out: handle to target, if bound
  OBJECT targetService; // Out: target service, if open
} SVC_GET_TARGET, *P_SVC_GET_TARGET;

**Comments**

target contains the target that was specified at `msgNew` time or by the last `msgSvcSetTarget`.

targetHandle contains the service manager handle onto our target if we have bound with the target, or `objNull` if we haven't yet bound.

targetService is the actual service object if the target has been opened, `objNull` if it isn't open.

**msgSvcSetTarget**

Change our target.

Takes `P_SVC_SET_TARGET`, returns `STATUS`.

```c
#define msgSvcSetTarget
MakeMsg(clsService, 16)
```

**Arguments**

typedef struct SVC_SET_TARGET {
  SVC_TARGET target;
} SVC_SET_TARGET, *P_SVC_SET_TARGET;

**Comments**

Closes the old target (if it is open), unbinds the old target (if it is bound) and attempts to bind with the new target. `style.waitForTarget` specifies whether we will wait for the target to show up if it does not exist.

Causes `msgSvcTargetChanged` to be sent.

**Return Value**

`stsNoMatch` new target doesn't exist and `style.waitForTarget` is false.

**Connection Messages**

**msgSvcGetConnected**

Gets the connected state of this service.

Takes `P_SVC_GET_SET_CONNECTED`, returns `STATUS`.

```c
#define msgSvcGetConnected
MakeMsg(clsService, 19)
```

**Arguments**

typedef struct SVC_GET_SET_CONNECTED {
  BOOLEAN connected; // connect state
} SVC_GET_SET_CONNECTED, *P_SVC_GET_SET_CONNECTED;

**msgSvcSetConnected**

Sets connection state of self.

Takes `P_SVC_GET_SET_CONNECTED`, returns `STATUS`.

```c
#define msgSvcSetConnected
MakeMsg(clsService, 35)
```
typedef struct SVC_GET_SET_CONNECTED {
    BOOLEAN connected; // connect state
} SVC_GET_SET_CONNECTED, *P_SVC_GET_SET_CONNECTED;

This message should only be used by auto-detecting services that interface directly to hardware when they have determined that their connection state has changed.

Propogates msgSMConnectedChanged to everyone who has bound to this service and is an observer of all service managers that this service is on.

If a binding service’s connectStyle is svcFollowTarget, then it’s connected state will mirror that of its target. This will be the case for most services, and is how the connect state propagates up the target links.

See Also
msgSMConnectedChanged (scrmgr.h)

Client Access Messages

msgSvcBindRequested
Client asked to bind to this service.
Takes P_SVC_BIND, returns STATUS.
#define msgSvcBindRequested MakeMsg(clsService, 2)

typedef struct SVC_BIND {
    OBJECT caller; // Object making the request.
    OBJECT manager; // Service manager the request is being made through.
} SVC_BIND, *P_SVC_BIND;

A client sent msgSMBind to a service manager. The service can refuse the request by returning stsFailed. The default superclass behavior is to return stsOK.

The service manager maintains a list of all the objects that have bound to this service instance. The caller is added to this list if this message returns stsOK. This list is available via msgSvcGetBindList.

Subclasses usually let ancestor handle this message. This message must always be passed to ancestor.

msgSvcUnbindRequested
Client asked to unbind from this service.
Takes P_SVC_BIND, returns STATUS.
#define msgSvcUnbindRequested MakeMsg(clsService, 3)

typedef struct SVC_BIND {
    OBJECT caller; // Object making the request.
    OBJECT manager; // Service manager the request is being made through.
} SVC_BIND, *P_SVC_BIND;

A client sent msgSMUnbind to a service manager or a client who was bound to the service was destroyed.

The service cannot veto this request. The caller is removed from the service instance’s bind list before this message is sent.

Subclasses usually let ancestor handle this message. This message must be passed to ancestor.
**msgSvcOpenRequested**

Client asked to open this service.

Takes P_SVC_OPEN_CLOSE, returns STATUS.

```c
#define msgSvcOpenRequested MakeMsg(clsService, 4)
```

**Arguments**

```c
typedef struct SVC_OPEN_CLOSE {
   OBJECT caller;  // Object making the request.
   OBJECT manager; // Service manager the request is
   // being made through.
   P_ARGS pArgs;   // Service-specific open or close
   // parameters.
   OBJECT service; // Out (msgSvcOpen): In (msgSvcClose):
   // uid of open handle or service.
} SVC_OPEN_CLOSE, *P_SVC_OPEN_CLOSE;
```

**Comments**

A client sent msgSMOpen to a service manager. The service instance can refuse the open request by returning stsFailed.

The service manager maintains a list of all the objects that have opened this service instance. The caller is added to this list if this message returns stsOK. This list is available via msgSvcGetOpenList.

The service instance is marked in use when one or more clients have it open. A service that has instances that are in use cannot be deinstalled.

If the style.exclusiveOpen is true then only one client can have the service open at a time. If style.checkOwner is true then the owner of the service is the only one that can open the service. Errors are returned to the client if these conditions aren't true; see servmgr.h for details.

Subclasses usually do some processing, then pass this message to superclass. This message must be passed to ancestor.

**msgSvcOpenDefaultsRequested**

Client wants open pArgs initialized.

Takes P_SVC_OPEN_CLOSE, returns STATUS.

```c
#define msgSvcOpenDefaultsRequested MakeMsg(clsService, 9)
```

**Message**

```c
typedef struct SVC_OPEN_CLOSE {
   OBJECT caller;  // Object making the request.
   OBJECT manager; // Service manager the request is
   // being made through.
   P_ARGS pArgs;   // Service-specific open or close
   // parameters.
   OBJECT service; // Out (msgSvcOpen): In (msgSvcClose):
   // uid of open handle or service.
} SVC_OPEN_CLOSE, *P_SVC_OPEN_CLOSE;
```

**Comments**

A client sent msgSMOpenDefaults to a service manager.

**msgSvcCloseRequested**

Client asked to close this service.

Takes P_SVC_OPEN_CLOSE, returns STATUS.

```c
#define msgSvcCloseRequested MakeMsg(clsService, 5)
```
```c
typedef struct SVC_OPEN_CLOSE {
    OBJECT caller; // Object making the request.
    OBJECT manager; // Service manager the request is being made through.
    P_ARGS pArgs; // Service-specific open or close parameters.
    OBJECT service; // Out (msgSvcOpen): In (msgSvcClose): uid of open handle or service.
} SVC_OPEN_CLOSE, *P_SVC_OPEN_CLOSE;
```

**Comments**

A client has sent msgSMClosed to a service manager or a client who had the service open was destroyed. The service cannot veto this request; it must perform any cleanup required at this time. The caller is removed from the open list before this message is sent.

Subclasses usually do some processing, then pass this message to superclass. This message must be passed to ancestor.

---

**msgSvcQueryLockRequested**

Client asked to QueryLock this service.

Takes pNull, returns STATUS.

```c
#define msgSvcQueryLockRequested MakeMsg(clsService, 6)
```

**Comments**

A client has sent msgSMQueryLock to a service manager. QueryLocking a service lets the client get access to the service without opening it. However, if style.exclusiveOpen is true then the QueryLock counts as an open as far as allowing only one open at a time.

Subclasses usually let ancestor handle this message. This message must be passed to ancestor.

---

**msgSvcQueryUnlockRequested**

Client asked to QueryUnlock this service.

Takes pNull, returns STATUS.

```c
#define msgSvcQueryUnlockRequested MakeMsg(clsService, 7)
```

**Comments**

A client has sent msgSMQueryUnlock to a service manager. This releases a previous QueryLock.

Subclasses usually let ancestor handle this message. This message must be passed to ancestor.

---

**msgSvcCharacteristicsRequested**

Client asked to get characteristics of this service.

Takes P_SVC_CHARACTERISTICS, returns STATUS.

```c
#define msgSvcCharacteristicsRequested MakeMsg(clsService, 54)
```

**Arguments**

```c
typedef struct SVC_CHARACTERISTICS {
    OBJECT handle; // Handle of item to get characteristics of.
    P_UNKNOWN pBuf; // Out through Ptr: Characteristics buffer.
    UI64 len; // In/Out: Buffer size. If 0 then the actual size is returned.
} SVC_CHARACTERISTICS, *P_SVC_CHARACTERISTICS;
```

**Comments**

A client sent msgSMGetCharacteristics to a service manager. The service will return service-specific characteristics via pArgs->pBuf. pArgs->len specifies the maximum size of the client’s buffer. If pArgs->len is 0 then the service should return the actual size of its characteristics in pArgs->len and not pass back any data.
```c
#define tagServiceClassOptionSheet
#define tagServiceFirstTime
// Next message up: 59
// Obsolete, here for backwards compatibility.

Function Prototype STATUS EXPORTED InitService(
    P_STRING pReserved1, // Set this to pNull.
    CLASS serviceClass, // class id.
    BOOLEAN autoCreate, // Create an instance for each state
    // node at install and warm boot times.
    U32 serviceType, // Global service type. See
    // svctypes.h. Usually set to 0.
    U32 initServiceFlags, // Or-in InitService flags.
    U32 reserved2, // Set this to 0
    U32 reserved3); // Set this to 0
```
This file contains the API definition for clsServiceMgr.

clsServiceMgr inherits from clsInstallMgr.

Provides access to a category of PenPoint service instances.

Introduction

A service manager represents a category of services in PenPoint. Service managers have well-known ids so they can be globally accessed. PenPoint creates several service managers by default. They are:

- **theModems**  Modems.
- **thePrinters**  Printers.
- **thePrinterDevices**  Devices that a printer can talk to.
- **theSendableServices**  All services that interface to the Send Manager. See sendserv.h.
- **theTransportHandlers**  Transport level network protocol handlers.
- **theLinkHandlers**  Link level network protocol stacks.
- **theHWXEngines**  Installable handwriting engines.
- **theMILDevices**  All MIL services (device drivers).
- **theParallelDevices**  Parallel port devices.
- **theSerialDevices**  Serial port devices.
- **theHighSpeedPacketHandlers**  High performance packet drivers.
- **theOutboxServices**  All outbox services.
- **theInboxServices**  All inbox services.
- **theDatabases**  All PIA databases.

Additional service managers can be created on the fly by third parties or by GO.

All of the service instances in a given category are on that service manager. All the instances on a service manager support the same API, so they can be used interchangeably.

Each service instance on a service manager is identified with a unique string name. For example, there might be three printers on **thePrinters**: "MyLaserJet", "MarketingPrinter1", and "LittleDotMatrix".

You can find a particular service instance or enumerate all the instances that are available. You can observe a service manager and be informed when a new instance is added or an existing one goes away.

Once you know which service instance you want to use you must open it in order to gain access. This returns the uid of the service. You can then send messages directly to the service object. You must close the service instance after you are done using it.
Basic Service Manager Usage

The simplest use of a service manager is to access a known service instance on the manager. Here's an example:

```c
SM_ACCESS access;
SM_RELEASE release;
access.pServiceName = "Service Instance Name";
access.caller = self;
ObjCallRet(msgSMAccess, aServiceManager, &access, s);
// access.service can now be sent messages.
...
// When you are done with the service, release it.
release.caller = self;
release.service = access.service;
release.handle = access.handle;
ObjCallRet(msgSMRelease, aServiceManager, &release, s);
```

Some service instances allow the client to specify pArgs. You must initialize the pArgs with msgSMAccessDefaults for these. For example:

```c
access.pServiceName = "Service Instance Name";
access.caller = self;
access.pArgs = &args;
ObjCallRet(msgSMAccessDefaults, aServiceManager, &access, s);
args.foo = ...;
ObjCallRet(msgSMAccess, aServiceManager, &access, s);
```

Advanced Service Manager Usage

Accessing a service instance is actually composed of several steps. msgSMAccess and msgSMRelease performs all of them at once; more sophisticated users might find situations where they need to control the intermediary steps themselves.

Each service instance has a 32 bit "handle" associated with it in addition to its name. This handle is a convenient shorthand for referencing a service instance. Most service manager messages use handles. Note that a handle is not a permanent id; it is dynamically generated when a service instance is first added to a service manager, and regenerated whenever PenPoint is rebooted. Handles should never be filed.

Enumerating all of the service instances on a service manager is done by getting a list of all the handles and going through the list. For example, here's some code that gets all the names of all the service instances on a manager list:

```c
OBJECT list;
LIST_ENTRY le;
IM_GET_SET_NAME getName;
ObjectCall(msgIMGetList, aServiceManager, &list);
ObjectCall(msgListNumItems, list, &n);
for (le.position = 0; le.position < n; le.position++) {
    ObjectCall(msgListGetItem, list, &le);
    getName.handle = (OBJECT) le.item;
    getName.pName = pName;
    ObjectCall(msgIMGetName, aServiceManager, &getName);
    // le.item is the handle, pName contains the name.
}
ObjCallWarn(msgDestroy, list, pNull);
```
If you know the name of a service, you can get its handle with `msgIMFind`:

```c
find.pName = "Service Instance Name";
ObjectCall(msgIMFind, aServiceManager, &find);
serviceInstanceHandle = find.handle;
```

The next step in accessing a service instance is binding. Binding tells a service instance that you are interested in it. After you have bound to a service you will get messages from that service telling you about changes in its state, such as when it becomes connected or disconnected.

```c
bind.handle = serviceInstanceHandle;
bind.caller = self;
ObjectCall(msgSMBind, aServiceManager, &bind);
```

Next you become the owner of the service instance. Ownership gives you the right to open the instance. It is the mechanism used to ensure that only one client is using a exclusive access device (such as a serial port) at a time. Some services are non-exclusive access (such as network devices). Setting owner is a no-op for these.

The owner protocol informs the both the new and old owners that an ownership change is being proposed. Either of them can veto the change. The service instance can also veto the change.

The owner of a service can be set to `objNull` to signify no owner. You should do this when you want to give up ownership of a service instance.

Here is an example of requesting an owner change:

```c
setOwner.owner = newOwner;
setOwner.handle = serviceInstanceHandle;
ObjectCall(msgSMSetOwner, aServiceManager, &setOwner);
```

Now you can open the service. An open request can optionally take `pArgs`. The format of the `pArgs` is service-specific. However, all the service instances on a particular service manager have the same `pArgs` format. The `pArgs` must be set to defaults with `msgSMOpenDefaults`.

A service that has open instances cannot be deinstalled. An open service instance cannot have its owner changed. Here is an example of opening a service instance:

```c
open.caller = self;
open.handle = serviceInstanceHandle;
opent.pArgs = &openArgs;
ObjectCall(msgSMOpenDefaults, aServiceManager, &open);
ObjectCall(msgSMOpen, aServiceManager, &open);
// open.service contains the service object at this point
```

Clients should close a service instance when they have completed using it:

```c
close.caller = self;
close.handle = serviceInstanceHandle;
close.service = open.service;
close.pArgs = pNull;
ObjectCall(msgSMClose, aServiceManager, &close);
```

Clients should unbind from a service instance when they are no longer interested in it.

```c
unbind.handle = serviceInstanceHandle;
unbind.caller = self;
ObjectCall(msgSMUnbind, aServiceManager, &unbind);
```
Additional Service Manager Functionality

Adding yourself as an observer of a service manager will cause all notification messages from the service manager and all the service instances on the service manager to go to you. These messages include:

- **msgIMInstalled**: A new service has been added to the service manager.
- **msgIMDeinstalled**: A service has been removed from the service manager.
- **msgIMInUseChanged**: A service has been opened or closed.
- **msgIMModifiedChanged**: A service has modified its state node.
- **msgSMConnectedChanged**: A service has become connected or disconnected.
- **msgSMOwnerChanged**: The owner of a service has changed.

Plus, any service instance can send service-specific notification messages via **msgSvcPropagateMsg** (see service.h). All observer messages include the handle of the service instance being affected and the uid of the service manager.

Sometimes a client needs to access a service object without becoming the owner, or need to override the open checks. This can be done, but it must be done with care. **msgSMQueryLock** and **msgSMQuery** can be used to do this.

QueryLocking a service returns the service uid without opening it. However, the call will fail if the service is exclusive-open and currently open. Also, a query lock will lock out other opens until the query lock is released. **msgSMQueryUnlock** must be sent to release the query lock.

**msgSMQuery** is just like **msgSMQueryLock**, except no open check is made.

Service managers automatically clean up if an object that owns or opens a service instance terminates before releasing the service instance.

There is a well-known list object, **theServiceManagers**, that is a list of all the service managers in the system. You can observe this list and get notification when a service manager is added and removed.

Creating New Service Managers

As stated above, PenPoint defines several default service managers. You can create additional service managers if you desire.

PenPoint will automatically create a service manager if a service instance tries to add itself to a service manager and the service manager doesn't exist. This allows services to be arbitrarily installed and deinstalled without having to worry about who creates and frees the service manager.
### Core Messages

#### msgNew

Creates a new service manager.

Takes P_SM_NEW, returns STATUS. Category: class message.

```c
typedef struct SM_NEW_ONLY {
    BOOLEAN autoDestroy; // Have the service manager be owned
    BOOLEAN noChecks; // Turn off error checking, client
    U32 unused2;
    U32 unused3;
    U32 unused4;
} SM_NEW_ONLY, *P_SM_NEW_ONLY;

#define serviceManagerNewFields
#define installMgrNewFields
#define SM_NEWONLY

typedef struct SM_NEW {
    serviceManagerNewFields
    SM_NEWONLY
} SM_NEW, *P_SM_NEW;
```

**Comments**

Clients (other than those who are creating their own service managers) do not call this message. The well-known service managers are created by the system at cold-boot time.

#### msgNewDefaults

Initializes the SM_NEW structure to default values.

Takes P_SM_NEW, returns STATUS. Category: class message.

```c
typedef struct SM_NEW {
    serviceManagerNewFields
} SM_NEW, *P_SM_NEW;
```

**Comments**

Sets

- `installMgr.style.createInitial = false;`
- `installMgr.style.copyOnInstall = false;`
- `installMgr.style.addToGlobalList = false;`
- `installMgr.style.createIcon = false;`
- `sm.autoDestroy = false;`
- `sm.noChecks = false;`

#### msgDump

Prints out the services known by this service manager and their state.

Takes OBJ_KEY, returns STATUS.

**Comments**

`dsServiceManager` provides an elaborate response to `msgDump`. This is very useful for debugging services!
**msgSMAccess**

Accesses a service instance, given its name.

Takes P_SM_ACCESS, returns STATUS.

```c
#define msgSMAccess
typedef struct SM_ACCESS {
    P_STRING pServiceName; // Service name.
    OBJECT caller; // Object making this call,
                   // typically self.
    P_ARGS pArgs; // Use this if service requires pArgs.
                   // Send msgSMAccessDefaults first.
    OBJECT handle; // Out: Service handle.
    OBJECT service; // Out: Service instance.
} SM_ACCESS, *P_SM_ACCESS;
```

**Arguments**

MakeMsg(clsServiceMgr, 43)

This is a convenience message that performs the sequence most clients do to access a service.

This message performs a find, bind, setOwner, and open for the specified service.

Note: This message cannot be used when you want to provide pArgs to a service.

**Return Value**

- stsNoMatch Item not found.
- stsSvcLocked Someone has this exclusive-open service query locked.
- stsSvcNotOwner Someone else is the owner of this owner-checked service.
- stsSvcAlreadyOpen Someone already has this exclusive-open service open.

**See Also**

msgIMFind

**msgSMAccessDefaults**

Sets pArgs defaults for msgSMAccess.

Takes P_SM_ACCESS, returns STATUS.

```c
#define msgSMAccessDefaults
typedef struct SM_ACCESS {
    P_STRING pServiceName; // Service name.
    OBJECT caller; // Object making this call,
                   // typically self.
    P_ARGS pArgs; // Use this if service requires pArgs.
                   // Send msgSMAccessDefaults first.
    OBJECT handle; // Out: Service handle.
    OBJECT service; // Out: Service instance.
} SM_ACCESS, *P_SM_ACCESS;
```

**Arguments**

MakeMsg(clsServiceMgr, 45)

This message should be used if the service you wish to access takes pArgs. This message sets up the defaults for the pArgs.

**Return Value**

- stsNoMatch Item not found.

**See Also**

msgSMAccess
msgSMRelease
Releases a service instance.
Takes P_SM_RELEASE, returns STATUS.
#define msgSMRelease
MakeMsg(clsServiceMgr, 44)

Arguments
typedef struct SM_RELEASE {
    OBJECT caller; // Object making this call,
    // typically self.
    OBJECT handle; // Service handle.
    OBJECT service; // Service instance.
} SM_RELEASE, *P_SM_RELEASE;

Comments
Call this message when you are finished using a service.
This is a convenience message that performs the sequence most clients do when they are finished with a service.
This message performs a close, sets the owner to objNull, and unbinds.

Return Value
stsFailed Service is not open by the caller.
Service-Specific Error Returns.

See Also
msgSMClose

msgSMBind
Binds to a service.
Takes P_SM_BIND, returns STATUS.
#define msgSMBind
MakeMsg(clsServiceMgr, 1)

Arguments
typedef struct SM_BIND {
    IM_HANDLE handle; // Service handle to bind to.
    OBJECT caller; // Object making this call.
} SM_BIND, *P_SM_BIND;

Comments
The caller is made an observer of this service. Service manager notification messages will be sent to the caller.
The caller is added to the bind list of the service instance.
Sends msgSvcBindRequested to the service being bound to. The service has the right to refuse the bind.
The service-specific error return that indicates a refusal is passed back to the client.

Return Value
stsBadObject Caller is not an object.
stsBadAncestor Caller has invalid ancestor.
Service-Specific Error Returns.

See Also
msgSvcBindRequested (service.h) (service.h)

msgSMUnbind
Unbinds from a service.
Takes P_SM_BIND, returns STATUS.
#define msgSMUnbind
MakeMsg(clsServiceMgr, 2)
**Message**

typedef struct SM_BIND {
    IM_HANDLE handle;  // Service handle to bind to.
    OBJECT caller;     // Object making this call.
} SM_BIND, *P_SM_BIND;

**Arguments**

Part 13 / Writing PenPoint Services

**Comments**

This removes the caller as an observer of the handle and removes the caller from the service instance's bind list.

Note: Clients must first close a service before unbinding from it.

The service manager will automatically send msgSMUnbind for all services that are bound to a client when that client object is freed. This means that you must not send msgSMUnbind from your msgFree routine; the object freed notification occurs before your msgFree routine is entered.

Sends msgSvcUnbindRequested to the service being unbound from.

**Return Value**

stsFailed Service is not bound by the caller.

---

**msgSMGetOwner**

Gets the current owner of a service.

Takes P_SM_GET_OWNER, returns STATUS.

#define msgSMGetOwner MakeMsg(clsServiceMgr, 31)

typedef struct SM_GET_OWNER {
    IM_HANDLE handle;  // Handle of item to get owner on.
    OBJECT owner;      // Out: current owner.
} SM_GET_OWNER, *P_SM_GET_OWNER;

**Arguments**

msgSMSetOwner

Sets a new service owner.

Takes P_SM_SET_OWNER, returns STATUS.

#define msgSMSetOwner MakeMsg(clsServiceMgr, 11)

typedef struct SM_SET_OWNER {
    IM_HANDLE handle;  // Handle of item to set owner on.
    OBJECT owner;      // New owner.
} SM_SET_OWNER, *P_SM_SET_OWNER;

**Arguments**

Old and new owners (whether they are clients or other services) will receive service messages which allow them to veto the ownership change and informs them that the change has taken effect. The message sequence is as follows:

1. msgSvcOwnerAcquireRequested is sent to the new owner. The new owner can veto the owner change by returning a status of anything other than stsOK or stsNotUnderstood. msgSMSetOwner returns with the abort status.
2. msgSvcOwnerReleaseRequested is sent to the old owner. The old owner can veto the owner change by returning a status of anything other than stsOK or stsNotUnderstood. If the old owner agrees to the ownership change it must immediately close the service if it is open.
3. msgSvcChangeOwnerRequested is sent to the service. This informs the service that ownership is going to be changed and allows it to veto. By default the services will veto the change if they are open.
4. msgSvcOwnerReleased is sent to the old owner.
5. msgSvcOwnerAcquired is sent to the new owner.
6. msgSMOwnerChanged is sent to everyone who is bound to the service or observing a service manager that the service is on.
Return Value
stsBadObject  New owner is not an object.
stsBadAncestor New owner has invalid ancestor.
stsSvcInUse  Service is open.
See Also
service.h, for definition of msgSvc... messages.

msgSMOpen
Opens a service, given its handle.
Takes P_SM_OPEN_CLOSE, returns STATUS.

#define msgSMOpen  MakeMsg(clsServiceMgr, 4)

typedef struct SM_OPEN_CLOSE {
    IM_HANDLE  handle;     // Handle of service to open.
    OBJECT     caller;     // Object making this call.
    P_ARGS     pArgs;      // Service-specific open parameters.
    OBJECT     service;    // In:  (SMClose) Out:  (SMOpen) Service
} SM_OPEN_CLOSE, *P_SM_OPEN_CLOSE;

Clients should do this only when they are ready to transfer data to the service, and should leave the
service open for as little time as possible.
A bind is automatically performed if the client is not yet bound.
The caller is added to the open list of the service instance.
Sends msgSvcOpenRequested to the service being opened. The service has the right to refuse the open.
The service-specific error return that indicates a refusal is passed back to the client.

Return Value
stsBadObject  Caller is not an object.
stsBadAncestor Caller has invalid ancestor.
stsSvcNotBound Caller is not bound to the service.
stsSvcLocked Someone has this exclusive-open service query locked.
stsSvcNotOwner Someone else is the owner of this owner-checked service.
stsSvcAlreadyOpen Someone already has this exclusive-open service open.

Service-Specific Error Returns

See Also
msgSMBind (service.h) (service.h)(service.h)

msgSMOpenDefaults
Initializes SMOpen pArgs to default value.
Takes P_SM_OPEN_CLOSE, returns STATUS.

#define msgSMOpenDefaults  MakeMsg(clsServiceMgr, 34)

typedef struct SM_OPEN_CLOSE {
    IM_HANDLE  handle;     // Handle of service to open.
    OBJECT     caller;     // Object making this call.
    P_ARGS     pArgs;      // Service-specific open parameters.
    OBJECT     service;    // In:  (SMClose) Out:  (SMOpen) Service
} SM_OPEN_CLOSE, *P_SM_OPEN_CLOSE;

See Also
msgSMOpen (service.h)
msgSMClose

Close an open service.

Takes P_SM_OPEN_CLOSE, returns STATUS.

```c
#define msgSMClose MakeMsg(clsServiceMgr, 5)
```

**Message Arguments**

```c
typedef struct SM_OPEN_CLOSE {
    IM_HANDLE handle; // Handle of service to open.
    OBJECT caller; // Object making this call.
    P_ARGS pArgs; // Service-specific open parameters.
    OBJECT service; // In: (SMClose) Out: (SMOpen) Service
} SM_OPEN_CLOSE, *P_SM_OPEN_CLOSE;
```

**Comments**
The caller is removed from the open list of the service instance.

Clients should send msgSMClose as soon as they are finished actively transferring data. Clients *must* first close a service before unbinding from it.

The service manager will automatically send msgSMClose for all services that are held open by a client when that client object is freed. This means that you must not send msgSMClose from your msgFree routine; the object freed notification occurs before your msgFree routine is entered.

Sends msgSvcCloseRequested to the service being opened.

**Return Value**
stsFailed Service instance is not open by the caller.

msgSMQueryLock

Gets the uid of a service and locks out any opens.

Takes P_SM_QUERY_LOCK, returns STATUS.

```c
#define msgSMQueryLock MakeMsg(clsServiceMgr, 8)
```

**Message Arguments**

```c
typedef struct SM_QUERY_LOCK {
    IM_HANDLE handle; // Handle of service instance to query.
    OBJECT service; // Out: Service object.
} SM_QUERY_LOCK, *P_SM_QUERY_LOCK;
```

**Comments**
This message is similar to msgSMOpen, in that it returns a service object, given a handle. However, it is not seen as an open by the service.

This message is meant for non-data transfer access to a service, for example, generating a service's option card.

The sender of this message does *not* have to be the owner of the service.

This message will fail if the service instance is exclusive open and currently in use (open). If this message succeeds then all opens will fail until msgSMQueryUnlock is sent.

This message will return the real uid of the service instance in the case of a multi-user service.

**Return Value**
stsSvcLocked Service instance is already query locked.

stsSvcInUse Service instance is open.
**msgSMQueryUnlock**

Unlocks a service that was locked via msgSMQueryLock.

Takes P_SM_QUERY_UNLOCK, returns STATUS.

```c
#define msgSMQueryUnlock MakeMsg(clsServiceMgr, 9)
```

**Arguments**

typedef struct SM_QUERY_UNLOCK {
    HANDLE handle;    // Handle of service instance to unlock.
} SM_QUERY_UNLOCK, *P_SM_QUERY_UNLOCK;

**msgSMQuery**

Gets the uid of a service.

Takes P_SM_QUERY_LOCK, returns STATUS.

```c
#define msgSMQuery MakeMsg(clsServiceMgr, 33)
```

**Message**

typedef struct SM_QUERY_LOCK {
    HANDLE handle;    // Handle of service instance to query.
    OBJECT service;   // Out: Service object.
} SM_QUERY_LOCK, *P_SM_QUERY_LOCK;

**Comments**

This message gets the uid of a service instance. It must be used very carefully. It bypasses all checking mechanisms, so the caller can get into trouble if he subsequently sends messages to the service that are not expected. Use msgSMQueryLock instead of msgSMQuery if at all possible.

**msgSMGetCharacteristics**

Gets the characteristics of the specified service instance.

Takes P_SM_GET_CHARACTERISTICS, returns STATUS.

```c
#define msgSMGetCharacteristics MakeMsg(clsServiceMgr, 42)
```

**Arguments**

typedef struct SM_GET_CHARACTERISTICS {
    HANDLE handle;    // Handle of item to get characteristics of.
    P UNKNOWN pBuf;   // Out through Ptr: Characteristics buffer.
    U16 len;          // In/Out: Buffer size. If 0 then the actual size is returned.
} SM_GET_CHARACTERISTICS, *P_SM_GET_CHARACTERISTICS;

**Comments**

Characteristics are service-specific properties of a particular service. For example, modem services might pass back whether Fax is supported, maximum baud rate, etc. All the services on a particular service manager return the same characteristics set.

Callers should first send this message with pArgs->len set to 0. This will return the size of the actual characteristics buffer. Callers should then allocate this space and make the call again with pArgs->len set to this size. pArgs->len can be less than the actual size, in which case only the number of bytes specified by pArgs->len is returned.

**msgSMSave**

Saves a service instance to a specified external location.

Takes P_SM_SAVE, returns STATUS.

```c
#define msgSMSave MakeMsg(clsServiceMgr, 36)
```
typedef struct SM_SAVE {
    IM_HANDLE  handle;  // Handle of service instance to save.
    BOOLEAN    reserved;  // Reserved.
    FS_FLAT_LOCATOR flat;  // Location to save to.
} SM_SAVE, *P_SM_SAVE;

The pArgs specify the parent directory that the service instance will save itself into. Note that the service instance’s current target is also saved. When the service instance is reloaded it will try and bind to this target.

See Also
msgSvcClassLoadInstance  
load a service instance from arbitrary location on disk (service.h).

### Auxiliary Messages

**msgSMFindHandle**

Finds a handle, given a service instance uid.

Takes P_SM_FIND_HANDLE, returns STATUS.

#define msgSMFindHandle MakeMsg(clsServiceMgr, 10)

**msgSMSetOwnerNoVeto**

Sets a new service owner without giving owners veto power.

Takes P_SM_SET_OWNER, returns STATUS.

#define msgSMSetOwnerNoVeto MakeMsg(clsServiceMgr, 30)

This message is the same as msgSMSetOwner, except the old owner and new owners do not get the chance to veto. msgSvcReleaseRequest and msgSvcAquireRequest are not sent. This message does the following:

1. The open status of the service is checked. If it is open (in use) the SetOwner fails, with a return status of stsSvcInUse.
2. msgSvcChangeOwnerRequested is sent to the service. This informs the service that ownership is going to be changed and allows it to veto the owner change by returning anything other than stsOK or stsNotUnderstood. msgSMSetOwner returns with the abort status.
3. msgSMOwnerChanged is sent to everyone who is bound to the service or observing a service manager that the service is on.
4. msgSvcOwnerReleased is sent to the old owner.
5. msgSvcOwnerAquired is sent to the new owner.

Return Value

stsSvcInUse  
Service is open.
**msgSMGetState**

Gets the state of a service.

Takes P_SM_GET_STATE, returns STATUS.

```c
define msgSMGetState
    MakeMsg(clsServiceMgr, 12)
```

Arguments

```c
typedef struct SM_GET_STATE {
    IM_HANDLE handle;  // In: Handle of service to get state on.
    BOOLEAN connected; // Out: Is service connected?
    OBJECT owner;      // Out: My owner, if any.
    OBJECT owned;      // Out: The service that I own, if any.
    U8 reserved[24];
} SM_GET_STATE, *P_SM_GET_STATE;
```

Comments

This message provides service state. There is some additional state (in use, modified) that is gotten via `msgIMGetState`. See instlmg.h for details.

---

**msgSMGetClassMetrics**

Gets the service's class metrics.

Takes P_SM_GET_CLASS_METRICS, returns STATUS.

```c
define msgSMGetClassMetrics
    MakeMsg(clsServiceMgr, 13)
```

Arguments

```c
typedef struct SM_GET_CLASS_METRICS {
    IM_HANDLE handle;  // Handle of item to get class metrics on.
    SVC_CLASS_METRICS metrics;
} SM_GET_CLASS_METRICS, *P_SM_GET_CLASS_METRICS;
```

Comments

This message passes back information about the service class. See service.h for a definition of SVC_CLASS_METRICS.

---

**msgIMDeinstall**

Remove and free a service instance.

Takes P_IM_DEINSTALL, returns STATUS.

Comments

This will remove the specified service instance from all the service managers that it is on, destroy its state file, and free it.

Note that a service is initially created by sending `msgNew` to the service class. Services automatically add themselves to service manager. Do not use `msgIMInstall` for this purpose; `msgIMInstall` should NEVER be used by clients.

Causes observer message `msgIMDeinstalled` to be propogated to all objects that are bound to the service instance and to the service managers.

This message causes `msgSvcDeinstallRequested` to be sent to the service instance. The instance can veto the deinstall at this point; if it does then the return value from `msgIMDeinstall` is the status that the service instance used to veto the deinstall.

See Also

`msgSvcDeinstallRequested`
Notification Messages

msgSMConnectedChanged
A service’s connection state changed.
Takes P_SM_CONNECTED_NOTIFY, returns STATUS. Category: observer notification.

#define msgSMConnectedChanged MakeMsg(clsServiceMgr, 20)

typedef struct SM_CONNECTED_NOTIFY {
    OBJECT manager; // manager that sent notification
    HANDLE handle;  // handle to service
    BOOLEAN connected; // new connect state
} SM_CONNECTED_NOTIFY, *P_SM_CONNECTED_NOTIFY;

msgSMOwnerChanged
A service’s owner has changed.
Takes P_SM_OWNER_NOTIFY, returns STATUS. Category: observer notification.

#define msgSMOwnerChanged MakeMsg(clsServiceMgr, 21)

typedef struct SM_OWNER_NOTIFY {
    OBJECT manager; // manager that sent notification
    HANDLE handle;  // handle to service
    OBJECT oldOwner; // old owner
    OBJECT owner;   // new owner
} SM_OWNER_NOTIFY, *P_SM_OWNER_NOTIFY;
SERVMISC.H

This file contains additional API definitions for clsService.

clsService inherits from clsStream.

Provides default behavior for services.

This header file defines auxiliary clsService messages that are not used by the majority of service clients.

```c
#ifndef SERVMISC_INCLUDED
#define SERVMISC_INCLUDED

Owner Messages

msgSvcGetMyOwner
Gets the current owner of this service, if any.
Takes P_OBJECT, returns STATUS.
#define msgSvcGetMyOwner MakeMsg(clsService, 21)

Comments
Passes back objNull if there is no current owner.

msgSvcGetOwned
Passes back the item that this service owns.
Takes P_OBJECT, returns STATUS.
#define msgSvcGetOwned MakeMsg(clsService, 31)

Comments
This message is only valid for autoOwnTarget services (style.autoOwnTarget is true).
If this service has become the owner of its target then this message passes back the item that it owns;
otherwise it returns objNull.

msgSvcOwnerReleaseRequested
Is it OK to remove you as the owner of a service?
Takes P_SVC_OWNED_NOTIFY, returns STATUS.
#define msgSvcOwnerReleaseRequested MakeMsg(clsService, 38)

Arguments
typedef struct SVC_OWNED_NOTIFY {
    OBJECT ownedService; // The service or MIL conflict
    OBJECT oldOwner;    // group which will have its
    OBJECT newOwner;    // owner changed.
    U8  reserved[16];
} SVC_OWNED_NOTIFY, *P_SVC_OWNED_NOTIFY;

Comments
A client sent msgSMSetOwner to a service manager for a service you currently own. See
servmgr.h/msgSMSetOwner for details on the entire owner change message protocol.
You can veto the ownership change by returning anything other than `stsOK` or `stsNotUnderstood`.

The service must not be in use for the owner change to occur. If you have the service open and want to give up ownership, you should close the service when you receive this message.

This message must be passed to ancestor.

**msgSvcOwnerAcquireRequested**

Is it OK to make you the new owner of a service?

Takes `P_SVC_OWNED_NOTIFY`, returns `STATUS`.

```c
#define msgSvcOwnerAcquireRequested MakeMsg(clsService, 39)
```

**Message Arguments**

```c
typedef struct SVC_OWNED_NOTIFY {
  OBJECT ownedService; // The service or MIL conflict group which will have its owner changed.
  OBJECT oldOwner; // The old owner.
  OBJECT newOwner; // The proposed new owner.
  U8 reserved[16];
} SVC_OWNED_NOTIFY, *P_SVC_OWNED_NOTIFY;
```

**Comments**

A client sent `msgSMSetOwner` to a service manager, proposing that you be the new owner of a service. See `servmgr.h/msgSMSetOwner` for details on the entire owner change message protocol.

You can veto the ownership change by returning anything other than `stsOK` or `stsNotUnderstood`.

This message must be passed to ancestor.

**msgSvcOwnerAcquired**

You are now the new owner of a service.

Takes `P_SVC_OWNED_NOTIFY`, returns `STATUS`.

```c
#define msgSvcOwnerAcquired MakeMsg(clsService, 29)
```

**Message Arguments**

```c
typedef struct SVC_OWNED_NOTIFY {
  OBJECT ownedService; // The service or MIL conflict group which will have its owner changed.
  OBJECT oldOwner; // The old owner.
  OBJECT newOwner; // The proposed new owner.
  U8 reserved[16];
} SVC_OWNED_NOTIFY, *P_SVC_OWNED_NOTIFY;
```

**Comments**

A client sent `msgSMSetOwner` to a service manager and requested that you become the new owner of the service. This message signifies that you are the new owner of the service. See `servmgr.h/msgSMSetOwner` for details on the entire owner change message protocol.

Any saved state that you have for the owned service should be restored (typically via `msgSvcSetMetrics`).

This message must be passed to ancestor.

**msgSvcOwnerReleased**

You are no longer the owner of a service.

Takes `P_SVC_OWNED_NOTIFY`, returns `STATUS`.

```c
#define msgSvcOwnerReleased MakeMsg(clsService, 30)
```
typedef struct SVC_OWNED_NOTIFY {
    OBJECT ownedService; /\ The service or MIL conflict
    // group which will have its
    // owner changed.
    OBJECT oldOwner; /\ The old owner.
    OBJECT newOwner; /\ The proposed new owner.
    U8 reserved[16];
} SVC_OWNED_NOTIFY, *P SVC_OWNED_NOTIFY;

A client sent msgSMSetOwner to a service manager for a service you currently own. This message signifies that you are no longer the owner of the service. See servmgr.h/msgSMSetOwner for details on the entire owner change. The ownership change actually happens when you return from this message.

Any state for the owned state that you are interested in preserving should be gotten (typically via msgSvcGetMetrics) and saved in your state file. You can manipulate the service as its owner until you return from this message.

This message must be passed to ancestor.

msgSvcChangeOwnerRequested
Owner change request message.
Takes P_SVC_OWNED_NOTIFY, returns STATUS.

#define msgSvcChangeOwnerRequested MakeMsg(clsService, 40)

typedef struct SVC_OWNED_NOTIFY {
    OBJECT ownedService; /\ The service or MIL conflict
    // group which will have its
    // owner changed.
    OBJECT oldOwner; /\ The old owner.
    OBJECT newOwner; /\ The proposed new owner.
    U8 reserved[16];
} SVC_OWNED_NOTIFY, *P SVC_OWNED_NOTIFY;

This message is sent to the service instance whose owner is being changed. The service instance can veto the ownership change by returning anything other than stsOK or stsNotUnderstood.

This message must be passed to ancestor if the service does not want to veto the owner change.

Save Messages

msgSvcSaveRequested
Client asked to save this instance to external media.
Takes P_FS_FLAT_LOCATOR, returns STATUS.

#define msgSvcSaveRequested MakeMsg(clsService, 34)

A client sent msgSMSave to a service manager.

Default superclass behavior is to save the state file and the current target only. Subclasses should ensure that their state file is up to date if they wish to make use of this behavior. Alternatively, subclasses can not pass this message to ancestor and perform whatever form of save they wish.

The pArgs references the parent directory in which this service instance should be saved. If a node with the same name as the service instance already exists within this directory, default superclass behavior is to overwrite the destination. Subclasses can perform other forms of behavior if the destination exists before passing this message to ancestor.

This message does not have to be passed to ancestor.
**msgSvcClassLoadInstance**

Loads an instance state file from disk and creates a new instance.

Takes `P_SVC_LOAD_INSTANCE`, returns `STATUS`. Category: class message.

```c
#define msgSvcClassLoadInstance MakeMsg(clsService, 47)
```

**Arguments**

```c
typedef struct SVC_LOAD_INSTANCE {
    FS_LOCATOR source; // Source state file location.
} SVC_LOAD_INSTANCE, *P_SVC_LOAD_INSTANCE;
```

**Comments**

This function copies the state node specified by `pArgs->source` into the INST directory of the service and starts up an instance of the service on this state file. This is very similar to what happens when a warm-boot occurs, or when state nodes are automatically loaded when a service is first installed.

If a service instance with the same name already exists, default behavior is to generate a unique name for the new service instance.

Subclasses do not normally process this message, but can if they wish to change the existing behavior.

**Return Value**

`stsFSNodeNotFound` source file not found.

**See Also**

`msgSMSave`

---

**Class Metrics Messages**

**msgSvcGetClassMetrics**

Gets metrics for the service class that controls this instance.

Takes `P_SVC_CLASS_METRICS`, returns `STATUS`.

```c
#define msgSvcGetClassMetrics MakeMsg(clsService, 23)
```

**Comments**

Note: This message can also be sent directly to the service class.

---

**Instance Metrics Messages**

**msgSvcGetMetrics**

Passes back the current configuration metrics.

Takes `P_SVC_GET_SET_METRICS`, returns `STATUS`.

```c
#define msgSvcGetMetrics MakeMsg(clsService, 32)
```

**Arguments**

```c
typedef struct SVC_GET_SET_METRICS {
    P_UNKNOWN pMetrics; // Out through Ptr: Metrics buffer.
    U16 len; // In/Out: Metrics buffer size in bytes. If 0 then the actual size is returned.
} SVC_GET_SET_METRICS, *P_SVC_GET_SET_METRICS;
```

**Comments**

Configuration metrics are service specific. This interface allows the caller to find out how large the metrics set for a given service are.

The caller should first send `msgSvcGetMetrics` with `pArgs->len` set to 0 to get the actual size of the metrics buffer. The caller should allocate a buffer of this size then send the message again.

Subclasses that have configuration metrics must handle this message.
msgSvcSetMetrics
Sets the configuration metrics.
Takes P_SVC_GET_SET_METRICS, returns STATUS.

#define msgSvcSetMetrics MakeMsg(clsService, 33)

typedef struct SVC_GET_SET_METRICS {
    P_UNKNOWN pMetrics; // Out through Ptr: Metrics buffer.
    U16 len; // In/Out: Metrics buffer size in bytes. If 0 then the actual size is returned.
} SVC_GET_SET_METRICS, *P_SVC_GET_SET_METRICS;

Configuration metrics are service specific. The caller should set pArgs->len to the size that was returned from msgSvcGetMetrics when the metrics were originally gotten. A caller should never try and synthesize a metrics buffer; he should only pass back a buffer that was gotten from msgSvcGetMetrics.

Subclasses can determine the version of a configuration buffer from its size. Subclasses should make sure that different versions of configuration information have different sizes.
Subclasses must update their state node when they handle this message.
Subclasses that have configuration metrics must handle this message.

Service Manager Messages

msgSvcAddToManager
Add this service instance to a service manager.
Takes P_SVC_ADD_TO_MANAGER, returns STATUS.

#define msgSvcAddToManager MakeMsg(clsService, 17)

typedef struct SVC_ADD_TO_MANAGER {
    OBJECT manager;
} SVC_ADD_TO_MANAGER, *P_SVC_ADD_TO_MANAGER;

This message allows a service to add itself to additional service managers after msgNew time.
This results in msgIMInstalled being sent to observers of the service manager.

msgSvcRemoveFromManager
Removes this service instance from a service manager.
Takes P_SVC_REMOVE_FROM_MANAGER, returns STATUS.

#define msgSvcRemoveFromManager MakeMsg(clsService, 18)

typedef struct SVC_REMOVE_FROM_MANAGER {
    OBJECT manager; // Manager to remove self from
} SVC_REMOVE_FROM_MANAGER, *P_SVC_REMOVE_FROM_MANAGER;

This message allows a service to remove itself from a service manager it is currently on.
This results in msgIMDeinstalled being sent to observers of the service manager and any objects which have bound to this service. It cleans up this service's bind list, removing anyone who bound via the specified service manager.

Note: service managers automatically remove a service when the service class is deinstalled. There is no need to do so explicitly.

Return Value
stsNoMatch Service instance is not on the specified service manager.
Client List Messages

msgSvcGetBindList

Gets a list of all the callers that have bound to this service.

Takes P_SVC_GET_LIST, returns STATUS.

#define msgSvcGetBindList

typedef struct SVC_GET_LIST {
P_OBJECT pList; // Out: list, allocated from process heap.
   // CLIENT MUST OSHeapBlockFree WHEN
   // FINISHED!
   U16 count; // Out: number of elements in list
} SVC_GET_LIST, *P_SVC_GET_LIST;

msgSvcGetOpenList

Gets a list of all the callers that have opened this service.

Takes P_SVC_GET_LIST, returns STATUS.

#define msgSvcGetOpenList

typedef struct SVC_GET_LIST {
P_OBJECT pList; // Out: list, allocated from process heap.
   // CLIENT MUST OSHeapBlockFree WHEN
   // FINISHED!
   U16 count; // Out: number of elements in list
} SVC_GET_LIST, *P_SVC_GET_LIST;

msgSvcGetOpenObjectList

Gets a list of the open objects which were returned for each open.

Takes P_SVC_GET_LIST, returns STATUS.

#define msgSvcGetOpenObjectList

typedef struct SVC_GET_LIST {
P_OBJECT pList; // Out: list, allocated from process heap.
   // CLIENT MUST OSHeapBlockFree WHEN
   // FINISHED!
   U16 count; // Out: number of elements in list
} SVC_GET_LIST, *P_SVC_GET_LIST;

Comments

This list is ordered the same as the open list. The caller in openlist[i] was given the object in openObjectList[i].

See Also

msgSvcGetOpenList

msgSvcGetManagerList

Gets a list of all the service managers that this service is on.

Takes P_SVC_GET_LIST, returns STATUS.

#define msgSvcGetManagerList

MakeMsg(clsService, 26)

MakeMsg(clsService, 27)

MakeMsg(clsService, 49)

MakeMsg(clsService, 28)
typedef struct SVC_GET_LIST {
    P_OBJECT pList; // Out: list, allocated from process heap.
    // CLIENT MUST OSHeapBlockFree WHEN
    // FINISHED!
    U16 count; // Out: number of elements in list
} SVC_GET_LIST, *P_SVC_GET_LIST;

msgSvcGetManagerHandleList

Gets a list of the svc mgr handles that this service is represented by.

Takes P_SVC_GET_LIST, returns STATUS.

#define msgSvcGetManagerHandleList MakeMsg(clsService, 50)

msgSvcGetDependentAppList

Gets a list of theInstalledApps handles for all dependent apps.

Takes P_SVC_GET_LIST, returns STATUS.

#define msgSvcGetDependentAppList MakeMsg(clsService, 51)

msgSvcGetDependentServiceList

Gets a list of theInstalledServices handles for all dependent services.

Takes P_SVC_GET_LIST, returns STATUS.

#define msgSvcGetDependentServiceList MakeMsg(clsService, 52)
Deinstallation/Destruction Messages

msgSvcClassTerminateOK

Deinstalls the entire service.

Takes P_OBJECT, returns STATUS. Category: class message.

#define msgSvcClassTerminateOK MakeMsg(clsService, 43)

Comments

Deinstallation is a two-phase process. The first phase allows any of the services or apps being deinstalled to cancel the entire deinstall. msgSvcClassTerminateOK is the veto phase. Returning anything other than stsOK signifies a veto. If anyone vetos the deinstall then msgSvcClassTerminateVetoed is sent to all services that were sent msgSvcClassTerminateOK. If nobody vetos the deinstall then msgSvcClassTerminate is sent.

The pArgs to msgSvcClassTerminateOK is used to pass back the object which is responsible for the veto.

Default superclass behavior is to send msgSvcDeinstallRequested to each instance of the service, and veto the deinstallation if any service instance vetos the deinstallation. The uid of the instance that vetoed the deinstall is passed back via the pArgs.

This approach allows multiple services and applications that are dependent on each other to be deinstalled in a coherent fashion.

Subclasses can override this message if they wish.

See Also

msgSvcDeinstall

msgSvcClassTerminateVetoed

Deinstall process was vetoed.

Takes P_SVC_TERMINATE_VETOED, returns STATUS. Category: class message.

#define msgSvcClassTerminateVetoed MakeMsg(clsService, 45)

typedef struct SVC_TERMINATE_VETOED {
    OBJECT vetoer; // Object that vetoed the deinstall.
    STATUS status; // Veto status.
} SVC_TERMINATE_VETOED, *P_SVC_TERMINATE_VETOED;

Comments

This message informs the service that the deinstallation sequence that started with msgSvcClassTerminateOK has been vetoed by one of the services or applications that was part of the deinstall.

pArgs->vetoer gives the uid of the object or class which vetoed the deinstall. pArgs->status gives the return status of the veto.

Default superclass behavior is to send msgSvcDeinstallVetoed to each instance of the service.

Subclasses can override this message if they wish.

See Also

msgSvcDeinstallVetoed

msgSvcClassTerminate

Terminate the service.

Takes pNull, returns STATUS. Category: class message.

#define msgSvcClassTerminate MakeMsg(clsService, 24)
Unconditionally terminate the service. All applications and services that are to be deinstalled have agreed to the deinstallation.

Default superclass behavior is to send `msgDestroy` to each instance of the service.

Subclasses must pass this message to ancestor.

### msgSvcClientDestroyedEarly

An active client was destroyed.

Takes `OBJECT`, returns `STATUS`.

```c
#define msgSvcClientDestroyedEarly MakeMsg(clsService, 48)
```

This message is sent to the service instance when a caller or service owner terminates unexpectedly. The `pArgs` is the uid of the caller or owner.

Superclass behavior is to clean up the service instance by sending `msgSMUnbind`, `msgSMClose` and `msgSMSetOwner` to self as appropriate.

Services that keep their own per-client information will need to process this message in order to clean up their state.

This message must be passed to ancestor.

### msgSvcDeinstallRequested

Client asked to destroy this service instance.

Takes `pNull`, returns `STATUS`.

```c
#define msgSvcDeinstallRequested MakeMsg(clsService, 8)
```

A client has sent `msgSMDeinstall` to a service manager (to get rid of just this service instance), or the entire service class is being deinstalled.

Deinstallation is a two phase process. All service instances that are going to be deinstalled are sent `msgSvcDeinstallRequested`. Each service has the chance to veto the deinstall by returning an error status. If all parties agree to the deinstall then `msgFree` is sent to each service instance. `msgFree` cannot be vetoed. It causes the service to be removed from all service managers.

If anybody vetos the deinstall then `msgSvcDeinstallVetoed` is sent to each service that is part of the deinstall process. Services should not accept any new clients while a deinstall is in process.

`msgSvcDeinstallVetoed` indicates that new clients can once again be accepted.

Default superclass behavior is to veto the deinstall if the service is in use (open). The superclass will also handle new client rejection while a deinstall is in process if it gets this message.

This message must be passed to ancestor.

Note: A service might get `msgSvcDeinstallRequested` more than once for a given deinstallation sequence.

### msgSvcDeinstallVetoed

Deinstallation process was vetoed.

Takes `P_SVC_DEINSTALL_VETOED`, returns `STATUS`.

```c
#define msgSvcDeinstallVetoed MakeMsg(clsService, 47)
```
typedef struct SVC_DEINSTALL_VETOED {
    OBJECT vetoer; // Object that vetoed the deinstall.
    STATUS status; // Veto status.
} SVC_DEINSTALL_VETOED, *P_SVC_DEINSTALL_VETOED;

One of the objects or classes in the deinstall process decided to veto the deinstall.

Services can once again accept new clients.

This message must be passed to ancestor.

**msgDestroy**

Frees a service instance.

Takes OBJ_KEY, returns STATUS.

Subclasses should destroy all dynamic resources. Warning: Do not destroy any clsService resources, such as the state node handle!

WARNING: Clients must NEVER send msgDestroy directly to a service instance; instead they should send msgIMDeinstall to a service manager which the service instance is on.

Note that service manager message msgSMRemoveReference allows a service instance to be removed from a single service manager without removing it from other service managers, or destroying the instance. See servmgr.h for details on msgIMDeinstall and msgSMRemoveReference.

**Miscellaneous Messages**

**msgSvcGetStyle**

Returns current style settings.

Takes P_SVC_STYLE, returns STATUS.

MakeMsg(clsService, 10)

**msgSvcSetStyle**

Changes style settings.

Takes P_SVC_STYLE, returns STATUS.

MakeMsg(clsService, 11)

**msgSvcGetFunctions**

Passes back a pointer to a table of function entry points.

Takes P_SVC_GET_FUNCTIONS, returns STATUS.

MakeMsg(clsService, 1)

This is for services that cannot afford the overhead of being accessed via object calls. The format of this pointer block is up to the subclass. Default superclass behavior is to set pFunctions to pNull, which means this service doesn't provide a table.

Subclasses should handle this message if they wish to provide a function interface to their service.

Default superclass behavior is to set pArgs->pFunctions to pNull and pArgs->info to 0.
msgSvcGetName

Gets the name of this service instance.
Takes P_SVC_GET_NAME, returns STATUS.

#define msgSvcGetName

typedef struct SVC_GET_NAME {
    P_STRING pName;    // Out: caller must allocate
                       // nameBufLength buffer here
} SVC_GET_NAME, *P_SVC_GET_NAME;

msgSvcNameChanged

The service's name has been changed.
Takes pNull, returns STATUS.

#define msgSvcNameChanged

This message is self-sent to the service instance when its name is changed. This occurs when
msgIMSetName is sent to a service manager that this service is on.
The service is already set to the new name when this message is received. msgSvcGetNarne can be used
to get the new name.
This message is informational only. It does not have to be passed to ancestor.

msgSvcPropagateMsg

Propagates a service-specific message.
Takes P_SVC_PROPAGATE_MSG, returns STATUS.

#define msgSvcPropagateMsg

typedef struct SVC_PROPAGATE_MSG {
    P ARGS pArgs;
    SIZEOF pArgsSize;
    MESSAGE msg;
} SVC_PROPAGATE.MSG, *P_SVC_PROPAGATE_MSG;

This message allows services to send their own informational messages to everyone who is bound to the
service and everyone who is an observer of any service manager that this service is on. This is similar to
what the system does with messages like msgSMConnectedChanged.
The first two arguments of the pArgs of your notification message must be:
OBJECT manager;   // manager that sent notification_HANDLE handle;   // handle to service
msgSvcPropagateNotify will fill these in with the correct service manager and handle for all of the
observers. For example:

typedef struct FOO_NOTIFY {
OBJECT manager;   // svc manager that sent notification.
OBJECT handle;   // handle to service.
FOO newFoo;      // new foo.
FOO oldFoo;      // old foo.
} FOO_NOTIFY, *P_FOO_NOTIFY;
FOO_NOTIFY fooNotify; SVC_PROPAGATE_MSG propagate;
propagate.pArgs = fooNotify; propagate.pArgsSize = SizeOf(fooNotify); propagate.msg = msgFoo;
ObjCallRet(msgSvcPropagateMsg, self, &propagate, s);
msgSvcAutoDetectingHardware
Is the hardware that this service ultimately talks to auto-detecting?
Takes P_BOOLEAN, returns STATUS.

#define msgSvcAutoDetectingHardware MakeMsg(clsService, 37)

Comments
This message is propagated to this service's target, then the target's target, etc. until it finds the service which actually interfaces to hardware (has no target). The hardware interface service is then asked if it can autodetect connect/disconnect.

Return Value
stsSvcValidConnectStyleNotFound target chain ended without reaching hardware service instance.

msgSvcClassPopUpOptionSheet
Creates an option sheet for the service's global options and pops it up.
Takes pNull, returns STATUS. Category: class message.

#define msgSvcClassPopUpOptionSheet MakeMsg(clsService, 57)

Comments
The option sheet is only displayed if this is the first time the service is installed. Subclasses do not normally process this message.

msgSvcClassGetInstallDir
Creates a directory handle on the service's installation directory.
Takes P_OBJECT, returns STATUS.

#define msgSvcClassGetInstallDir MakeMsg(clsService, 58)

Comments
The service class creates a clsDirHandle object which references the location on external media that the service was installed from. If the external volume is not connected, the user is asked to attach it.

If this service was bundled with PenPoint then there is no valid external volume beyond installation time. stsFailed is returned in this case.

NOTE: CALLER IS RESPONSIBLE FOR DESTROYING THE DIR HANDLE WHEN DONE.

Return Value
stsOK The external volume is attached. The user tapped the Cancel button when prompted to attach the external volume. The external volume cannot be determined because this application was bundled with PenPoint.

Descendants: You normally do not handle this message.

Notification Messages

msgSvcTargetChanged
A service's target has changed.
Takes P_SVC_TARGET_CHANGE_NOTIFY, returns STATUS. Category: observer notification.

#define msgSvcTargetChanged MakeMsg(clsService, 53)

Arguments
typedef struct SVC_TARGET_CHANGE_NOTIFY {
    OBJECT manager;    // svc manager that sent notification.
    OBJECT handle;     // handle to service.
    SVC_TARGET oldTarget; // old target.
    SVC_TARGET newTarget; // new target.
} SVC_TARGET_CHANGE_NOTIFY, *P_SVC_TARGET_CHANGE_NOTIFY;

Comments
This message is broadcast to all service managers that this service is on.
SVCTYPES.H

This file contains the type tags for services. These tags are used to provide categories of service classes. This allows UIs like the printer manager UI to decide what types are available.

```c
#ifndef SVCTYPES_INCLUDED
#define SVCTYPES_INCLUDED
#ifndef GO_INCLUDED
#include <go.h>
#endif
#define svcTypePrinter
#define svcTypeEMail
MakeTag(clsService, 1)
MakeTag(clsService, 2)
```
Part 14 / Miscellaneous
This file contains the API definition for clsMILPowerDevice.

clsMILPowerDevice inherits from clsService.

theBattery is a well-known instance of clsMILPowerDevice. theBattery provides access to the primary battery of the computer.

theBatteries is a well-known instance of clsServiceManager. theBatteries is the service manager that manages the instances of clsMILPowerDevice that represent the computer's batteries (including theBattery).

clsMILPowerDevice provides an object interface to the computer's power devices (i.e. batteries).

ifndef BATTERY_INCLUDED
#define BATTERY_INCLUDED
ifndef MILSERV_INCLUDED
#include <milserv.h>
#endif
#endif

## Types and Constants

**// battery flags**

#define milRawVoltsSupported flag0
#define milPercentLeftSupported flag1
#define milSecondsLeftSupported flag2
#define milSetLevelSupported flag3

## Messages

**msgBatteryGetMetrics**

Passes back the battery's metrics.

Takes P_BATTERY_METRICS, returns STATUS.

```c
#define msgBatteryGetMetrics MakeMsg(clsMILPowerDevice, 1)
```

**Arguments**

```c
typedef struct BATTERY_METRICS {
    U16 batteryFlags;
    U16 maxMillivolts;
    U16 warnMillivolts;
    U16 failMillivolts;
    U16 currentMillivolts;
    U16 percentOfBatteryLeft;
    U16 maxSeconds;
    U16 secondsOfBatteryLeft;
} BATTERY_METRICS, *P_BATTERY_METRICS;
```
msgBatterySetLevel
Sets the percentage of battery remaining.
Takes UI6, returns STATUS.
#define msgBatterySetLevel MakeMsg(clsMILPowerDevice, 2)

Comments
The MIL request milPowerSetBatteryLevel is sent to the MIL device unit represented by the receiver.

msgBatteryLow
Sent when a battery level is dangerously low.
Takes void, returns STATUS. Category: observer notification.
#define msgBatteryLow MakeMsg(clsMILPowerDevice, 128)

msgBatteryCritical
Sent when a battery drops level below the shutdown level.
Takes void, returns STATUS. Category: observer notification.
#define msgBatteryCritical MakeMsg(clsMILPowerDevice, 129)
This file contains the API definition for dynarray. Dynarrays provide a set of dynamic array routines. The functions described in this file are contained in XLIST.LIB.

Implements a dynamic array of elements. Standard interface routines for indexing, inserting, deleting, and other common operations are provided. This interface is primarily used internally to GO, and is therefore tailored to meet internal needs.

A dynamic array is a simple data structure that contains some array information fields, and a pointer to a block of memory. This block of memory is equal to pArray->entries * pArray->elementSize.

The number of entries is specified at initialization time in DynArrayNew, and can be changed via DynArrayContract, or DynArrayExpand. These are implicitly called from DynArrayInsert and DynArrayDelete when inserting an item into a list that does not have enough entries available, or when deleting an item from the list. At any time, the value returned by DynArrayCount, or pArray->entries, will be equal to the number of entries allocated in the array in pArray->pData. The size of the array in pArray->pData will be equal to pArray->entries * pArray->elementSize.

The maximum index set in the array at any given time, independent of the number of entries in the array, is referred to as maxCount. This is equal to the greatest array index number set via DynArraySet or inserted via DynArrayInsert. It is also updated in DynArrayGetPtr, even if the user is getting the pointer to a cleared data pointer that has not been set or inserted. DynArrayGetPtr is also used during binary searches, and hence that function will modify maxCount if the binary search expands to empty elements in the list. This is necessary because client functions can modify the contents of the element via DynArrayGetPtr, because they have direct access to the data. Typical users of dynamic arrays will not call DynArrayGetPtr in such a manner as to modify maxCount.

In summary, entries is the amount of space allocated by the array, and maxCount is the number of elements set or inserted into the array.

When memory is allocated for entries in the array, via DynArrayInsert, DynArrayNew, or DynArrayExpand, it is initialized to 0.

```c
#ifndef DYNARRAY_INCLUDED
#define DYNARRAY_INCLUDED
#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef OSHRIP_INCLUDED
#include <osheap.h>
#endif
```
Common #defines and typedefs

Dynamic Array

This data structure is the dynamic array data structure. A dynamic array created and manipulated is simply a pointer to this data structure that is passed to the dynarray functions. Accessing the fields in this data structure is possible, but care should be taken as changing their values could have drastic side effects. This data structure is sometimes referred to as the array header.

typedef struct DYNARRAY {
    OS_HEAP_ID heap;     // heap used for allocations
    U16 entries;         // total # entries in the array
    U16 elemSize;        // size in bytes of individual elements
    P_U8 pData;          // pointer to the array of values
    U16 maxCount;        // Max index accessed in the array via
                         // DynArraySet, DynArrayGetPtr, DynArrayBinSearch
                         // Updated when inserting, deleting, or contracting
                         // array.
} DYNARRAY, *P_DYNARRAY;

Public Functions

DynArrayNew

Allocates a new dynamic array. Passes back the P_DYNARRAY header.

Returns STATUS.

Function Prototype

STATUS DynArrayNew(
    OS_HEAP_ID heap,     // In: heap for memory allocation
                         // NULL=>osProcessHeapId
    U16 elementSize,     // In: size in bytes of each array element
    U16 startSize,       // In: initial array size in number of elements
    U16 extraHeader,     // In: additional bytes to allocate in the header
    P_DYNARRAY *ppArray);  // Out: pointer to the header pointer

Comments

Allocates memory for the array header, the P_DYNARRAY that is passed to the dynarray functions. Allocates memory for the initial elements in the array. Parameters include: the allocation heap to perform memory allocations, the size of an individual element, the initial size of the array (pArray->elements will the same as this value when this function returns), and any extra space to be allocated in the P_DYNARRAY pointer. This space can be used by clients to store list-wide information or flags. Passes back a pointer to the array data structure, P_DYNARRAY.

DynArrayFree

Destroys the dynamic array and frees memory used by the array.

Returns STATUS.

Function Prototype

STATUS DynArrayFree(
    P_DYNARRAY pArray);  // In: array header. Will be freed.

Comments

Will free all memory allocated by the array to store the header information and the elements. Does not do anything with the entries in the array.
**DynArrayExpand**

Expands the array by the specified number of entries.

Returns STATUS.

**Function Prototype**

```c
STATUS DynArrayExpand(
    P_DYNARRAY pArray,   // In: array header
    UI6 add);            // In: Number of elements to add
```

**Comments**

Expands the array by a number of entries, updating pArray->entries, the returned value of calling DynArrayCount, and the reallocation of pArray->pData to be equal to pArray->entries * pArray->elementSize. This function is called when calling DynArrayInsert to add space for one more entry. It is also called in DynArraySet if the index is greater than the number of entries.

**See Also**

DynArraySet

---

**DynArrayContract**

Contracts the array by the number of entries.

Returns STATUS.

**Function Prototype**

```c
STATUS DynArrayContract(
    P_DYNARRAY pArray,   // In: array header
    UI6 truncate);       // In: Number of elements to free
```

**Comments**

Will contract the number of entries in the array, and free the memory associated with those entries. Will resize the amount of memory allocated by the array pArray->pData to be pArray->entries * pArray->elementSize. If the maxCount (return code of DynArrayCount) is greater than the new number of entries allocated, maxCount will be adjusted. Called from DynArrayDelete to contract the array when deleting items.

**See Also**

DynArrayDelete

---

**DynArrayGet**

Passes back the index'th element in the array.

Returns STATUS.

**Function Prototype**

```c
STATUS DynArrayGet(
    P_DYNARRAY pArray,   // In: array header
    UI6 index,           // In: element index
    P_UNKNOWN pData);    // Out: pointer to data buffer. Must be elementSize.
```

**Comments**

Will pass back the contents of the index'th element in the array. Will copy the memory of size elementSize containing the data for the element into pData. It is the clients responsibility to ensure that this data pointer is large enough.

---

**DynArraySet**

Sets the index'th item to the given value. Update maxCount.

Returns STATUS.

**Function Prototype**

```c
STATUS DynArraySet(
    P_DYNARRAY pArray,   // In: array header
    UI6 index,           // In: element index
    P_UNKNOWN pData);    // In: pointer to data or NULL for zero fill
```
Sets the contents of the index'th item to the given value. Will copy the contents of the pData pointer to the memory for the index'th element in the array. It is the client's responsibility to ensure that pData is correct. If index is greater than maxCount, it will update maxCount. If the index is greater than the number of entries, the array is expanded via DynArrayExpand to be large enough. Called from DynArrayInsert to set the value of the new index.

See Also
DynArrayInsert

**DynArrayGetPtr**

Passes back a pointer to the index'th element in the array.

Returns STATUS.

**Function Prototype**

```c
STATUS DynArrayGetPtr(
    P_DYNARRAY pArray,   // In: array header
    U16 index,           // In: element index
    PP_UNKNOWN pData);   // Out: pointer to data buffer
```

**Comments**
Will pass back the direct pointer to the index'th element in the dynamic array. Care should be taken when accessing this pointer, as it is memory that is allocated and managed by the array. Accessing the data in this manner WILL cause the maxCount to be increased if maxCount < index. This function is called during a binary search via DynArrayBinSearch. Hence that function could modify maxCount.

See Also
DynArrayBinSearch

**DynArrayInsert**

Inserts a new element in the array.

Returns STATUS.

**Function Prototype**

```c
STATUS DynArrayInsert(
    P_DYNARRAY pArray,   // array header
    U16 index,           // element index
    P UNKNOWN pData      // new data to insert or NULL
);
```

**Comments**
The new element is indexed by index. If the array is not big enough, will expand the array appropriately. Elements are copied from the index'th location to the next location.

See Also
DynArrayExpand

**DynArrayDelete**

Deletes the index'th element from the array.

Returns STATUS.

**Function Prototype**

```c
STATUS DynArrayDelete(
    P_DYNARRAY pArray,   // array header
    U16 index            // element index to delete
);
```

**Comments**
Will delete the index'th element from the array. If index is > entries, will return stsOK and do nothing. Will move all elements greater than the index down by one in the array. Will adjust maxCount if necessary. Will call DynArrayContract with parameter of one.
DynArrayCount

Passes back the number of entries allocated in the array.

Returns STATUS.

Function Prototype

```c
STATUS DynArrayCount(
    P_DYNARRAY pArray, // In: array header
    P_U16 pCount); // Out: pointer to the count
```

Comments
Passes back the number of entries allocated in the array. This number is the amount of space allocated, and not the number of items stored in the array. That value is returned by DynArrayMax.

DynArrayMax

Passes back the highest index stored.

Returns STATUS.

Function Prototype

```c
STATUS DynArrayMax(
    P_DYNARRAY pArray, // In: array header
    P_U16 pMax); // Out: pointer to the max index
```

Comments
Will return the highest index stored via DynArraySet or DynArrayGetPtr, plus one. This is the "maxCount" field, and is used to indicate the highest array entry that has a valid value.

DynArrayElemSize

Passes back the size, in bytes, of each element.

Returns STATUS.

Function Prototype

```c
STATUS DynArrayElemSize(
    P_DYNARRAY pArray, // In: array header
    P_U16 pSize); // Out: pointer to the size
```

Comments
Passes back the size allocate in the array for each element. The pArray->pData size will be the value passed back by this function * the value passed back by DynArrayCount.

DynArrayBinSearch

Performs a binary search on the array.

Returns STATUS.

typedef S16 FunctionPtr(P_BIN_PROC)(P_UNKNOWN, P_UNKNOWN);

Arguments
typedef struct DYNARRAY_SEARCH {
    P_UNKNOWN pData; // In: Pointer to search data.
    U16 start; // In: start index, Out: first occurrence
    U16 stop; // In: end index, Out: last occurrence
    P_BIN_PROC pCompare; // In: routine to perform comparisons
    S16 result; // Out: 0 if equal, -1 less, 1 greater
} DYNARRAY_SEARCH, *P_DYNARRAY_SEARCH;

Function Prototype

```c
STATUS DynArrayBinSearch(
    P_DYNARRAY pArray, // In: array header
    P_DYNARRAY_SEARCH pSearch); // In: search data
```

Comments
Performs a binary search on the array. Assumes that the array is "sorted" from lowest value to highest value. Will access the value of data in the array via DynArrayGetPtr. Hence care should be taken when using the data in the comparison callback routine.

Return Value
stsNoMatch No matching data could be found within the range.
DynArrayGetPtr

P_BIN_PROC is the comparison routine callback. Will be called to test items. Called parameters containing pointers to an element in the array, and a pointer to a test 'element' to check for comparison. Returns 0 for equal, -1 for less, 1 for greater.

DYNARRAY_SEARCH is the parameter into the DynArrayBinSearch function. Takes the search data pointer to locate, a starting index into the array to search, a stopping index into the array to search, and a comparison callback function to test the data pointer against elements in the array. If result is 0, passes back the starting and ending indices that match. If result is -1, the target data pointer was less than both the starting and ending indices searched. Similarly, if result is 1, the target data pointer was greater than both indices.
This file contains the API definition for GO's modified binary search. The function described in this file is contained in MISC.LIB.

The fundamental difference between this binary search and the search that is part of the standard runtime is that if the search fails, this search indicates where a searched for element should have been located, thereby aiding insertion of a new element.

```c
#ifndef GOSEARCH_INCLUDED
#define GOSEARCH_INCLUDED
$Revision: 1.6 $
#ifndef GO_INCLUDED
#include <go.h>
#endif

typedef P_UNKNOWN (CDECL *ACCESS_FUNC)(
  const P_UNKNOWN, // context
  const U32); // index

typedef int (CDECL *COMPARE_FUNC)(
  const P_UNKNOWN, // context
  const P_UNKNOWN, // key1
  const P_UNKNOWN); // key2

GO INCLUDED

binarySearch performs a binary search on a sorted, indexed data structure.

Performs a binary search for specified key within dataStructure.

Returns STATUS.

STATUS EXPORTED binarySearch(
  const P_UNKNOWN key,
  const P_UNKNOWN dataStructure,
  const U32 count,
  COMPARE_FUNC compare,
  ACCESS_FUNC access,
  const U16 itemSize,
  PP_UNKNOWN pFoundOrInsert,
  P_U32 pIndex);

binarySearch performs a binary search on a sorted, indexed data structure.

The caller provides an count of the number of items in the data structure, an access function that translates an item index into an address for the item key, and a comparison function to compare a pair of keys.

A detailed description of the parameters follows.

key   key to search for.

dataStructure handle of data structure to search.

count number of items in data structure.

compare pointer to comparison function (see below).

access pointer to access function (see below). If Nil, dataStructure is assumed to be the address of a sorted, contiguous array of items (itemSize bytes long) with the item key at the start of each item.
**itemSize**  size of item in bytes (only used if access is Nil).

**pFoundOrInsert**  pointer to key (see below).

**pIndex**  pointer to index (see below).

The access function is provided with the client provided **dataStructure** and a (zero origin) index. It is responsible for returning the key for the indexed item. This key must be comprehensible to the comparison function, but is otherwise uninterpreted by the search.

The comparison function is responsible for actually comparing two keys, and returning values as follows.

\[
< 0: \text{ when } key_1 < key_2, \\
== 0: \text{ when } key_1 == key_2, \\
> 0: \text{ when } key_1 > key_2.
\]

**key1** is always the key originally passed to **binarySearch** as a parameter. **key2** is always a key generated from **dataStructure** by the access function.

When **binarySearch** returns, *pFoundOrInsert contains either:

the first occurrence of the desired key, if it was found; or
NULL, if the key was not found but was greater than the keys of all the items in **dataStructure**; or
the first key larger than the desired key.

In addition, when **binarySearch** returns, *pIndex contains either count, if *pFoundOrInsert == NULL, or the index used to access the key returned via *pFoundOrInsert.

The return value is:

**stsOK**  if desired key located, or

**stsNoMatch**  if desired key not located
This file contains the Personal Dictionary Class API. This class contains methods that maintain an ordered ASCII list of words and can produce a compressed list (called the template), which is specially organized for use with handwriting translation software.

clsPDict inherits from clsObject.

thePersonalDictionary is a well known instance of clsPDict.

The word list maintained by thePersonalDictionary is used by default whenever spelling-assisted handwriting translation is performed.

Common typedefs

Personal Dictionary Metrics

This structure is used in conjunction with msgPDictGetMetrics to get two very important parameters of a personal dictionary: the number of words in it and a pointer to the compressed template. The word count is useful for a variety of things, but the compressed template is valuable because it can be used directly in the pTemplate field of a translator object (see xlate.h)

typedef struct PDICT_METRICS {
    U16 wordCount; // number of words in the personal dictionary (RO)
    P_UNKNOWN pXTemplate; // pointer to compressed template
} PDICT_METRICS, * P_PDICT_METRICS;

Personal Dictionary New Structs

typedef struct PDICT_NEW_ONLY {
    IM_HANDLE handle; // if objNull then use current pdict.
    U32 spare;
} PDICT_NEW_ONLY, * P_PDICT_NEW_ONLY;

typedef struct PDICT_NEW {
    OBJECT_NEW_ONLY object;
    PDICT_NEW_ONLY pdict;
} PDICT_NEW, * P_PDICT_NEW;
This structure is used for converting a word index into a word and vice versa. (That is, for example, to get word #5 from the PDict or to find out which word number in the PDict "PenPoint" is.)

```c
typedef struct PDict_NUM_WORD {
    U16 number;
    P CHAR pWord;
} PDict_NUM_WORD, * P_PDict_NUM_WORD;
```

### Messages

**msgPDictGetMetrics**

Gets a copy of the personal dictionary metrics structure.

Takes P_PDict_METRICS, returns STATUS.

```c
#define msgPDictGetMetrics MakeMsg(clsPDict, 1)
```

**msgPDictEnumerateWords**

Fills a list of pointers to strings with pointers to all the words in the personal dictionary.

Takes PP_CHAR, returns STATUS.

```c
#define msgPDictEnumerateWords MakeMsg(clsPDict, 2)
```

The pArgs must be the address of the base of an array of pointers to filled in. This array must have an entry for every word in then the dictionary plus one for the final null (get the metrics to out how many words are in the PDict. The words will be in ASCII sequence, and because the pointers all point to an internal structure, no memory is allocated. N.B. you must treat this as strictly read-only!

**msgPDictAddWord**

Adds a word to the personal dictionary.

Takes P_PDict_NUM_WORD, returns STATUS.

```c
#define msgPDictAddWord MakeMsg(clsPDict, 3)
```

The routine SpellAddWord(), defined in spell.h, is a better way for clients to add words to the Personal Dictionary, since it has a API, strips excess punctuation, checks for duplicates, etc.

**msgPDictAddWord** adds the string from the PDict_NUM_WORD structure, the zero-based offset of the new word in the personal, and passes back that offset in the number component of PDict_NUM_WORD structure.

Although the ASCII representation of the Personal Dictionary is immediately, the compressed template is not rebuilt until the time **msgPDictUpdateTemplate** is called. Handwriting Translation this automatically when it needs the template, but spelling does.
**msgPDictDeleteWord**

Deletes a word from the personal dictionary.
Takes `P_PDICT_NUM_WORD`, returns STATUS.

```c
#define msgPDictDeleteWord MakeMsg(clsPDict,4)
```

**Message Arguments**

```c
typedef struct PDICT_NUM_WORD {
    U16 number;
    P_CHAR pWord;
} PDICT_NUM_WORD, *P_PDICT_NUM_WORD;
```

The reverse of `msgPDictAddWord`, this message removes the word from personal dictionary and passes back the zero-based offset of the where it formerly was.

Like `msgPDictAddWord`, this only affects the ASCII representation of Personal Dictionary. The next handwriting translation operation rebuild the template, but if you need it built before that (for, to change the behavior of spelling), send.

**msgPDictNumToWord**

Locates a word in the personal dictionary by index number, passing back the word at that offset.
Takes `P_PDICT_NUM_WORD`, returns STATUS.

```c
#define msgPDictNumToWord MakeMsg(clsPDict,5)
```

**Message Arguments**

```c
typedef struct PDICT_NUM_WORD {
    U16 number;
    P_CHAR pWord;
} PDICT_NUM_WORD, *P_PDICT_NUM_WORD;
```

Words are indexed in ASCII collating sequence from zero.

**msgPDictFindWord**

Checks if a word is in the personal dictionary.
Takes `P_CHAR`, returns STATUS.

```c
#define msgPDictFindWord MakeMsg(clsPDict,6)
```

**Message Arguments**

```c
```

`stsOK` means it was found; `stsFailed` means it was not.

**msgPDictDeleteNum**

Locates a word in the personal dictionary by index number and deletes the word at that offset.
Takes `P_PDICT_NUM_WORD`, returns STATUS.

```c
#define msgPDictDeleteNum MakeMsg(clsPDict,7)
```

**Message Arguments**

```c
typedef struct PDICT_NUM_WORD {
    U16 number;
    P_CHAR pWord;
} PDICT_NUM_WORD, *P_PDICT_NUM_WORD;
```

Words are indexed in ASCII collating sequence from zero. The number the word to delete is the number field from the `PDICT_NUM_WORD`; the actual word deleted is passed back in `pWord`. (This MUST set to point to something by the caller! Max size is +1. Setting `pWord` to Nil(`P_CHAR`) passes nothing back.)
msgPDictWordToNum

Given a word, computes its offset within the personal dictionary.

Takes P_PDICT_NUM_WORD, returns STATUS.

#define msgPDictWordToNum MakeMsg(clsPDict, 8)

typedef struct PDICT_NUM_WORD {
    U16 number;
    P_CHAR pWord;
} PDICT_NUM_WORD, *P_PDICT_NUM_WORD;

Words are counted from zero in ASCII collating sequence.

msgPDictUpdateTemplate

Recomputes the compressed template from the word list and updates the pointer.

Takes P_UNKNOWN, returns STATUS.

#define msgPDictUpdateTemplate MakeMsg(clsPDict, 9)

When the ASCII form of the personal dictionary is modified, the template is not automatically modified. Since compression be time consuming, this is deferred until it is absolutely. This routine is called by Handwriting Translation at the end of every translation.

If the current template is not out of date, this just copies old value into argument.

Miscellaneous

Base of the template of the Personal Dictionary. Handwriting needs to be able to get at this very quickly, so it's as an exported global variable to allow it to avoid the.

extern P_UNKNOWN PASCAL pPDictBase;
#define hlpPDApBackground MakeTag(clsPDApp, 1)
This file contains the API definition for class clsPowerButton.

clsPowerButton inherits from clsObject.

"thePowerButton" is a well known object that provides notification when the machine is turned off and on.

#ifndef POWER_INCLUDED
#define POWER_INCLUDED
#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif

## Messages

### msgPBMachinePoweringUp
Notifies clients that the machine is powering up.


```c
#define msgPBMachinePoweringUp MakeMsg(clsPowerButton, 1)
```

**Comments**

Sent by the system to observers of thePowerButton. Indicates that the machine is in the process of powering up.

The system will not power up until all observers of thePowerButton are notified. The system will wait until the notification message has completed for each client.

### msgPBMachinePoweringDown
Notifies clients that the machine is powering down.


```c
#define msgPBMachinePoweringDown MakeMsg(clsPowerButton, 2)
```

**Comments**

Sent by the system to observers of thePowerButton. Indicates that the machine is in the process of powering down.

Most applications do not need to observe the power button, since theSystem sends the appropriate messages to all applications and services when the machine powers down.

The system will not power down until all observers of thePowerButton are notified. The system will wait until the notification message has completed for each client.
This file contains the API definition for class clsPowerMgr.

clsPowerMgr inherits from clsObject.

"thePowerMgr" is a well known object that provides system power management.

```c
#ifndef POWERMGR_INCLUDED
#define POWERMGR_INCLUDED
#endif
#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif
```

### Common #defines and typedefs

```c
typedef U16 PM_POWER_STATE;
typedef U16 PM_POWER_METRICS, *P_PM_POWER_METRICS;
```

### Messages

#### msgPMSetPowerState

Sets the machine power state.

Takes PM_POWER_STATE, returns nothing.

```c
#define msgPMSetPowerState MakeMsg(clsPowerMgr, 1)
#define pmStandby flag0 // power down to stand by state.
#define pmPowerOff flag1 // power down to complete off.
#define pmForceBoot flag2 // Force a cold boot on the machine
#define pmQuickestPowerOnState (pmStandby | pmPowerOff)
    // quickest allowable power on state.
```

Initiates the powering down of the machine. The machine can be powered down in "standby" state (i.e. RAM is maintained, but the rest of the system is shut down) or "complete off" state.

Powering down the machine will force all data to be saved to disk (if applicable) and will notify all observers of the power button of this event (see power.h).

If the client is unfamiliar with the hardware configurations, use pmQuickestPowerOnState. This mode will power down the machine to the state that will cause the machine to come up in the quickest possible time.

pmForceBoot will force the machine to reset and cold boot the software. Caution: Under certain configurations this may cause loss of data!!! Specifically, under a RAM only configuration, all the contents of RAM will be lost.
msgPMGetPowerMetrics
Passes back the machine power information.
Takes P_PM_POWER_METRICS, returns STATUS.

#define msgPMGetPowerMetrics
#define pmStandbyPowerSupported
#define pmNoPowerSupported
#define pmStandbyButtonSupported
#define pmChargerConnectedSupported
#define pmIdleStateSavesPower
#define pmChargerConnected
#define pmSomeDevicePoweredDown
#define flag0 // only ram is alive
#define flag1 // everything is off
#define flag2 // power button usage
#define flag3 // power connection
#define flag4 // idle = low power state?
#define flag5 // is power connected?
#define flag15 // something is off

Comments
Passes back information on what power states are supported on this machine. The machine can support either 1) standby or 2) power off or 3) both or 4) none. Setting none indicates that the software is unable to change the power state of the machine.
This message also returns information on the charger and whether a standby button is supported.

msgPMDevicesPowerOn
Turns power on to all devices in the system.
Takes nothing, returns STATUS.

#define msgPMDevicesPowerOn

msgPMDevicePoweringOn
Notifies observers that a device is powering up.

#define msgPMDevicePoweringOn

Comments
Sent by the system to observers of thePowerMgr. Indicates that a device (specified by MIL logical Id) is powering up.

msgPMDevicePoweringOff
Notifies observers that a device is powering down.

#define msgPMDevicePoweringOff

Comments
Sent by the system to observers of thePowerMgr. Indicates that a device (specified by MIL logical Id) is powering off.

msgPMAllDevicesPoweredOn
Notifies observers that all devices have powered up.

#define msgPMAllDevicesPoweredOn

Comments
Sent by the system to observers of thePowerMgr.
DATE_FIELD_NEW, API1:586
DATE_FIELD_NEW_ONLY, API1:586
DATE_FIELD_STYLE, API1:585–586
Dbg, API1:48
DbgFlag, API1:48
DbgFlagGet, API1:50
DbgFlagSet, API1:49
Debug, API1:49
Debugger, API1:148
Decode31, API1:229
DIALENV_AREA_CITY, API2:383
DIALENV_BUILD_DIALSTR, API2:384
DIALENV_COUNTRY, API2:383
DIALENV_DIAL_STRING, API2:384
DIALENV_ENVIRONMENT, API2:384
DIALENV_FIELD_NEW, API2:385–386
DIALENV_INTL_ACCESS, API2:384
DIALENV_LONG_DIST, API2:384
DIALENV_MACRO_CODE, API2:384
DIALENV_MACRO_IDS, API2:386
DIALENV_NEW, API1:385
DIALENV_OPTCARD_NEW, API2:387–388
DIALENV_OPTCARD_NEW_ONLY, API2:387
DIALENV_OUTSIDE_LINE, API1:383
DIALENV_SUFFIX, API2:384
DIALENV_TELEPHONE_NUMBER, API2:384
DirIdGetParent, API2:95
DPrintf, API1:49
DumpRect, API1:239
DV_GET_OPEN_VOLS, API2:211
DV_NEW, API2:209
DV_NEW_ONLY, API2:208
DV_STYLE, API2:208, API2:210
DynArray, API2:642
DynArrayGet, API2:643
DynArrayGetPtr, API2:644
DynArrayInsert, API2:644
DynArrayMax, API2:645
DynArrayNew, API2:642
DynArraySet, API2:643
DynResId, API2:493
EMBEDDED_WIN_BEGIN_MOVE_COPY, API1:161
EMBEDDED_WIN_GET_DEST, API1:164
EMBEDDED_WIN_INSERT_CHILD, API1:165
EMBEDDED_WIN_METRICS, API1:174
EMBEDDED_WIN_MOVE_COPY, API1:162–163
EMBEDDED_WIN_MOVE_COPY_OK, API1:163
EMBEDDED_WIN_NEW, API1:174
EMBEDDED_WIN_NEW_ONLY, API1:174
EMBEDDED_WIN_PROVIDE_ICON, API1:162
EMBEDDED_WIN_SHOW_CHILD, API1:166
EMBEDDED_WIN_STYLE, API1:173
EMBEDDED_PRINT_INFO, API1:205
Encode31, API1:229
Enum16, API1:55
Enum32, API1:55
Even, API1:56
EXCL_VOL_ACCESS, API2:98
EXPORT_DOC, API2:216
EXPORT_FORMAT, API2:216–217
EXPORT_LIST, API2:216
FIELD_ACTIVATE_POPUP, API1:395
FIELD_CREATE_POPUP, API1:396
FIELD_NEW, API1:392
FIELD_NEW_ONLY, API1:392
FIELD_NOTIFY, API1:391, API1:399
FIELD_STYLE, API1:391, API1:393
FIELD_XLATE, API1:392
FIM_FIND_ID, API2:535
FIM_GET_INSTALLED_ID_LIST, API2:536
FIM_GET_NAME_FROM_ID, API2:535
FIM_GET_SET_ID, API2:534
FIM_NEW, API2:534
FIM_PRUNE_CONTROL, API2:536
FindListItem, API2:78
FindListItemX, API2:77
FIXED_FIELD_NEW, API1:588
FIXED_FIELD_NEW_ONLY, API1:588
FIXED_FIELD_STYLE, API1:587–588
FlagClr, API1:56
FlagOff, API1:56
FlagOn, API1:56
FlagSet, API1:56
FlapNew, API1:391
FONTLB_NEW, API1:401–402
FONTLB_NEW_ONLY, API1:401
FONTLB_STYLE, API1:401–402
FRAME_NEW, API1:406–407
FRAME_NEW_ONLY, API1:406
FRAME_STYLE, API1:405, API1:408–409
FRAME_ZOOM, API1:405, API1:413
FS_CHANGE_INFO, API2:68
FS_CONNECT_VOL, API2:97
FS_DIR_NEW_MODE, API1:58
FS_DISCONNECT_VOL, API1:97
FS_EXCL_VOL_ACCESS, API2:98
FS_EXIST, API1:57
FS_FILE_NEW_MODE, API1:58
FS_FLAT_LOCATOR, API1:56
FS_GET_PATH, API1:63
FS_GET_PATH_MODE, API1:58
FS_GET_SET_ATTR, API1:63
FS_GET_VOL_METERICS, API1:61
FS_INSTALL_VOL, API1:96
FS_LOCATOR, API1:56, API1:69
FS_MAKE_NATIVE, API1:67
FS_MOVE_COPY, API1:64–65
FS_MOVE_COPY_EXIST, API1:66
FS_MOVE_COPY_MODE, API1:58
FS_MOVE_COPY_NOTIFY, API1:66
FS_NEW, API1:59–60
FS_NEW_ONLY, API1:59
FS_NODE_EXISTS, API1:62
FS_NODE_FLAGS, API1:56
FS_NOTIFY_OP, API1:66
FS_NOTIFY_RTN_INFO, API1:66
FS_NOTIFY_TIME, API1:65
FS_REMOVE_VOL, API1:97

INDEX 659
msgSMQueryUnlock, API2:619
msgSMRelease, API2:615
msgSMSave, API2:619
msgMSMSetOwner, API2:616
msgMSMSetOwnerNoVeto, API2:620
msgSMUnbind, API2:615
msgSpMgrGesture,
msgSpMgrFindMisspelling,
msgSpMgrCreateContext,
msgSpMgrAcceptMisspelling,
msgSpPaperSetSizes, API1:725
msgSpPaperSetScribble,
msgSpPaperSetFlags,
msgSpPaperSetCellMetrics,
msgSpPaperLocate,
msgSpPaperGetSizes,
msgSpPaperGetScribble,
msgSpPaperGetFlags, API 1:723
msgSpPaperGetCellMetrics,
msgSpPaperDeleteStrokes,
msgSpPaperComplete, API1:726
msgSpPaperAbort,
msgSMRelease, API2:615
msgSRInvokeSearch,
msgSRGetChars,
msgSPaperSetTranslator,
msgSPaperGetXlateDataAndStrokes,
msgSPaperGetXlateData,
msgSPaperGetTranslator,
msgSMSetOwnerNoVeto,
msgSMSetOwner, API2:616
msgSMSave, API2:619
msgSMQueryUnlock, API2:619
msgSMRelease, API2:615
msgSMSave, API2:619
msgMSMSetOwner, API2:616
msgMSMSetOwnerNoVeto, API2:620
msgSMUnbind, API2:615
msgSpMgrGesture,
msgSpMgrFindMisspelling,
msgSpMgrCreateContext,
msgSpMgrAcceptMisspelling,
msgSpPaperSetSizes, API1:725
msgSpPaperSetScribble,
msgSpPaperSetFlags,
msgSpPaperSetCellMetrics,
msgSpPaperLocate,
msgSpPaperGetSizes,
msgSpPaperGetScribble,
msgSpPaperGetFlags, API 1:723
msgSpPaperGetCellMetrics,
msgSpPaperDeleteStrokes,
msgSpPaperComplete, API1:726
msgSpPaperAbort,
msgSMRelease, API2:615
msgSRInvokeSearch,
msgSRGetChars,
msgSPaperSetTranslator,
msgSPaperGetXlateDataAndStrokes,
msgSPaperGetXlateData,
msgWinDevBindPixelmap, API 1:286
msgWinDelta, API 1:285
msgWinCopyRect, API 1:285, API 1:311
msgWinDevPrintPage,
msgWinDevSizePixelmap, API 1:286,
msgWinBeginRepaint, API 1:284,
msgWinCleanRect, API 1:311
msgWinEndRepaint, API 1:31
msgWinDumpTree, API 1:318
msgWinDevPrintPage,
msgWinDevSizePixelmap, API 1:286,
msgWinBeginRepaint, API 1:284,
msgWinCleanRect, API 1:311
msgWinEndRepaint, API 1:31
msgWinDumpTree, API 1:318
msgWinDevPrintPage,
msgWinDevSizePixelmap, API 1:286,
msgWinBeginRepaint, API 1:284,
msgWinCleanRect, API 1:311
msgWinEndRepaint, API 1:31
msgWinDumpTree, API 1:318
msgWinDevPrintPage,
msgWinDevSizePixelmap, API 1:286,
msgWinBeginRepaint, API 1:284,
msgWinCleanRect, API 1:311
msgWinEndRepaint, API 1:31
msgWinDumpTree, API 1:318
msgWinDevPrintPage,
msgWinDevSizePixelmap, API 1:286,
msgWinBeginRepaint, API 1:284,
OSTaskSharedHeapId, API2:156
OSTaskMemInfo, API2:141
OSTaskPriorityGet, API2:141
OSTaskTerminate, API2:140
OSThisShareApp, API2:152
OSThisProcess, API2:141
OSThisTask, API2:141
OSThisWinDev, API2:150
OSTimeAsyncSema, API2:150
OSTimerIntervalSema, API2:150
OSTimerStop, API2:150
OSTimerTransactionValid, API2:150
OSTone, API2:152
OSVirToPhys, API2:169
OSWinDevPose, API2:153
OutRange, API1:56

*P_BROADCAST_ADDR, API2:420
PAGE_NUM_NEW, API1:515
PAGE_NUM_NEW_ONLY, API1:515
PAGE_NUM_STYLE, API1:515–516
PAPER_CONFIG, API1:152–153
PDICT_METRICS, API2:649–650
PDICT_NEW, API2:649
PDICT_NEW_ONLY, API2:649
PDICT_NUM_WORD, API2:650–652
PEN_DATA, API1:708
PEN_METRICS, API1:708–709
PEN_STROKE, API1:708
PEN_TIP_STATE_TYPE, API1:707
PenCurrentStandardData, API1:710
PenExpander, API1:709
PenStrokeRetrace, API1:709
PenStrokeUnpack16, API1:710
PenStrokeUnpack32, API1:710
PIC_SEG_ARC_RAYS, API1:244
PIC_SEG_ELLIPSE, API1:243
PIC_SEG_FONT_STYLE, API1:242
PIC_SEG_GRAPHIC, API1:242, API1:247, API1:249–250
PIC_SEG_HIT_LIST, API1:248
PIC_SEG_LIST, API1:247
PIC_SEG_NEW, API1:245
PIC_SEG_NEW_ONLY, API1:244
PIC_SEG_OBJECT, API1:244, API1:246
PIC_SEG_PAINT, API1:242
PIC_SEG_PAINT_OBJECT, API1:247
PIC_SEG_PLINE_TYPE, API1:242
PIC_SEG_POLYGON, API1:243
PIC_SEG_POLYLINE, API1:243
PIC_SEG_RECT, API1:243
PIC_SEG_SPLINE, API1:243, API1:246
PIC_SEG_TEXT, API1:243
PIM_NEW, API2:567
PIX_DEV_METRICS, API1:322
PIX_DEV_ORIENTATION, API1:322
POINT, API1:738
POPUP_CHOICE_NEW, API1:517–518
POPUP_CHOICE_NEW_ONLY, API1:517
POPUP_CHOICE_STYLE, API1:517–518
PPORT_METRICS, API2:451
PPORT_NEW, API2:453
PPORT_STATUS, API1:452
PPORT_TIME_DELAYS, API1:452–453
PREF_CHANGED, API2:482
PREF_SYSTEM_FONT, API2:477
PREF_SYSTEM_FONT_INFO, API2:483
PREF_TIME_INFO, API2:481
PREF_TIME_MODE, API2:481
PREFS_NEW, API2:482
PREFS_NEW_ONLY, API2:482
PrefsDateTime, API2:484
PrefsSysFontInfo, API2:483
PrefsTimeToDateTime, API2:484
PRFRAME_EXPAND, API1:201
PRFRAME_NEW, API1:199–200
PRFRAME_SEND, API1:200
PRINT_AREA, API1:209
PRINT_DATA, API1:208
PRINT_EMBEDDED_ACTION, API1:206,
API1:209–210
PRINT_HFDATA, API1:204
PRINT_MARGINS, API1:204
PRINT_METRICS, API1:205, API1:207–208
PRINT_PAGE, API1:206–207
PRINT_PROTOCOLS, API1:209
PRINT_SETUP, API1:204
PRINTABLE_AREA, API1:210–211
PRLAYOUT_METRICS, API1:213–214
PRLAYOUT_NEW, API1:213
PRLAYOUT_NEW_ONLY, API1:213
PRLAYOUT_PAGE, API1:214
PR_MARGIN_METRICS, API1:215
PR_MARGIN_NEW, API1:215
PR_MARGIN_NEW_ONLY, API1:215
PRN_ENUM_MODELS, API1:155
PRN_ENUM_MODE, API1:2494
PRN_FILE_NEW, API2:495
PRN_FILE_NEW_ONLY, API2:495
PRN_FIND, API2:496
PRN_INFO, API2:496
PRN_LIST_NEW, API2:504
PRN_LIST_NEW_ONLY, API2:504
PRN_NEW, API1:152
PRN_NEW_ONLY, API1:152
PRN_TEXTOUT, API1:156
PROGRESS_METRICS, API1:524,
API1:526–527
PROGRESS_NEW, API1:525
PROGRESS_NEW_ONLY, API1:524
PROGRESS_PROVIDE_LABEL, API1:528
PROGRESS_REGION, API1:524,
API1:527–528
PROGRESS_STYLE, API1:524, API1:526
PROGRESS_VIS_INFO, API1:528
PROTOCOL_ADDRESS, API2:419
PROTOCOL_INFO, API2:419
PutList, API2:78
PutListItem, API2:76

RATIONAL, API2:289
RC_INPUT, API2:485
RC_TAGGED_STRING, API2:486
RCAPP_GOTO_DOC, API1:218
RECT16, API1:234
Rect16Empty, API1:236
Rect16Intersect, API1:235
Rect16To32, API1:234
RECT32, API1:233
Rect32Empty, API1:236
Rect32EnclosesRect32, API1:235
Rect32Intersect, API1:235
Rect32To16, API1:234
RectInit, API1:234
RectRight, API1:234
RectTop, API1:234
REMOVE_PROTOCOL, API2:421
RemoveListItem, API2:78
RemoveListItemX, API2:77
RES_AGENT, API2:504
RES_ENUM, API2:503
RES_ENUM_MODE, API2:494
RES_FILE_NEW, API2:495
RES_FILE_NEW_ONLY, API2:495
RES_FIND, API2:496
RES_INFO, API2:496
RES_LIST_NEW, API2:504
RES_LIST_NEW_ONLY, API2:504
RES_NEW_MODE, API2:494
WIN, API2:455-456

WIN_METRICS, API1:565

WIN_NEW, API1:563

WIN_DELTA, API1:567

WIN_ALIGN, API1:564

WIN_STYLE, API1:564

WIN_METRICS, API1:562

WIN_SIZE, API1:566-567

WIN_SCALE, API1:233

SComposeText, API2:122

SCR_REMOVED_STROKE, API2:329

ScreenOnlyStringPrint, API2:148

SCROLL_WIN_ALIGN, API1:567

SCROLL_WIN_DELTA, API1:562,

API1:566-567

SCROLL_WIN_METRICS, API1:562,

API1:565

SCROLL_WIN_NEW, API1:563

SCROLL_WIN_SIZE, API1:566

SCROLL_WIN_STYLE, API1:561,

API1:563-564

SCROLLBAR_ACTION, API1:531

SCROLLBAR_NEW, API1:532

SCROLLBAR_NEW_ONLY, API1:532

SCROLLBAR_PROVIDE, API1:532,

API1:534

SCROLLBAR_SCROLL, API1:532-534

SCROLLBAR_STYLE, API1:531, API1:533

SEL_CHOICE_MGR_INFO, API1:540,

API1:542

SEL_CHOICE_MGR_NEW, API1:540-541

SEL_CHOICE_MGR_NEW_ONLY, API1:540

SEL_OWNERS, API1:291, API1:293-294

SEND_ENUM_ITEMS, API1:256

SEND_SRV_ADDR_WIN, API1:455-456

SEND_SRV_CONVERT_ADDR_DATA, API1:457

SEND_TYPE, API1:35

SetAttr, API1:76

SetSingleAttr, API1:76

SHADOW_TRUE, API1:543-44

SHADOW_FALSE, API1:543

SHADOW_STYLE, API1:543-44

SHORT_TX_FRAME, API1:421

SIM_GET_METRICS, API1:571

SIM_NEW, API1:571

SIO_BREAK_SEND, API1:463

SIO_BREAK_STATUS, API1:463

SIO_CONTROL_IN_STATUS, API1:462

SIO_CONTROL_OUT_SET, API1:462

SIO_DATA_BITS, API1:462

SIO_EVENT_HAPPENED, API1:466

SIO_EVENT_MASK, API1:461

SIO_EVENT_SET, API1:465-466

SIO_EVENT_STATUS, API1:465

SIO_FLOW_CONTROL_CHAR_SET, API1:463

SIO_FLOW_CONTROL_SET, API1:465

SIO_FLOW_TYPE, API1:465

SIO_INIT, API1:466

SIO_INPUT_BUFFER_STATUS, API1:464

SIO_LINE_CONTROL_SET, API1:462

SIO_METRICS, API1:466-467

SIO_NEW, API1:467

SIO_OUTPUT_BUFFER_STATUS, API1:464

SIO_PARITY, API1:462

SIO_RECEIVE_ERRORS_STATUS, API1:463

SIO_REPLACE_CHAR, API1:467

SIO_STOP_BITS, API1:462

SIZE16, API1:233

SIZE32, API1:233

SizeOf, API1:56

SM_ACCESS, API1:614

SM_BIND, API1:615-616

SM_CONNECTED_NOTIFY, API1:622

SM_FIND_HANDLE, API1:620

SM_GET_CHARACTERISTICS, API1:619

SM_GET_CLASS_METRICS, API1:621

SM_GET_OWNER, API1:616

SM_GET_STATE, API1:621

SM_NEW, API1:613

SM_NEW_ONLY, API1:613

SM_OWNER_NOTIFY, API1:621

SM_QUERY_LOCK, API1:618-619

SM_QUERY_UNLOCK, API1:619

SM_RELEASE, API1:615

SM_SAVE, API1:620

SM_SET_OWNER, API1:616, API1:620

SORT_BY, API1:186

SP_MGRGESTURE, API1:303-304

SP_TOKEN, API1:552

SPAPER_CELLMETRICS, API1:724

SPAPER_LOCATE, API1:725

SPAPER_NEW, API1:722

SPAPER_NEW_ONLY, API1:722

SPAPER_XDATA, API1:727

SPELL_CASE, API1:299

SPELL_CASE_CONTEXT, API1:299

SPELL_DICT_LIST, API1:299

SPELL_LIST, API1:299

SPELL_XLATE, API1:299

SpellAddToAnyDict, API1:301

SpellAddToDict, API1:301

SpellCheck, API1:300

SpellCorrect, API1:300

SpellCorrectWord, API1:301

SpellDictSelect, API1:300

SpellGetOptionsX, API1:300

SpellLineSetCase, API1:302

SpellOptionsX, API1:300

SpellWordSetCase, API1:301

SR_FLAGS, API1:305

SR_GET_CHARS, API1:306

SR_INVOKE_SEARCH, API1:308

SR_METRICS, API1:305, API1:308

SR_NEXT_CHARS, API1:305

SR_POSITION_CHARS, API1:307

SR_REPLACE_CHARS, API1:306

STAT_MENU_STYLE, API1:518

STATUS_GET, API1:422

StdError, API1:551

StdErrorRes, API1:553

StdioStreamBind, API1:82

StdioStreamToObject, API1:82

StdioStreamUnbind, API1:82

StdMsg, API1:551

StdMsgCustom, API1:553

StdProgressDown, API1:552

StdProgressUp, API1:552

StdSystemError, API1:551
INDEX 681

StdUnknownError, API1:550
STREAM_BLOCK_SIZE, API2:82
STREAM_NEW, API2:79
STREAM_READ_WRITE, API2:80
STREAM_READ_WRITE_TIMEOUT, API2:80-81
STREAM_SEEK, API2:81
STREAM_SEEK_MODE, API2:81
STRLB_NEW, API1:556
STRLB_NEW_ONLY, API1:555
STRLB_PROVIDE, API1:557
STRLB_STYLE, API1:555-556
STROBLNEW, API2:309
STROBLNEW ONLY, API2:309
Sts, API1:59
StsChk, API1:59
StsFailed, API1:59
StsJmp, API1:59
StsOK, API1:59
StsPrint, API1:59
StsRet, API1:59
StsWarn, API1:59
SVC_ADD_TO_MANAGER, API2:627
SVC_BIND, API2:604
SVC_CHARACTERISTICS, API2:606
SVC_CLASS_METRICS, API2:597
SVC_DEINSTALL_VETOED, API2:632
SVC_GET_FUNCTIONS, API2:632
SVC_GET_LIST, API2:628-629
SVC_GET_NAME, API2:633
SVC_GET_SET_CONNECTED, API2:603-604
SVC_GET_SET_METRICS, API2:626-627
SVC_GET_SET_MODIFIED, API2:601
SVC_GET_TARGET, API2:603
SVC_INIT_SERVICE, API2:598
SVC_LOAD_INSTANCE, API2:626
SVC_NEW, API2:600
SVC_NEW_ONLY, API2:600
SVC_OPEN_CLOSE, API2:605-606
SVC_OPEN_CLOSE_TARGET, API2:602
SVC_OWNED_NOTIFY, API2:623-625
SVC_REMOVE_FROM_MANAGER, API2:627
SVC_SET_TARGET, API2:603
SVC_STYLE, API2:599
SVC_TARGET, API2:597
SVC_TARGET_CHANGE_NOTIFY, API2:634
SVC_TERMINATE_VETOED, API2:630
SYS_BOOT_PROGRESS, API2:574
SYS_BOOT_STATE, API2:575
SYS_BOOT_TYPE, API2:574
SYS_CREATE_LIVE_ROOT, API2:576
SYS_GET_LIVE_ROOT, API2:576
SYS_IS_HANDLE_LIVE, API2:576
SYS_NEW, API2:575
SYS_NEW_ONLY, API2:575
SYS_SET_SECURITY_OBJECT, API2:577
SYSDC_ARC_RAYS, API1:274, API1:276
SYSDC_CACHE_IMAGE, API1:278
SYSDC_CAP, API1:262
SYSDC_CHAR_METRICS, API1:255
SYSDC_COPY_IMAGE, API1:279
SYSDC_EXTENTS16, API1:255
SYSDC_FONT_ATTR, API1:254
SYSDC_FONT_METRICS, API1:254
SYSDC_FONT_SPEC, API1:254
SYSDC_FONT_STATE, API1:260
SYSDC_FONT_WIDTHS, API1:254
SYSDC_IMAGE_FLAGS, API1:277
SYSDC_IMAGE_INFO, API1:277-278
SYSDC_VERIFY, API1:262
SYSDC_LINE, API1:262
SYSDC_MIX_PAT, API1:266
SYSDC_MIX_RGB, API1:265
SYSDC_MODE, API1:260
SYSDC_NEW, API1:258
SYSDC_NORMAL, API1:258
SYSDC_PAGE_TURN, API1:282
SYSDC_PIXEL, API1:275
SYSDC_PIXELS, API1:283
SYSDC_POLYGON, API1:274-275
SYSDC_RGB, API1:263
SYSDC_ROP, API1:261
SYSDC_SCREEN_SHOT, API1:283
SYSDC_STATE, API1:259-260
SYSDC_TEXT_OUTPUT, API1:255
SysDefFontId, API1:279
SysDefFontString, API1:280

INDEX 681

TA_ALIGN_BASE, API2:12
TA_CHAR_ATTRS, API2:12
TA_CHAR_MASK, API2:12
TA_MANY_TABS, API2:14
TA_PARA_ALIGN, API2:14
TA_PARA_ATTRS, API2:14
TA_PARA_MASK, API2:14
TA_TAB_LEADER, API2:13
TA_TAB_STOP, API2:13
TA_TAB_TYPE, API2:13
TAB_BAR_NEW, API1:574
TAB_BAR_STYLE, API1:573, API1:575
TAB_BUTTON_METRICS, API1:581-582
TAB_BUTTON_NEW, API1:581-582
TAB_BUTTON_NEW_ONLY, API1:581
Tag, API1:58
TagAdmin, API1:58
TagAndFlags, API1:58
TagFlags, API1:58
TagNum, API1:58
TagPaperStyle, API2:250
TBL_BEGIN_ACCESS, API2:317
TBL_BOOL_OP, API2:318
TBL_COL_DESC, API2:312
TBL_COL_GET_SET_DATA, API2:315
TBL_COL_NUM_FIND, API2:320
TBL_CONVERT_ROW_NUM, API2:320
TBL_CREATE, API2:313
TBL_END_ACCESS, API2:318
TBL_EXIST, API2:313
TBL_FIND_ROW, API2:319
TBL_FREE_BEAHVE, API2:313
TBL_GET_COL_DESC, API2:317
TBL_GET_SET_ROW, API2:315-316
TBL_GET_STATE, API2:317
TBL_HEADER, API2:316
TBL_LAYOUT_CONSTRAINT, API1:602
TBL_LAYOUT_COUNT, API1:603
TBL_LAYOUT_GRID, API1:606-607
TBL_LAYOUT_GRID_VALUE, API1:606
TBL_LAYOUT_INDEX, API1:605
TBL_LAYOUT_METRICS, API1:603-604
TBL_LAYOUT_NEW, API1:603
TBL_LAYOUT_SIZE, API1:603
TBL_LAYOUT_STYLE, API1:602, API1:605
TBL_NEW, API2:313
TBL_NEW_ONLY, API2:313
TBL_SEARCH_SPEC, API2:318
TBL_STATE, API2:317
TBL_STRING, API2:312
TBL_TYPES, API2:312
TD_METRICS, API2:18, API2:21
TD_NEW, API2:18-19
TD_NEW_ONLY, API2:18
TEACH_DATA, API1:770
TEACH_STATUS, API1:769
TEIsBlank, API2:4
TEIsLineBreak, API2:4
TEIsSentenceEnd, API2:4
TEIsSpecialPunct, API2:4
TEIsWord, API2:5
INDEX

TEXT_AFFECTED, API2:19, API2:29
TEXT_BUFFER, API2:18, API2:20-21
TEXT_CHANGE_ATTRS, API2:19, API2:23, API2:26
TEXT_COUNTER_CHANGED, API2:19, API2:29
TEXT_DIRECTION, API2:18
TEXT_EMBED_OBJECT, API2:15, API2:25
TEXT_ENUM_EMBEDDED, API2:15, API2:28
TEXT_FIELD_NEW, API1:591
TEXT_FIELD_NEW_ONLY, API1:591
TEXT_FIELD_STYLE, API1:591-592
TEXT_GET_ATTRIBS, API2:19, API2:25
TEXT_READ, API2:15, API2:26
TEXT_REPLACED, API2:19, API2:30
TEXT_SPAN, API2:18, API2:22-23
TEXT_SPAN_AFFECTED, API2:18
TEXT_WRITE, API2:15, API2:27
TextCreateTextScrollWin, API2:41
TextDeleteMany, API2:16
TextFindNextParaTab, API2:16
TextInitCharAttr, API2:17
TextInitCharMask, API2:17
TextInitParaAttr, API2:17
TextInitParaMask, API2:17
TextInsertOne, API2:16
TEXTIP_METRICS, API2:42-43
TEXTIP_SP, API2:43
TIFF_METRICS, API1:289
TIFF_NEW, API1:288
TIFF_NEW_ONLY, API1:287
TIFF_SAVE, API1:291-292
TIFF_SAVE_STYLE, API1:290
TIFF_STYLE, API1:287
TILE_LOCATOR, API1:293
TilePopUp, API1:293
TIMER_ALARM_INFO, API2:179
TIMER_ALARM_MODE, API2:179
TIMER_INTERVAL_INFO, API2:178
TIMER_NOTIFY, API2:179
TIMER_REGISTER_INFO, API2:177-178
TITLE_BAR_NEW, API1:579
TITLE_BAR_NEW_ONLY, API1:579
TITLE_BAR_STYLE, API1:579-580
TK_TABLE_ADD_AT, API1:598
TK_TABLE_ADD_SIBLING, API1:598
TK_TABLE_ENTRY, API1:594
TK_TABLE_INIT, API1:598
TK_TABLE_METRICS, API1:597
TK_TABLE_NEW, API1:595
TK_TABLE_NEW_ONLY, API1:594
TK_TABLE_STYLE, API1:593, API1:596
TkTableFillArrayWithFonts, API1:599
TkTableFreeArray, API1:599
TConstraint, API1:603
TOGGLE_TABLE_NEW, API1:621
TOGGLE_TABLE_NEW_ONLY, API1:621
TP_ACCEPT, API2:469
TP_BIND, API2:470
TP_CONNECT, API2:470
TP_LISTEN, API2:470
TP_NEW, API2:469
TP_NEW_ONLY, API2:469
TP_RECV, API2:470
TP_RECVFROM, API2:470
TP_SEND, API2:471
TP_SENDER, API2:471
TP_SENDER_TO, API2:471
TP_SENDER_FROM, API2:471
TRACK_METRICS, API1:612, API1:616-620
TRACK_NEW, API1:612, API1:615
TRACK_STYLE, API1:612, API1:615-616
TV_CARD_INDEX, API2:7
TV_CHAR_OPTION, API2:7
TV_EMBED_METRICS, API2:34,
 TV:37-38
TV_NEW, API2:41
TV_NEW_ONLY, API2:40
TV_PARA_OPTION, API2:8
TV_RESOLVE, API2:35, API2:39
TV_SCROLL, API2:36, API2:39
TV_SELECT, API2:36, API2:40
TV_STYLE, API2:33, API2:39-40
TV_META_DATA, API2:8
TV_MAKE_ITEM, API2:8
TV_MAKE_CARD_TAG, API2:8
TV_MAKE_CHAR_OPT_TAG, API2:8
TV_MAKE_PARA_OPT_TAG, API2:8
TV_MAKE_TAG, API2:8
TV_MAKE_VIEW_OPT_TAG, API2:8
TV_MAKE_XXX_TAG, API2:8
TX_DESC, API2:419
TX_FRAME, API2:420

INDEX

U, L, API2:131
UNDO_ITEM, API2:327-328, API2:330
UNDO_METRICS, API2:327, API2:329
UNDO_NEW, API2:328
UNDO_NEW_ONLY, API2:328
USER_BYTES, API2:365
USER_COLUMN_TYPE, API2:191

INDEX

VIEW_NEW, API1:219-220
VIEW_NEW_ONLY, API1:219
VCNCreate, API2:90
VDDelete, API2:91
VNDirPosDeleteAdjust, API2:91
VDUpdate, API2:90
VNFlush, API2:95
VNSet, API2:89
VNSetAttrInfo, API2:93-94
VNSetAttrInfo, API2:89
VNSetDirId, API2:91
VNSetName, API2:93
VNSetNumAttrs, API2:93
VNSetSize, API2:92
VNMaketable, API2:94
VNMovet, API2:91
VNNextChild, API2:89
VN_OPEN_ACCESS, API2:87
VN_OPEN_ATTR_FLAGS, API2:87
VN_OPEN_AT, API2:87
VNDirect, API2:92
VNSetRefCount, API2:95
VNRename, API2:90
VNSetAttrInfo, API2:94
VNSetSize, API2:93
VNSet, API2:92
VOL CACHE, API2:86
VOL_CAN_FLAGS, API2:87
VOL_COMMON, API2:87
VOL_DUPLICATE_MEDIA, API2:102
VOL_FORMAT_MEDIA, API2:100-101
VOL_FORMAT_MEDIA_INIT, API2:100
VOL_FORMAT_VOLUME, API2:116
VOL_INFO, API2:87
VOL_MEDIA_CAPACITIES, API2:100
VOL RTNS, API2:95
VOL_UPDATE_VOLUMES, API2:99
VOL_GODIR_CMN_ATTRS, API2:104
VOL_GODIR_INFO, API2:105
VOL_GODIR_RTNS, API2:113
VOL_GODIR_VNODE, API2:105
VOL_GODIR_VNODE_COMMON, API2:104
VOL_GODIR_VNODE_FLAGS, API2:104
VolSetVolName, API2:88
VolSpecificMsg, API2:88
VolStatus, API2:88
VolUpdateVolInfo, API2:88
Your comments on our software documentation are important to us. Is this manual useful to you? Does it meet your needs? If not, how can we make it better? Is there something we’re doing right and you want to see more of?

Make a copy of this form and let us know how you feel. You can also send us marked up pages. Along with your comments, please specify the name of the book and the page numbers of any specific comments.

Please indicate your previous programming experience:

- [ ] MS-DOS
- [ ] Mainframe
- [ ] Macintosh
- [ ] Minicomputer
- [ ] None
- [ ] Other _________

Please rate your answers to the following questions on a scale of 1 to 5:

<table>
<thead>
<tr>
<th>Question</th>
<th>1 Poor</th>
<th>2 Poor</th>
<th>3 Average</th>
<th>4 Good</th>
<th>5 Excellent</th>
</tr>
</thead>
<tbody>
<tr>
<td>How useful was this book?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Was information easy to find?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Was the organization clear?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Was the book technically accurate?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Were topics covered in enough detail?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Additional comments:

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________

Your name and address:

Name _________________________________________________________________

Company _____________________________________________________________

Address _____________________________________________________________

City ___________________________ State ___________ Zip ________

Mail this form to:

Team Manager, Developer Documentation
GO Corporation
919 E. Hillsdale Blvd., Suite 400
Foster City, CA 94404-2128

Or fax it to: (415) 345-9833
**PenPoint™ Application Programmatic Interface, Volume II**

Together with Volume I, *PenPoint™ Application Programmatic Interface, Volume II* provides a complete reference to the classes, messages, functions, and structures provided by the PenPoint Software Development Kit (SDK).

The parts in the *PenPoint API Reference* are organized in parallel with the parts in the *PenPoint Architectural Reference* (also available from Addison-Wesley). This volume contains datasheets on the APIs to the:

- Text subsystem
- PenPoint file system
- PenPoint operating system kernel
- Utility classes
- Connectivity classes
- PenPoint resource classes
- Installation interface
- PenPoint services
- Miscellaneous other classes

Other volumes in the GO Technical Library are:

*PenPoint Application Writing Guide* provides a tutorial on writing PenPoint applications, including many coding samples.  
*PenPoint User Interface Design Reference* describes the elements of the PenPoint Notebook User Interface, sets standards for using those elements, and describes how PenPoint uses the elements.  
*PenPoint Development Tools* describes the environment for developing, debugging, and testing PenPoint applications.  
*PenPoint Architectural Reference, Volume I* presents the concepts of the fundamental PenPoint classes.  
*PenPoint Architectural Reference, Volume II* presents the concepts of the supplemental PenPoint classes.  
*PenPoint API Reference, Volume I* provides a complete reference to the supplemental PenPoint classes, messages, and data structures.

**GO Corporation** was founded in September 1987 and is a leader in pen computing technology for mobile professionals. The company's mission is to expand the accessibility and utility of computers by establishing its pen-based operating system as a standard.

---

**GO Corporation**

919 East Hillsdale Blvd.  
Suite 400  
Foster City, CA 94404

Addison-Wesley Publishing Company